

SIEMENS EDA

Calibre® Query Server Manual

Software Version 2024.1
Document Revision 25

SIEMENS

Unpublished work.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 Calibre Query Server		19
Query Server Workflow.....		19
Input Databases		21
Query Server Database Modes.....		23
Devices.....		24
Nets		26
Syntax Conventions.....		26
Chapter 2		
Query Server Tcl Shell.....		29
calibre -qs.....		30
Tcl Basics.....		32
Query Server Tcl Shell Command Summary		34
Administrative Commands		35
CCI and Annotated Geometry Commands		35
Design Data Query Commands		35
Device Commands		37
LVS Custom Report Commands		37
LVS Rule File Query Commands		38
Net Commands		39
Netlisting Commands		40
Port Commands		41
Short Isolation Commands		41
Short Repair Database Commands		42
Soft Connection Checking Commands		44
Hcell List Generation Commands		44
Miscellaneous Commands		44
Summary of dfm:: Commands Used in the Tcl Shell		45
Command Reference Dictionary		48
qs::add_mask_polygon		49
qs::add_text		52
qs::close_svdb		55
qs::comparison_summary_is_available		56
qs::copy_svdb		57
qs::create_lvs_summary_report		58
qs::delete_mask_polygon		59
qs::delete_text		60
qs::device_table		62
qs::device_templates_used		63

qs::device_valid	64
qs::extraction_summary_is_available	65
qs::find_net_interactions	66
qs::find_path	67
qs::get_data	70
qs::get_devices_at_location	72
qs::get_nets_at_location	74
qs::get_placements_at_location	76
qs::get_run_summary_data	78
qs::get_shorts	82
qs::group_softchk_results	84
qs::help	87
qs::inc	88
qs::isolate_short	89
qs::isolate_shorts	93
qs::list_mask_polygon	96
qs::list_shorts	98
qs::list_text	99
qs::net_devicenames	101
qs::net_layers	102
qs::net_not_connected	103
qs::net_pins	105
qs::net_text_map	107
qs::net_trace	108
qs::net_valid	110
qs::open_svdb	111
qs::parse_path	113
qs::password	115
qs::perf_stats	116
qs::placement_valid	118
qs::print_shorts	119
qs::quit	120
qs::ran_erc	121
qs::ran_softchk	122
qs::rules_connectivity	123
qs::rules_file_name	124
qs::rules_layer_types	125
qs::rules_layout_case	126
qs::rules_lvs_compare_case	127
qs::rules_lvs_db_layer	128
qs::rules_lvs_db_connectivity_layer	129
qs::rules_lvs_downcase_device	130
qs::rules_lvs_ground_name	131
qs::rules_lvs_power_name	132
qs::rules_lvs_spice_replicate_devices	133
qs::rules_precision	134
qs::rules_source_case	135
qs::rules_unit_length	136
qs::softchk	137

Table of Contents

qs::softchks	140
qs::status	143
qs::write_lvs_comparison_summary_report_to_file	145
qs::write_lvs_extraction_summary_report_to_file	146
short_db::add_mask_polygon	147
short_db::add_text	150
short_db::assign_short	154
short_db::comment_short	156
short_db::delete_mask_polygon	157
short_db::delete_text	158
short_db::get_change_set	160
short_db::get_serial_number	161
short_db::get_short_path_endpoints	162
short_db::get_short_paths	163
short_db::get_status	164
short_db::goto_short	165
short_db::init_database	166
short_db::isolate_short	167
short_db::isolate_shorts	172
short_db::list_change_sets	176
short_db::list_shorts	177
short_db::list_text	178
short_db::mark_fixed	180
short_db::merge_short_revs	181
short_db::print_shorts	183
short_db::read_shorts_db	185
short_db::remove_change_set	187
short_db::rename_change_set	189
short_db::select_connects	190
short_db::set_distribution_configuration	193
short_db::set_status	196
short_db::show_change_set	197
short_db::softchk	199
short_db::switch_change_set	202
short_db::user_info	203
Using Short Database Commands	204
Using LVS Custom Report Commands	207
Chapter 3	
Hcell and Hierarchical Analysis	211
Current Hcell List	212
Evaluation of Hcells for LVS Comparison	213
Evaluation of Hcells for Netlist Extraction	216
Hcell Selection Limitations	217
Netlist Hcell Analysis and Reporting Flow	217
Tcl Shell Hcell Analysis Commands	219
hcells::add_matching_hcells	220
hcells::automatch	222

hcells::clear_hcell	224
hcells::clear_hcells	225
hcells::evaluate_current_hcells	226
hcells::expand_unbalanced	227
hcells::hcell	228
hcells::hcells	229
hcells::placementmatch	230
hcells::print_hcells	231
hcells::print_hierarchy	232
hcells::read	234
hcells::select	237
hcells::status	239
Hcell Evaluation Report Format	240
Hcell Analysis and Hierarchy Tree Report Format	242
Unbalanced Hcell Reporting	246
Cell Matches Using Placement Matching	246
Chapter 4	
Key Concepts for Standard Mode Operation	247
calibre -query	248
Acknowledgment Messages	252
Responses and Response Files	253
Command Context in the Standard Query Server	255
Client Contexts	257
Chapter 5	
Standard Query Commands	261
Command Format	261
Communication and Control Commands	263
ACTIVATE	264
CONNECT	265
DISCONNECT	266
ECHO	267
HELP COMMANDS	268
PASSWORD	269
QUIT	270
RESPONSE DIRECT	271
RESPONSE FILE	272
STOP ON	274
TERMINATE	275
Interrupts	275
Query Setup Commands	276
COMPARISON RESULT	277
CONTEXT	278
FILTER CULL	280
FILTER DEVICES	281
FILTER DEVICELAYERS	283
FILTER DISTANCE	284

Table of Contents

FILTER LAYERS	285
FILTER {IN OUT} LAYERS	286
FILTER WINDOW	288
MARKER SIZE	289
MAXIMUM VERTEX COUNT	290
STATUS	291
Results Transformation Commands	294
MAGNIFY RESULTS	296
REFLECTX RESULTS	298
ROTATE RESULTS	299
TRANSLATE RESULTS	300
Cell Query Commands	301
CELLS CORRESPONDING	302
CELL CORRESPONDING LAYOUT	303
CELL CORRESPONDING SOURCE	304
CELL TOP	305
CELLS LAYOUT	306
CELLS SOURCE	307
EXTENT	308
Browse Deviceless or Pseudo Cells Commands	309
BROWSE DEVICELESS CELLS	310
BROWSE PSEUDO CELLS	313
Device Query Commands	315
DEVICE BAD	316
DEVICE INFO	317
DEVICE LAYOUT	320
DEVICE LOCATION	321
DEVICE NAMES	322
DEVICE PINS	323
DEVICE SOURCE	325
DEVICE TABLE	326
DEVICE TEMPLATES USED	328
DEVICE VALID	329
Net Query Commands	330
NET BROWSE DEVICES	332
NET BROWSE NETS	334
NET BROWSE PORTS	336
NET BROWSE SHAPES	338
NET DEVICENAMES	340
NET EXTERNAL SHAPES	341
NET LAYERS	342
NET LAYOUT	343
NET LOCATION	344
NET NAMES	345
NET NOT CONNECTED	348
NET PINS	350
NET PORTNAMES	352
NET PORTS	353
NET SHAPES	355

NET SOURCE	357
NET TEXT MAP	359
NET TRACE	360
NET VALID	362
Placement Query Commands	363
PARSE PATH	364
PLACEMENT BROWSE DEVICES	367
PLACEMENT BROWSE NETS	368
PLACEMENT BROWSE PLACEMENTS	369
PLACEMENT BROWSE PORTS	370
PLACEMENT INFO	371
PLACEMENT LAYOUT	372
PLACEMENT LOCATION	373
PLACEMENT NAMES	375
PLACEMENT SOURCE	376
PLACEMENT TRANSFORM	377
PLACEMENT VALID	378
PLACEMENTS OF	379
Port Query Commands	380
PORT INFO	381
PORT LOCATION	383
PORT NAMES	384
PORT TEXT MAP	385
Rule File Query Commands	386
LVS SETTINGS REPORT WRITE	387
RULES CONNECTIVITY	391
RULES FILE NAME	393
RULES LAYER TYPES	394
RULES {LAYOUT SOURCE} CASE	395
RULES LAYOUT NETLIST	396
RULES LVS COMPARE CASE	397
RULES LVS DB LAYER	398
RULES LVS DB CONNECTIVITY LAYER	399
RULES LVS DOWNCASE DEVICE	400
RULES LVS {GROUND POWER} NAME	401
RULES LVS PDSP	402
RULES LVS REPORT	403
RULES LVS SPICE REPLICATE DEVICES	404
RULES {LAYOUT SOURCE} PATH	405
RULES PRECISION	406
RULES {LAYOUT SOURCE} SYSTEM	407
RULES SVDB DIRECTORY	408
RULES UNIT LENGTH	409
Hcell Analysis Commands	410
Hcell List Management Using Standard Commands	411
NETLIST ADD MATCHING HCELLS	413
NETLIST AUTOMATCH	414
NETLIST CLEAR HCELL	416
NETLIST CLEAR HCELLS	417

Table of Contents

NETLIST EVALUATE CURRENT HCELLS	418
NETLIST EVALUATION THRESHOLD	419
NETLIST EXPAND UNBALANCED	420
NETLIST HCELL	421
NETLIST HCELLS	422
NETLIST PLACEMENTMATCH.....	423
NETLIST READ	424
NETLIST REPORT HCELLS	426
NETLIST REPORT HIERARCHY	427
NETLIST SELECT HCELLS	428
NETLIST STATUS	429
Historical Hcell Reporting and Selection Scripts	429
Hierarchy Reporting	430
Generating Hcell and Xcell Lists	431
 Chapter 6	
Calibre Connectivity Interface	433
Tcl Shell CCI Command Reference	434
Conflicting Annotated Device Layers	435
qs::annotated_device_layer_map	436
qs::agf_map.....	438
qs::log_cci_commands	442
qs::port_table	444
qs::set_agf_options	448
qs::write_agf	453
qs::write_cell_extents	455
qs::write_layout_netlist_names	457
qs::write_lvs_settings_report	459
qs::write_separated_properties	462
Rule File Query Commands	462
ERC TVF Commands	464
erc::is_preflight_run	465
erc::setup_device_parameters.....	467
Stacked Transistor Handling in ERC TVF	473
Property Definitions and Netlist Behavior for CCI Flows	477
Generating CCI Output Files	480
CCI Generated Files Reference	485
Cross-Reference System File Formats	486
Placement Hierarchy (LPH/SPH) File Format.....	487
Instance and Net Cross-Reference File Formats	490
Customized SPICE Netlist Format.....	497
Layout Netlist Names File Format	502
Port Table File Format	504
Reduction Data File Format	506
Separated Properties File Format	509
Customized Files for Pushdown Separated Properties (PDSP) Flow	514
Writing PDSP Properties in CCI	515
Generating a Flat Netlist for a Specific Device	518

Viewing Connect and Device Layers	519
Standard Interface CCI Command Reference	520
Annotated Device Commands	521
ANNOTATED DEVICE LAYER MAP	522
FILTER {IN OUT} ANNOTATED LAYERS	524
Annotated Geometry Format Commands	525
Layer Output Control for AGF Files	527
AGF ANNOTATED DEVICES	529
AGF CLEAR FILTER BOX LAYERS	531
AGF CLEAR MERGE LAYERS	532
AGF CLEAR PCELL PROMOTE LAYERS	533
AGF FILTER BOX LAYERS	534
AGF FILTER {IN OUT} BOX LAYERS	536
AGF FILTER MAP	538
AGF FORMAT	539
AGF HIERARCHY	540
AGF HIERARCHY CLEAR EXPAND CELLS	541
AGF HIERARCHY EXPAND CELL	542
AGF HIERARCHY EXPAND DEVICELESS CELLS	543
AGF MAGNIFY USER UNITS	544
AGF MAP	545
AGF MERGE LAYER	547
AGF MERGE PIN LAYERS	549
AGF {DEVPROP NETPROP PLACEPROP} NAME	551
AGF {DEVPROP NETPROP PLACEPROP} NUMBER	553
AGF PCELL PROMOTE LAYERS	555
AGF RESET	556
AGF SEED PROPERTY	557
AGF SVRF MAP	560
AGF UNITS	562
AGF WRITE	563
Cell Extents Report Command	565
CELL EXTENTS WRITE	566
Customized Layout SPICE Netlist Commands	568
LAYOUT NETLIST ANNOTATED DEVICES	570
LAYOUT NETLIST CELL PREFIX	572
LAYOUT NETLIST NO CELL PREFIX	573
LAYOUT NETLIST COMMENT PROPERTIES	574
LAYOUT NETLIST DEVICE LOCATION	575
LAYOUT NETLIST DEVICE LOWERCASE	576
LAYOUT NETLIST DEVICE TEMPLATES	577
LAYOUT NETLIST EMPTY CELLS	579
LAYOUT NETLIST HIERARCHY	580
LAYOUT NETLIST HIERARCHY CLEAR EXPAND CELLS	581
LAYOUT NETLIST HIERARCHY EXPAND CELL	582
LAYOUT NETLIST HIERARCHY PREFIX	583
LAYOUT NETLIST HSPICE CR	585
LAYOUT NETLIST HSPICE USER	586
LAYOUT NETLIST NAMES	587

Table of Contents

LAYOUT NETLIST PIN LOCATIONS	589
LAYOUT NETLIST PORT PADS	591
LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS.....	593
LAYOUT NETLIST REPORT BAD ANNOTATED DEVICES.....	594
LAYOUT NETLIST RESET	595
LAYOUT NETLIST SEPARATED PROPERTIES	596
LAYOUT NETLIST STRING PROPERTIES	598
LAYOUT NETLIST TRIVIAL PINS	599
LAYOUT NETLIST UNIQUE NAMES.....	600
LAYOUT NETLIST WRITE.....	602
LAYOUT SEPARATED PROPERTIES WRITE	604
STATUS LAYOUT NETLIST	606
Layout Netlist Names File Command	607
LAYOUT NAMETABLE WRITE	608
Port Table File Commands	610
PORT TABLE WRITE	611
PORT TABLE NAME POLYGON PORTS	613
Reduction Data File Command.	614
REDUCTION DATA WRITE	616
Cross-Reference System File Commands.....	618
HIERARCHY SEPARATOR	619
HIERARCHY WRITE	620
INSTANCE XREF WRITE	622
LAYOUT NET XREF WRITE	624
NET XREF WRITE	626
XREF XNAME	628

Chapter 7	
Query Server Runtime Messages	631
Error and Warning Messages.....	631
Failure Messages	638
Note Messages	641

Index

Third-Party Information

List of Figures

Figure 1-1. Query Server Workflow	20
Figure 2-1. Mask Polygons	50
Figure 2-2. Masking Polygons with short_db::add_mask_polygon	148
Figure 4-1. Top Cell is A	256
Figure 4-2. Viewing and Query Cells are Both A	256
Figure 4-3. Viewing Cell is A, Query Cell is B, Query Instance is X2	257
Figure 4-4. Viewing and Query Cells are Both B	257
Figure 6-1. Series Stacked Transistors	473
Figure 6-2. Parallel Stacked Transistors	474
Figure 6-3. Resistor Cell Placed in TOP	529
Figure 6-4. Resistor Instance Promoted in AGF	530

List of Tables

Table 1-1. Steps for Using Query Server	20
Table 1-2. Syntax Conventions	26
Table 2-1. Tcl Special Characters	32
Table 2-2. Tcl Shell Command Types	34
Table 2-3. Administrative Commands	35
Table 2-4. Design Data Query Commands	36
Table 2-5. Device Commands	37
Table 2-6. LVS Custom Report Commands	37
Table 2-7. LVS Rule File Query Commands	38
Table 2-8. Net Commands	39
Table 2-9. Netlisting Commands	40
Table 2-10. Port Commands	41
Table 2-11. Short Isolation Commands	41
Table 2-12. Shorts Repair Database Commands	42
Table 2-13. Soft Connection Checking Commands	44
Table 2-14. Miscellaneous Commands	44
Table 2-15. Supported dfm:: Commands	45
Table 2-16. LVS Connectivity and Device Extraction Data Options	78
Table 2-17. ERC Data Retrieval Options	79
Table 2-18. Circuit Extraction Referenced Files Options	79
Table 2-19. LVS Comparison Data Retrieval Options	80
Table 2-20. LVS Comparison Referenced Files Options	80
Table 2-21. -by_cell and -by_layer Options	91
Table 2-22. -by_cell and -by_layer Options	169
Table 2-23. -by_cell and -by_layer Options	174
Table 3-1. Hcell Analysis Commands	219
Table 4-1. Client Table	258
Table 5-1. Communication and Control Commands	263
Table 5-2. Results Transformation Commands	294
Table 5-3. Commands Observing Results Transformations	294
Table 5-4. Cell Query Commands	301
Table 5-5. Device Query Commands	315
Table 5-6. Net Query Commands	330
Table 5-7. Cell Placement Query Commands	363
Table 5-8. Port Query Commands	380
Table 5-9. Rule File Query Commands	386
Table 5-10. Netlist and Hierarchy Analysis Commands	410
Table 6-1. Tcl Shell CCI Commands	434
Table 6-2. Files in output Directory	483
Table 6-3. Files in Current Directory	484

Table 6-4. PDSP Netlist Configuration Commands	514
Table 6-5. Annotated Device Commands	521
Table 6-6. AGF Commands	525
Table 6-7. Customized Layout SPICE Netlist Commands	568
Table 6-8. Cross-Reference System File Commands	618
Table 7-1. Error and Warning Messages	631
Table 7-2. Tcl Shell Runtime Messages	636
Table 7-3. Failure Messages	638

Chapter 1

Introduction to Calibre Query Server

Calibre® Query Server is an intermediary between databases in a Standard Verification Database (SVDB) and another client application, such as a script or user interface. While the Query Server is primarily intended for program-to-program communication, you can also use it from the command line to examine the contents of an SVDB.

The Query Server can provide information such as the following:

- A list of all cells or hierarchical cells (hcells) in a layout design.
- The location of a net or device.
- A list of all devices attached to a given net, and vice-versa.
- Cell, net, and device correspondences between the layout and source.
- Hcell list effectiveness.

In addition, the Query Server can produce customized mask layout and SPICE output for third-party flows through the [Calibre Connectivity Interface \(CCI\)](#).

The Query Server has two interface modes: Tcl shell and standard. The Tcl shell is the preferred one, as it receives greater development attention. The Tcl shell is also multithreaded while the standard interface is not. Each has capabilities that are unique to the interface in addition to capabilities they share.

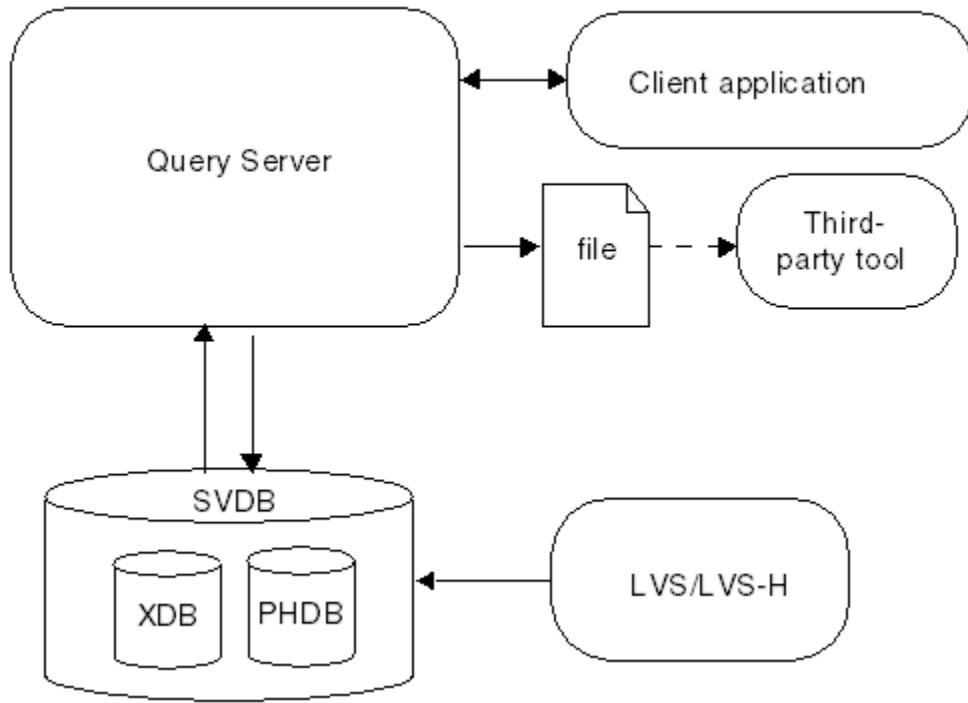
Query Server Workflow	19
Input Databases	21
Query Server Database Modes	23
Devices	24
Nets	26
Syntax Conventions	26

Query Server Workflow

The Query Server operates on databases contained in an SVDB (standard verification database). The Query Server has a variety of database outputs that allow for inspection of your designs, as well as for use in downstream simulation tools.

This figure shows the relationship between the tools and the information flow in LVS.

Figure 1-1. Query Server Workflow



The Query Server either works in Tcl shell mode (`calibre -qs`) or standard mode (`calibre -query`). The command sets are different for each mode.

An SVDB created by the rule file [Mask SVDB Directory](#) statement using the PHDB and/or XDB keywords is needed for many Query Server commands. The Mask SVDB Directory QUERY keyword subsumes both PHDB and XDB and is typically used. The CCI keyword is required for the Calibre Connectivity Interface set of commands.

The client application is typically a terminal shell. Many commands have output to STDOUT, while others have output to a specified file.

Using the Query Server in either Tcl shell or standard mode involves four basic steps.

Table 1-1. Steps for Using Query Server

Step	Description
1	Run Calibre nmLVS to create an SVDB database using the Mask SVDB Directory QUERY statement. Specify CCI if planning to use Calibre Connectivity Interface commands. Calibre PERC can also create an SVDB with PHDB connectivity data only (no XDB), and this SVDB can be used in the Query Server.
2	Start the Query Server and load the SVDB. Tcl shell: calibre -qs -svdb svdb Standard: calibre -query svdb

Table 1-1. Steps for Using Query Server (cont.)

Step	Description
3	Execute commands interactively in the shell. or Run a command script. Tcl shell: run calibre -qs -svdb svdb -exec qs.tcl or source Tcl script from interactive shell Standard: calibre -query svdb < query_script
4	Use outputs as needed.

It is also possible to run the Query Server using a “here” document in either run mode. Following is an example using the Query Server Tcl shell invoked from the C shell (works similarly in a Bourne shell):

```
% calibre -qs -svdb svdb << EOF
qs::status
qs::rules_file_name
...
qs::quit
EOF
```

Licensing

The Calibre Query Server license is required to run the Query Server. To run the Query Server Tcl Shell for short isolation, you also need the Calibre nmLVS and Calibre nmLVS-H licenses.

For more information on licensing, refer to “[Calibre Query Server](#)” in the *Calibre Administrator’s Guide*.

Input Databases

An SVDB directory contains the databases that the Query Server accesses.

The two most important SVDB files for the Query Server are the Persistent Hierarchical Database ([PHDB](#)) and the Cross-reference Database ([XDB](#)). Either or both may be present in the SVDB, depending on what you specify in the [Mask SVDB Directory](#) statement in the rule file. The PHDB contains information produced during the connectivity extraction (calibre -spice) phase. The XDB contains cross-reference information produced during the LVS comparison phase.

Note

 The SVDB can be generated during a flat LVS run. However, this will cause Query Server commands that expect design hierarchy to exist not to work. See “[Flat Database Mode](#)” on page 23.

SVDB databases may be password protected. If so, the application attempting to access them will prompt for a password.

PHDB

A PHDB (persistent hierarchical database) contains information about selected hierarchical shapes, connectivity, and extracted devices. A PHDB is stored in the SVDB when appropriate options are used with the Mask SVDB Directory statement.

The PHDB database contains the following:

- Complete text of the rule file(s) used for extraction.
- Environment variable settings used in the rule file.
- Cell and cell placement information (hierarchical LVS only).
- Summary extracted device information including device coordinates, types, calculated properties, and pin information.
- Summary extracted device information including device seed shape coordinates, types, calculated properties, pins, and pin location information.
- Net connectivity information through the hierarchy of cell placements, including pin and port information.
- Geometry information for database layers that appear in [Connect](#) and [Sconnect](#) operations, that serve as [Device](#) seed or pin layers, or that are target layers (second layer argument) of [Stamp](#) operations.
- Geometry information for database layers that appear in [LVS DB Layer](#) and [LVS DB Connectivity Layer](#) statements.

XDB

The XDB (Cross Reference Database) contains information generated during LVS comparison, where the XDB associates devices and nets with one another. This database contains these items:

- Net cross-reference for nets matched during LVS, including nets matched to multiple nets.
- Unmatched incorrect nets and indeterminate nets.
- Filtered instances or nets that are removed by the LVS Filter Unused family of statements.
- Instance cross-reference for instances matched during LVS, including smashed instances and gate membership information.
- Unmatched instances.

- Original netlist placement hierarchy information (for hierarchical LVS only).

Information from the PHDB and XDB is available independently. Some commands rely on both databases to generate results.

SVDB Database Compatibility

In general, SVDB databases created with earlier versions of Calibre are compatible with later versions of the Query Server.

In some cases, enhancements are made to the SVDB structure and the Query Server that make backward-compatibility impossible. In such cases where there are incompatibilities, you may need to create a new SVDB with a later version of Calibre. The Query Server issues a message when the SVDB is not compatible with the current Calibre version.

Query Server Database Modes

The databases available to the Query Server for probing define the mode in which the Query Server can operate. In each mode, a different set of commands can be used.

PHDB and XDB Only Modes

The Query Server normally provides information gathered during circuit extraction (PHDB) and LVS comparison (XDB).

If one or the other database is missing because of how the run was conducted, a subset of commands that require only information from the available database continue to work. For example, if a command finds the correspondence between a layout and a source object, but the XDB is not present, the command issues a discrepancy message.

Flat Database Mode

Flat mode LVS runs generate both PHDB and XDB databases. These databases are similar to a hierarchical run that has only one cell. However, they differ in that they follow the flat LVS instance-naming convention rather than the hierarchical one. All devices and nets appear in a single top-level cell. Commands which involve placement arguments, or involve multiple cells, do not run in flat mode.

The flat LVS naming methodology differs from the hierarchical methodology in the following ways:

- Device names from the PHDB are simply numbered, they do not have a SPICE element type letter. For example, a MOS device named M0 in the hierarchical PHDB would be named “0” in the flat PHDB.

- Since all shapes are flattened before extraction, no hierarchical names appear in a flat PHDB. All layout nets and device names appear as names or numbers in the top-level cell.
- Source names appear as flattened hierarchical names. They differ from their hierarchical counterparts in that subcircuit calls are represented with X characters in both devices and nets. For example, a net that is inside a cell placement in the hierarchical system has the name XAA/N21, while the same net in the flat system has the name AA/N21. Similarly, a device that has the hierarchical name XAA/MS0 has the flat name AA/MS0.

Note



Mixing of hierarchical extraction (calibre -spice) and flat comparison (calibre -lvs) is not recommended.

The Tcl shell mode (calibre -qs) does not process a flat database generated by a “calibre -lvs” run (no -hier and no -flatten options). The standard mode (calibre -query) can process a flat database generated this way, however. Nevertheless, running flat LVS without the -flatten option should be avoided. If the -flatten option is used, then the Query Server can open the generated database in either -qs or -query modes.

Devices

Devices in a Query Server database correspond to devices recognized during circuit extraction by using Device statements in the rule file. The set of recognized devices and the set of badly formed devices (if any) can be accessed by using various Query Server commands.

Information about devices can be accessed through the [dfm::get_device_data](#) command in the Tcl shell. The corresponding standard command is [DEVICE INFO](#).

A device table is frequently needed in a Calibre Connectivity Interface (CCI) flow. A device table consists of an indexed set of entries containing information about each device type in the design. There is one such table for the entire design, and it is independent of hierarchical

context. The [qs::device_table](#) and [DEVICE TABLE](#) commands produce a device table, which has this format:

```
Device_Table <precision>
Table Count
0 0 1 <date>
Device Entry <device number>
0 0 <number of list entries> <date>
<device_layer_name>
<device_type>
<element_name>
<model_name>
<netlist_element_name>
<netlist_model_name>
<pin_count>
<pin_info_N>
...
<auxiliary_layer_count>
<auxiliary_layer_N>
...
<property_count>
<property_name_N>
...
```

Pin and property names are lowercase. Layer names have the same text case as the rule file. Indices “N” range from 0 to $n-1$, where n is the total count of indexed elements. The device number corresponds to the Device statement in the rule file as the statements are used from start to end, indexed beginning with 0. In an extracted netlist, the device number is indicated by the “\$D” comment coded property.

Command responses that describe devices or device pins use the indices in the device table to reduce the size of the response. For example, in a response giving the device pins on a net, each pin is described by the following items:

device_path — pathname ending with device instance name.

device_number — 0-based index of the device entry in the device table.

pin_index — 0-based index of the pin in the pin lists of the device entry.

x y — coordinates of the pin in the current view cell.

A device type is one of the following constants: DIODE, RESISTOR, CAPACITOR, MOS, BIPOLAR, USER.

Additional information about devices in the table can be accessed through the [dfm::get_device_data](#) and [DEVICE INFO](#) commands mentioned previously.

Related Topics

[dfm::get_devices \[Calibre YieldServer Reference Manual\]](#)

[dfm::get_device_instances \[Calibre YieldServer Reference Manual\]](#)
[Device Query Commands](#)

Nets

Nets in an SVDB may be referenced by name or by number and correspond to the net IDs in an extracted layout netlist. Depending on the command, either the net name or the numeric ID is returned.

Nets may be specified as hierarchical paths. For example: X1/X2/3 would refer to net 3 in the instance path from X1 to X2. In the standard interface, this path is rooted in the current context cell.

Related Topics

[dfm::get_nets \[Calibre YieldServer Reference Manual\]](#)
[dfm::get_data \[Calibre YieldServer Reference Manual\]](#)
[Net Query Commands](#)

Syntax Conventions

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

Table 1-2. 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 1-2. 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.
	Vertical bars indicate an exclusive 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 {element_name [‘(‘model_name‘)’]}
```

```
device_layer {pin_layer [‘(‘pin_name‘)’] ...}
```

```
[‘<‘auxiliary_layer‘> ...]
```

```
[‘(‘swap_list‘)’ ...]
```

```
[BY NET | BY SHAPE]
```


Chapter 2

Query Server Tcl Shell

The Tcl shell is a component of the Query Server that is called by the calibre -qs command. The Tcl shell responds to Query Server Tcl commands, general Tcl commands, and certain shell commands from the host operating system. Customary Tcl conventions apply in the Tcl shell.

To use the Query Server Tcl shell, you must have the following:

- Calibre version 2008.3 or later, including an SVDB database.
- Calibre Query Server license. For short isolation, a Calibre nmLVS/Calibre nmLVS-H license pair is also needed. Refer to the *Calibre Administrator's Guide* for license information.

calibre -qs	30
Tcl Basics	32
Query Server Tcl Shell Command Summary	34
Command Reference Dictionary	48
Using Short Database Commands	204
Using LVS Custom Report Commands.....	207

calibre -qs

Calibre Query Server Tcl shell mode.

You can invoke the Tcl shell from your usual terminal session. You can issue commands directly at the command line prompt or in batch mode using a Tcl command file.

Usage

```
calibre -qs [-svdb svdb_dir [-rev revname] [top_cell]] [-turbo [n] [-turbo_all]] [-cb]  
[-exec tcl_script [args]]
```

Arguments

- **-qs**
A required argument that invokes the Tcl shell.
- **-svdb *svdb_dir***
An optional argument set that specifies a Mask SVDB Directory to open.
- **-rev *revname***
An optional argument set that specifies the SVDB revision to load. Revisions are generated using dfm::create_rev, dfm::save_rev, and short_db::merge_short_revs.
- ***top_cell***
An optional argument that indicates the top-level cell. This is required to prevent ambiguity when there is more than one top-level cell in the SVDB. To prevent ambiguity of the options, *top_cell* is also required when you specify -svdb with -exec.
- **-turbo [*n*]**
An optional argument set that enables multithreaded parallel processing for PHDB layers when [Mask SVDB Directory](#) SI, XRC, or XACT is used during circuit extraction. The value of *n* specifies the number of CPUs to use. If you do not specify *n*, the application uses the maximum number of CPUs on the machine.
- **-turbo_all**
An option used in conjunction with -turbo. When the number of CPUs specified with -turbo are not available, the tool normally runs with fewer threads than requested. This command line option causes the tool to exit if it cannot get the number of CPUs specified.
- **-cb**
Specifies to use the Calibre Cell/Block license. Refer to the [Calibre Administrator's Guide](#) for license information.
- **-exec *tcl_script* [*args*]**
An optional argument set used to pass an existing Tcl command script to the Query Server Tcl shell. The *tcl_script* is run, and then the Tcl shell exits. The optional *args* is a list of arguments that are passed to the *tcl_script*. The -exec argument set should be the final one on the command line to prevent other options from being interpreted as part of the *args* list.

If you use the `-svdb` option with `-exec`, then you must specify the `top_cell` argument also. This prevents “`-exec`” from being interpreted as the top cell name.

Description

Each Query Server Tcl session opens a database revision, performs queries or edits on the revision, and ultimately closes the same revision or saves to a different revision. A database revision is the collection of on-disk data for a specific session. By default, the command [qs::open_svdb -svdb svdb](#) opens an original PHDB database revision in read-only mode.

Each revision can be in either one of two states: frozen or non-frozen. A frozen revision cannot be modified but can serve as the starting point for new revisions. After opening a frozen revision (such as the original revision), you can create an editable (read or write) revision using `dfrm::create_rev`. Alternatively, a previously saved revision can be opened using [qs::open_svdb -svdb svdb_dir -rev revname](#).

To exit the Tcl shell, use [qs::quit](#). Alternatively, you can type “exit” (without quotation marks) or “exit -force” if you have made changes to the database.

Initialization File

You can specify an initialization file for the Tcl shell to source on startup. The shell searches for the initialization file in this order:

1. The `CALIBREQSRC` environment variable is set to the pathname of the file.
2. The current working directory contains the initialization file of the name `.calibrereqsrc`.
3. The user’s home directory contains the initialization file of the name `.calibrereqsrc`.

The first file found in the sequence is sourced by the Tcl shell.

Examples

Example 1

This shows the start of an interactive session:

```
% calibre -qs -svdb svdb
...
-----
----- CALIBRE::LVS QUERY SERVER -----
----- READING PHDB -----
#####
>
```

At this point, the Tcl shell is ready to receive commands.

Example 2

This command line executes commands in a file called *qs_script*:

```
calibre -qs -turbo -svdb svdb -exec qs_script
```

Related Topics

[calibre -query](#)

Tcl Basics

You should know some fundamental Tcl language ideas when using the Tcl shell.

- **Tcl is case sensitive.**

The core set of Tcl commands are all lowercase, as are the Query Server Tcl command extensions.

- **Order matters.**

The first word on a line is interpreted as a command name, and all other words on the line are command arguments. Commands are generally executed in the order they appear in a script. Tcl procedures (procs) generally need to be defined before they are called.

- **Certain characters have special meanings in Tcl.**

The following are the special characters you are most likely to encounter:

Table 2-1. Tcl Special Characters

Character	Meaning
;	Terminates the previous command, allowing you to place more than one command on the same line.
#	Indicates the start of a comment when it is the first non-whitespace character on a line or after at semicolon.
\	Causes backslash substitution. Normally this is used to remove the special meaning of a character. Be careful not to have whitespace on the same line after a backslash that terminates a line as this can cause Tcl interpretation errors.
\n	Creates a new line. There usually should be no space after the n.
\$	Instructs the Tcl interpreter to access the value stored in the variable name that follows.
[]	Instructs the Tcl interpreter to treat everything within the brackets as the result of a command. Command substitution can be nested within words.

Table 2-1. Tcl Special Characters (cont.)

Character	Meaning
{ }	Instructs the Tcl interpreter to treat the enclosed words as a list. The Tcl interpreter accepts the enclosed string as is, without performing any variable substitution or evaluation.
" "	Instructs the Tcl interpreter to treat the enclosed words as a single string. Variable and command substitution occur inside the quoted string.

- **Comments are a type of command.**

In Tcl, comments are indicated by a leading hash character (#). It must be the first non-whitespace character of the command; nothing after it on the line is acted upon.

Since the # must be the first character of the command, this means the first line in the following example is valid because it is preceded by a semicolon, but the third line is not, because the second line ends with a backslash.

```
} ;# End loop
qs::inc short_itr \
# "increment iterator"
```

- **Namespace considerations.**

Tcl commands are often prefixed with namespace of the form “name::”. However, “qs::”, “short_db::”, and “dfm::” cannot be imported.

Some YieldServer (dfm::) commands are enabled in the Query Server Tcl shell and access data in the PHDB. These are documented in the [Calibre YieldServer Reference Manual](#).

- **Iterators.**

Iterators are Tcl objects that can be thought of as opaque lists. The internal structure of an iterator is not generally predictable. Certain iterators can be incrementally stepped through using a loop to access all the values. In some cases, an iterator may store only one entry, in which case it cannot be incremented. It is important not to treat an iterator as a string by using it in a string-related context. It is generally a good practice to test if an iterator is empty, such as here:

```
while {$iterator ne ""} { ... }
```

Some other fundamental ideas may be found at <https://www.tcl-lang.org/about/language.html>.

Query Server Tcl Shell Command Summary

The Query Server Tcl shell provides a number of command types that are useful for accessing data in an SVDB directory.

Table 2-2. Tcl Shell Command Types

Command Types	Description
Administrative Commands	Control opening and closing of SVDBs, exiting, display of help messages, and performance statistics.
CCI and Annotated Geometry Commands	Perform functions similar to standard Calibre Connectivity Interface (CCI) commands.
Device Commands	Provide information about layout devices.
Design Data Query Commands	Provide information about the layout or source design.
dfm:: Commands	Calibre YieldServer command supported in the Tcl shell.
LVS Custom Report Commands	Control and write custom LVS Summary report.
LVS Rule File Query Commands	Return information about the LVS rule file.
Net Commands	Query information about layout nets.
Netlisting Commands	Read, configure, or write SPICE netlists
Port Commands	Provide information about layout ports.
Short Isolation Commands	Initiate short isolation, access short isolation data, or issue reports. In most cases, the Short Repair Database Commands (short_db::) should be used instead.
Short Repair Database Commands	Control aspects of short repairs database access, contents, and short isolation.
Soft Connection Checking Commands	Provide access to LVS Softchk functionality through the data stored in the PHDB.
Tcl Shell Hcell Analysis Commands	Control selection of hcells. The historical method for doing Hcell analysis using the non-Tcl Query Server is discussed under Hcell List Management Using Standard Commands .
Miscellaneous Commands	Provide access to various information.

Some commands in [Table 2-2](#) fulfill more than one role and appear as subsets of other command types.

Administrative Commands

The administrative commands provide high-level control of the Query Server Tcl shell and session information.

Table 2-3. Administrative Commands

Command	Description
<code>dfm::create_timer</code>	Returns a timer object.
<code>dfm::get_timer_data</code>	Returns time and memory use information from a timer object.
<code>dfm::reset_timer</code>	Restarts a timer object.
<code>qs::close_svdb</code>	Closes the database without quitting Query Server.
<code>qs::copy_svdb</code>	Creates a copy of an SVDB database.
<code>qs::exit</code>	Exits the Query Server Tcl shell (same as <code>qs::quit</code>).
<code>qs::help</code>	Displays a table of all Query Server Tcl shell commands.
<code>qs::open_svdb</code>	Opens an SVDB database.
<code>qs::password</code>	Sets a password for the current SVDB.
<code>qs::perf_stats</code>	Starts the performance monitor.
<code>qs::status</code>	Provides the state of server settings.
<code>qs::quit</code>	Exits the Query Server Tcl shell.

CCI and Annotated Geometry Commands

Calibre Connectivity Interface (CCI) functions, which include annotated geometry format (AGF) commands, are implemented in both the Calibre Query Server Tcl shell and in Calibre YieldServer.

The CCI commands are documented under “[Tcl Shell CCI Command Reference](#)” on page 434.

Design Data Query Commands

Design data query commands return information about the layout or source design and, in some cases, navigate design hierarchy. In certain cases, the commands return iterators, which are opaque objects referencing design elements.

Commands in the `dfm::` scope are documented in the *Calibre YieldServer Reference Manual*.

Table 2-4. Design Data Query Commands

Command	Description
<code>dfm::ascend_hierarchy</code>	Returns an iterator referencing placements in the parent cell of the current placement.
<code>dfm::ascend_path_context</code>	Moves a layout path context up from a placement to the parent cell.
<code>dfm::create_filter</code>	Returns a filter object used to constrain outputs of commands that use filters.
<code>dfm::descend_hierarchy</code>	Returns an iterator referencing placements in a cell placement.
<code>dfm::descend_path_context</code>	Moves a layout path context down into a placement.
<code>dfm::get_cells</code>	Returns a cell iterator.
<code>dfm::get_data</code>	Returns the specified type of data for the current database object.
<code>dfm::get_db_extent</code>	Returns the database extent as a Tcl list of lower-left and upper-right vertices as X and Y coordinates in database units.
<code>dfm::get_layout_name</code>	Returns corresponding layout information for source design objects.
<code>dfm::get_path_context</code>	Returns a hierarchical layout path context navigation object.
<code>dfm::get_placements</code>	Returns a layout placement iterator.
<code>dfm::get_source_name</code>	Returns corresponding source information for layout design objects.
<code>dfm::get_top_cell</code>	Returns the name of the layout top-level cell.
<code>dfm::get_xref_cells</code>	Returns an iterator that references cell names in a cross-reference database (XDB).
<code>dfm::get_xref_cell_data</code>	Returns information from a cell cross-reference iterator generated by <code>dfm::get_xref_cells</code> .
<code>dfm::inc</code>	Increments an iterator to point to the next entry in the sequence.
<code>qs::get_placements_at_location</code>	Returns the pathname and cell name of a placement at a specified location.
<code>qs::parse_path</code>	Returns cell names corresponding to instances in an instance pathname in a Tcl list of lists. For primitive devices, returns their Device type and subtype (if any). Net names that terminate a path are returned in a separate list.
<code>qs::placement_valid</code>	Indicates whether a layout or source placement path is valid.

Device Commands

Device commands provide information about layout devices.

Commands in the dfm:: scope are documented in the *Calibre YieldServer Reference Manual*.

Table 2-5. Device Commands

Command	Description
dfm::create_filter	Creates a filter object for dfm::get_device_data and qs::get_devices_at_location.
dfm::get_device_data	Returns details from a device iterator.
dfm::get_device_instances	Returns a device instance iterator.
dfm::get_devices	Returns a device iterator.
dfm::get_layout_name	Returns corresponding layout information for source design objects.
dfm::get_source_name	Returns corresponding source information for layout design objects.
qs::device_table	Returns a table of information about all devices in the layout design. This is similar to what the DEVICE TABLE command provides.
qs::device_templates_used	Returns a report of the Device statements actually used to generate instances in the extracted netlist.
qs::device_valid	Indicates whether a layout or source device path is valid.
qs::get_devices_at_location	Returns the pathname of a device placement at a specified location.

LVS Custom Report Commands

LVS Custom Report commands allow you to configure an LVS report.

A custom LVS report can be generated using a Tcl script and the commands listed in [Table 2-6. LVS Summary Report](#). [LVS Summary Report](#) must appear in the rule file. An example of these commands and an LVS Summary Report appears under “[Using LVS Custom Report Commands](#)” on page 207.

Table 2-6. LVS Custom Report Commands

Command	Description
qs::comparison_summary_is_available	Returns a 1 if comparison summary information is available.
qs::create_lvs_summary_report	Writes an LVS summary report to an output file.
qs::extraction_summary_is_available	Returns a 1 if extraction summary information is available.

Table 2-6. LVS Custom Report Commands (cont.)

Command	Description
<code>qs::get_run_summary_data</code>	Collects information for a custom LVS Summary Report.
<code>qs::ran_erc</code>	Returns a 1 if ERC has been run.
<code>qs::ran_softchk</code>	Returns a 1 if <code>SOFTCHK</code> has been run.
<code>qs::write_lvs_comparison_summary_report_to_file</code>	Writes a summary of comparison information to an output file.
<code>qs::write_lvs_extraction_summary_report_to_file</code>	Writes a summary of extraction information to an output file.

LVS Rule File Query Commands

The LVS rule file query commands return information about settings in the LVS rules. These commands are helpful in certain third-party tool flows that require information about the contents of the Standard Verification Database (SVDB).

Commands in the `dfm::` scope are documented in the *Calibre YieldServer Reference Manual*.

Table 2-7. LVS Rule File Query Commands

Command	Description
<code>dfm::get_layout_magnify</code>	Returns the value of the <code>Layout Magnify</code> setting.
<code>dfm::get_layout_system</code>	Returns the <code>Layout System</code> setting.
<code>dfm::get_layout_system2</code>	Returns the <code>Layout System2</code> setting.
<code>dfm::get_source_system</code>	Returns the <code>Source System</code> setting.
<code>qs::device_templates_used</code>	Returns a report of the <code>Device</code> statements actually used to generate instances in the extracted netlist.
<code>qs::rules_connectivity</code>	Returns rule file connectivity statement information.
<code>qs::rules_file_name</code>	Returns the name of the rule file used to create the SVDB database.
<code>qs::rules_layer_types</code>	Returns a Tcl list of lists of rule file layers and their types.
<code>qs::rules_layout_case</code>	Returns the <code>Layout Case</code> setting.
<code>qs::rules_lvs_compare_case</code>	Returns the <code>LVS Compare Case</code> setting.
<code>qs::rules_lvs_db_layer</code>	Returns the <code>LVS DB Layer</code> arguments.
<code>qs::rules_lvs_db_connectivity_layer</code>	Returns the <code>LVS DB Connectivity Layer</code> arguments.

Table 2-7. LVS Rule File Query Commands (cont.)

Command	Description
<code>qs::rules_lvs_downcase_device</code>	Returns the LVS Downcase Device setting.
<code>qs::rules_lvs_ground_name</code>	Returns the LVS Ground Name settings.
<code>qs::rules_lvs_power_name</code>	Returns the LVS Power Name settings.
<code>qs::rules_lvs_spice_replicate_devices</code>	Returns the LVS Spice Replicate Devices setting.
<code>qs::rules_precision</code>	Returns the Precision setting.
<code>qs::rules_source_case</code>	Returns the Source Case setting.
<code>qs::rules_unit_length</code>	Returns the Unit Length setting.
<code>qs::write_lvs_settings_report</code>	Returns a summary report of the LVS rule file settings for rule file query commands.

Net Commands

The net commands query information about layout nets.

Table 2-8. Net Commands

Command	Description
<code>dfm::get_layout_name</code>	Returns corresponding layout information for source design objects.
<code>dfm::get_source_name</code>	Returns corresponding source information for source design objects.
<code>dfm::get_nets</code>	Returns a net iterator.
<code>qs::find_net_interactions</code>	Writes a results database containing shapes external to a given cell that interact with a net in the cell.
<code>qs::get_nets_at_location</code>	Returns the pathname of a net placement at a specified location.
<code>qs::net_devicenames</code>	Returns a list of instance paths of all devices on a specified layout net.
<code>qs::net_layers</code>	Returns a list of layers that comprise a net.
<code>qs::net_not_connected</code>	Returns placements of a layout cell for which a specified net is not connected to a reference net.
<code>qs::net_pins</code>	Returns names and locations of intentional device pins on a layout net.

Table 2-8. Net Commands (cont.)

Command	Description
<code>qs::net_text_map</code>	Returns the mapping of layout net names to system-generated node numbers.
<code>qs::net_trace</code>	Returns the pathname of the highest representative of a net and all intermediate pathnames between the net and its highest representative.
<code>qs::net_valid</code>	Indicates whether a layout or source net path is valid.

Calibre nmLVS Reconnaissance ERC

Calibre nmLVS Recon ERC has one command used exclusively by it in a Tcl script: `qs::find_path`. This command finds a topological path of shapes between two points on different nets.

Netlisting Commands

Netlisting commands read, configure, or write SPICE netlists.

Commands in the `dfm::` scope are documented in the *Calibre YieldServer Reference Manual*.

Table 2-9. Netlisting Commands

Command	Description
<code>dfm::close_netlist</code>	Closes a netlist handle generated by <code>dfm::read_netlist</code> .
<code>dfm::list_layout_netlist_options</code>	Lists options that are set using <code>dfm::set_layout_netlist_options</code> .
<code>dfm::read_netlist</code>	Reads a SPICE netlist specified by Layout Path or Source Path in the rule file and configures the netlist transformations. Used with <code>dfm::write_spice_netlist</code> .
<code>dfm::set_layout_netlist_options</code>	Configures the <code>dfm::write_spice_netlist</code> command for writing a layout netlist from the PHDB.
<code>dfm::set_netlist_options</code>	Configures the output of <code>dfm::write_spice_netlist</code> from a netlist read by <code>dfm::read_netlist</code> . Controls how comment-coded properties are written.
<code>dfm::write_spice_netlist</code>	Writes a SPICE netlist. If used without a <code>dfm::read_netlist</code> handle, writes a SPICE netlist similar to calibre -spice based upon the PHDB.
<code>qs::write_layout_netlist_names</code>	Writes a Layout Netlist Names correspondence file.
<code>qs::write_separated_properties</code>	Writes push-down separated properties to a file.

Port Commands

Port commands provide information about layout ports.

Commands in the dfm:: scope are documented in the *Calibre YieldServer Reference Manual*.

Table 2-10. Port Commands

Command	Description
dfm::create_filter	Creates a filter object for dfm::get_ports -filter.
dfm::get_port_data	Returns information about a port. The information is similar to what the PORT INFO command provides.
dfm::get_ports	Returns an iterator or a Tcl list that describes the ports in the design.
qs::port_table	Returns a listing of information about cell ports in the layout design.

Short Isolation Commands

The short isolation commands provide access to Calibre short isolation functionality through the data stored in the PHDB.

The commands in this set facilitate short resolution, as do the short_db:: commands. The short_db:: commands are preferable in most circumstances. For more information on the shorts repair database commands, see “[Shorts Repair Database Commands](#)” on page 42.

Table 2-11. Short Isolation Commands

Command	Description
qs::add_mask_polygon	Suppresses short isolation on a specified layer.
qs::add_text	Adds a text object to a short iterator.
qs::delete_mask_polygon	Deletes mask polygons (created by qs::add_mask_polygon).
qs::delete_text	Deletes the specified text label (created by qs::add_text).
qs::get_data	Returns data from a short iterator.
qs::get_shorts	Returns an iterator to detected shorts.
qs::inc	Increments an iterator.
qs::isolate_short	Performs short isolation using an iterator.
qs::isolate_shorts	Isolates shorts from an SVDB database.
qs::list_mask_polygon	Lists the mask polygons in the system.
qs::list_shorts	Returns information about original short circuit elements as a Tcl list.
qs::list_text	Returns a Tcl list of text added by qs::add_text.

Table 2-11. Short Isolation Commands (cont.)

Command	Description
qs::print_shorts	Returns information about original short circuit elements.

Short Repair Database Commands

The short repair database commands manage the process of fixing shorts. To apply shorts database commands, the underlying PHDB database must have been created using the Mask SVDB Directory SI keyword.

These commands may be used in Calibre nmLVS Reconnaissance Short Isolation. Most of these commands are also used in Calibre nmLVS Recon Softchk scripts. The ones that are not are indicated in the table with footnote 1.

The change set family of commands allows for these changes to be played back in a different session so errors can be identified and fixed. A number of commands allow the user to amend and change text properties of a database for further short isolation runs. See “[Using Short Database Commands](#)” on page 204 for an example session with these commands.

The short database family of commands has equivalent qs:: commands. The short_db:: commands are generally easier to use than the qs:: equivalents and have more features in some cases.

Table 2-12. Shorts Repair Database Commands

Command	Description
short_db::add_mask_polygon	Suppresses short isolation on a specified layer.
short_db::add_text	Adds a text object to the short isolation database.
short_db::assign_short	Assigns or de-assigns a short to a specific user.
short_db::comment_short	Adds or returns comments on a short.
short_db::delete_mask_polygon	Delete mask polygons for current serial number.
short_db::delete_text	Deletes a text object in the short isolation database.
short_db::get_change_set	Returns the name of the current change set.
short_db::get_serial_number	Returns the current short's serial number.
short_db::get_short_path_endpoints	Returns endpoint data for a short path ID. ¹
short_db::get_short_paths	Returns a list of short path IDs. ¹
short_db::get_status	Returns a status name of a short.
short_db::goto_serial_number	This command is an alias for short_db::goto_short.
short_db::goto_short	Makes the short with a specified serial number the current short.

Table 2-12. Shorts Repair Database Commands (cont.)

Command	Description
<code>short_db::init_database</code>	Initializes an interactive short repairs database. ¹
<code>short_db::isolate_short</code>	Isolates a short in the shorts database. ¹
<code>short_db::isolate_shorts</code>	Finds shorts in the short isolation database. ¹
<code>short_db::list_change_sets</code>	Returns a list of all change sets.
<code>short_db::list_shorts</code>	Returns information about original short circuit elements as a Tcl list, including short serial numbers.
<code>short_db::list_text</code>	Returns a list of text labels in the shorts database.
<code>short_db::mark_fixed</code>	Marks the shorts associated with the serial number as “FIXED”.
<code>short_db::merge_short_revs</code>	Merges database revisions containing shorts into a single revision and sets it as the default. ¹
<code>short_db::print_shorts</code>	Returns information about original short circuit elements.
<code>short_db::read_shorts_db</code>	Reads a <i>.shorts</i> results database and makes the repair status and path endpoint data available to the current session. ¹
<code>short_db::remove_change_set</code>	Deletes the contents of a change set.
<code>short_db::rename_change_set</code>	Renames a change set.
<code>short_db::select_connects</code>	Selects Connect and Sconnect operations to be processed during stand-alone short isolation. This command is only valid when used in Calibre nmLVS Recon SI scripts. ¹
<code>short_db::set_distribution_configuration</code>	Configures runtime parameters for remote hosts in a stand-alone short isolation run that specifies a Query Server script. ¹
<code>short_db::set_status</code>	Assigns or removes a status label.
<code>short_db::show_change_set</code>	Reports the changes to a short that are stored in a change set.
<code>short_db::softchk</code>	Performs soft connection checks in a Calibre nmLVS Recon run on a specified short.
<code>short_db::switch_change_set</code>	Defines a new change set.
<code>short_db::user_info</code>	Shows or modifies user information for a short.

1. Not used in Calibre nmLVS Reconnaissance Softchk scripts.

Soft Connection Checking Commands

The soft connection checking commands access LVS Softchk data stored or perform soft connection checks on layer data stored in the PHDB.

Table 2-13. Soft Connection Checking Commands

Command	Description
qs::group_softchk_results	Groups softchk UPPER, CONTACT, and LOWER polygons that interact with a polygon on a given layer at a specified location. These grouped polygons are output to an ASCII RDB file. This facilitates easier understanding of shapes involved in a conflict. Used only in Calibre nmLVS Recon Softchk.
qs::ran_softchk	Returns a 1 if LVS Softchk has been run. Otherwise a 0 is returned.
qs::get_run_summary_data -softchk_database	Returns the name of the LVS Softchk results database.

Calibre nmLVS Reconnaissance Softchk

Calibre nmLVS Recon Softchk employs exclusive commands used in a Tcl script: [qs::softchk](#) and [qs::softchks](#). These are not used in the Query Server Tcl shell.

Most of the commands in the short_db:: scope may also be used in a -recon -softchk Tcl script, although some cannot. See “[Short Repair Database Commands](#)” on page 42.

Hcell List Generation Commands

Hcell lists can be generated using a subset of Tcl shell commands.

See “[Tcl Shell Hcell Analysis Commands](#)” on page 219.

Miscellaneous Commands

Commands in this category perform unique functions unrelated to other commands.

Table 2-14. Miscellaneous Commands

Command	Description
dfm::write_rdb	Writes specified layers to an ASCII RDB file.

Table 2-14. Miscellaneous Commands (cont.)

Command	Description
tvf::svrf_var	Returns the value of a variable defined with the Variable statement in the rule file. This command is permitted in the Query Server Tcl shell, Calibre YieldServer, and in the TVF runtime interface.

Summary of dfm:: Commands Used in the Tcl Shell

A subset of Calibre YieldServer commands are supported in the Query Server Tcl shell.

Commands in the dfm:: scope are documented in the *Calibre YieldServer Reference Manual*.

Table 2-15. Supported dfm:: Commands

Command	Description
dfm::ascend_hierarchy	Returns an iterator referencing placements in the parent cell of the current placement.
dfm::ascend_path_context	Moves a layout path context up from a placement to the parent cell.
dfm::close_netlist	Closes a netlist handle generated by dfm::read_netlist.
dfm::create_filter	(CCI command.) Returns a filter object used to constrain outputs of commands that use filters.
dfm::create_timer	Returns a timer object.
dfm::descend_hierarchy	Returns an iterator referencing placements in a cell placement.
dfm::descend_path_context	Moves a layout path context down into a placement.
dfm::get_cells	Returns a cell iterator.
dfm::get_connect_warnings	Returns a circuit extraction warning iterator.
dfm::get_data	Returns the specified type of data for the current database object.
dfm::get_db_extent	Returns the database extent as a Tcl list of lower-left and upper-right vertices as X and Y coordinates in database units.
dfm::get_device_data	Returns details from a device iterator.
dfm::get_device_instances	Returns a device instance iterator.
dfm::get_devices	Returns a device iterator.
dfm::get_device_pins	Returns a device pin iterator.
dfm::get_layout_magnify	Returns the value of the Layout Magnify setting.

Table 2-15. Supported dfm:: Commands (cont.)

Command	Description
dfm::get_layout_name	Returns corresponding layout information for source design objects.
dfm::get_layout_path	Returns the Layout Path setting in the rules.
dfm::get_layout_path2	Returns the Layout Path2 setting in the rules.
dfm::get_layout_system	Returns the Layout System setting in the rules.
dfm::get_layout_system2	Returns the Layout System2 setting in the rules.
dfm::get_nets	Returns a net iterator.
dfm::get_path_context	Returns a hierarchical layout path context navigation object.
dfm::get_placements	Returns a layout placement iterator.
dfm::get_port_data	Returns information about a port. The information is similar to what the PORT INFO command provides.
dfm::get_ports	Returns an iterator or a Tcl list that describes the ports in the design.
dfm::get_source_name	Returns corresponding source information for layout design objects.
dfm::get_source_path	Returns the Source Path setting in the XDB rules.
dfm::get_source_system	Returns the Source System setting in the XDB rules.
dfm::get_timer_data	Returns time and memory use information from a timer object.
dfm::get_top_cell	Returns the name of the layout top-level cell.
dfm::get_xref_cells	Returns an iterator that references cell names in a cross-reference database (XDB).
dfm::get_xref_cell_data	Returns information from a cell cross-reference iterator generated by dfm::get_xref_cells .
dfm::inc	Increments an iterator to point to the next entry in the sequence.
dfm::list_layout_netlist_options	Lists options that are set using dfm::set_layout_netlist_options .
dfm::read_netlist	Reads a SPICE netlist specified by Layout Path or Source Path in the rule file and configures the netlist transformations. Used with dfm::write_spice_netlist .
dfm::reset_timer	Restarts a timer object.
dfm::set_layout_netlist_options	(CCI command.) Configures the dfm::write_spice_netlist command for writing a layout netlist from the PHDB.

Table 2-15. Supported dfm:: Commands (cont.)

Command	Description
dfm::set_netlist_options	(CCI command.) Configures the output of dfm::write_spice_netlist from a netlist read by dfm::read_netlist. Controls how comment-coded properties are written.
dfm::write_ixf	(CCI command.) Writes an instance cross-reference (IXF) file.
dfm::write_lph	(CCI command.) Writes a layout placement hierarchy (LPH) file.
dfm::write_nxf	(CCI command.) Writes a net cross-reference (NXF) file.
dfm::write_rdb	Writes specified layers to an ASCII RDB file.
dfm::write_reduction_data	(CCI command.) Writes a file containing information about transformation reductions occurring in LVS comparison.
dfm::write_sph	(CCI command.) Writes a source placement hierarchy (SPH) file.
dfm::write_spice_netlist	(CCI command.) Writes a SPICE netlist. If used without a dfm::read_netlist handle, writes a SPICE netlist similar to calibre -spice based upon the PHDB.
dfm::xref_xname	(CCI command.) Configures the subcircuit call format of cross-reference file commands.

Command Reference Dictionary

You can issue the command `qs::help` in the shell to see a list of all the commands. Complete reference descriptions appear on the following pages.

This command provides help messages within the shell:

```
> qs::help
Query Server Commands:
  (For usage help on commands try <command name> -help)
```

The `dfm::` family of commands is documented in the [*Calibre YieldServer Reference Manual*](#).

qs::add_mask_polygon

Short isolation command. Using [short_db::add_mask_polygon](#) instead is generally preferred.

Suppresses short isolation on a specified layer.

Usage

```
qs::add_mask_polygon -layer name -it short_iterator
    -ec {x1 y1 x2 y2 ... [{x1 y1 x2 y2 ...}...]}
    [-cell name] [-dbu] [-top_coordinates] [-cell_coordinates] [-help]
```

Arguments

- **-layer *name***
A required argument and layer name. The layer must appear in a rule file [Connect](#) statement.
- **-it *short_iterator***
A required argument and short iterator.
- **-ec {*x1 y1 x2 y2* ... [{*x1 y1 x2 y2* ...}...]}**
A required argument and Tcl list of lists of polygon vertex coordinates. Coordinates are floating-point numbers in user units by default. At least two vertices must be specified. If only two vertices are specified, they are interpreted as the lower-left and upper-right vertices of a rectangle. Additional vertices may be specified to make polygons of any shape. Each sub-list specifies a distinct polygon.
- **-cell *name***
An optional argument set that specify a cell in which to isolate shorts.
- **-dbu**
An option that specifies database units are used. When this option is specified, coordinates are integers.
- **-top_coordinates**
An option that specifies the top-level coordinates are used. This is the default.
- **-cell_coordinates**
An option that specifies the cell-level coordinates are used. The cell of the current index location of the short iterator is used.
- **-help**
An option that shows the command usage.

Return Values

For successful attachment, the following is returned:

```
<layer> <cell> (polygons) <n> (vertices) <n>
```

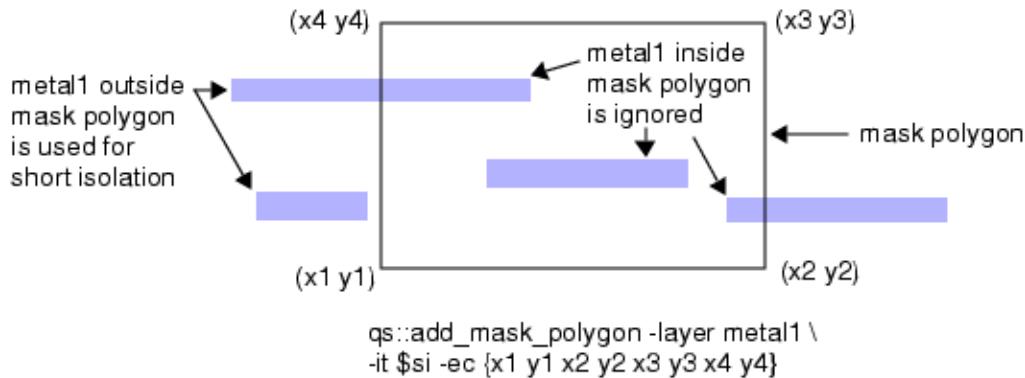
[Tcl Shell Runtime Messages](#).

Description

Creates mask polygons in memory that suppress short isolation on a specified layer and within the areas of the specified mask polygons.

The **-layer *name*** option specifies the layer to mask for short isolation. Any portion of a *name* layer that lies within the area of a mask polygon is ignored when using [qs::isolate_short](#). The layer specified using the *name* argument must appear in a Connect statement in the rule file for the layer to be masked for short isolation. Specifying a layer that is not in a Connect statement is allowed, but does not do anything useful. [Figure 2-1](#) illustrates how [qs::add_mask_polygon](#) works.

Figure 2-1. Mask Polygons



The **-it** argument specifies a short iterator in which to generate mask polygons.

The **-ec** argument specifies mask polygon vertex coordinates. At least one list of polygon vertex coordinates must be specified. Each coordinate list contains a set of x y values that specify vertices. At least two pairs of x and y coordinates must be specified for any vertex list.

If multiple mask polygons are specified and they touch or overlap, then they are merged into a single polygon. Subsequently, the command behaves as if the merged polygon had been specified.

The **-cell** option specifies the cell to mask for short isolation. When this option is used, the list of coordinates given by **-ec** must be within the coordinate space of the cell. The masked polygon is removed within all cell instances. This argument has no interaction with **-top_coordinates** or **-cell_coordinates**.

The **-dbu** option specifies that the given coordinates are in integral database units. When this option is not specified, all coordinates are floating-point numbers.

By default coordinates are in top-cell space and in user units. The -top_coordinates option explicitly specifies that top-cell space is used. The -cell_coordinates option specifies cell-space coordinates. The cell for cell space coordinates is determined by the current short in the iterator.

Examples

This example shows the creation of a short iterator and the addition of a mask polygon to the iterator. Then it shows an attempt to isolate a short where a mask polygon causes shorted polygons to be ignored.

```
> set si [qs::get_shorts -report_file shorts.rep]
Short Iterator

> qs::add_mask_polygon -it $si -layer metall1 -ec {{47000 302000 47000
271000 48000 271000 48000 312000}} -dbu
metall1 TOP (polygons) 1 (vertices) 4

> qs::isolate_short -it $si
(Masked out 1 original layer(s) ...)
    Short circuit (index: 0) (cell: a1220) (net_id: 1) (text: VDD) .
        VDD at (10,83)
        VSS at (10,1)
    GLOBAL HEURISTICS: HCC=8 FCC=2(2) HGC=89 FGC=89
    SHORT 0: HCC=1 FCC=1(1) HGC=28 FGC=28 VDD VSS (2/2)/1/1
    NOTE: Unable to isolate short - no short found.
    (Restored 1 original layer(s) ...)
```

Related Topics

[short_db::add_mask_polygon](#)

[qs::delete_mask_polygon](#)

[qs::list_mask_polygon](#)

[Short Isolation Commands](#)

qs::add_text

Short isolation command. Using [short_db::add_text](#) is generally preferred.

Adds a text object to a short iterator.

Usage

```
qs::add_text -it short_iterator -label text_label -layer name -pt {x y}  
[-top_coordinates | -cell_coordinates] [-dbu] [-help]
```

Arguments

- **-it *short_iterator***
A required argument and short iterator.
- **-label *text_label***
A required argument and a string that serves as a net name.
- **-layer *name***
A required argument and layer name (or number if the layer is drawn). See the following Description section for details about specifying this parameter.
- **-pt {*x* *y*}**
A required argument and list of two coordinate values. By default, the *x* and *y* parameters are floating-point numbers that specify a top-level location in user units.
- **-top_coordinates**
An option that specifies the top-level coordinates are used. This is the default.
- **-cell_coordinates**
An option that specifies the cell-level coordinates are used. The cell of the current index location of the short iterator is used.
- **-dbu**
An option that specifies integral database units are used.
- **-help**
An option that shows the command usage.

Return Values

For successful attachment, the following is returned:

```
<label> <x> <y> <layer>
```

For unsuccessful text label addition, a failure message is returned.

[Tcl Shell Error Messages](#).

Description

Adds a new text label to a short iterator at the specified location. A polygon to which the text label is attached must enclose the coordinates of the **-pt** argument list for the text attachment to be successful. If you attempt to add text to a location where there is no polygon, the command reports a failure to add the text. Top-level coordinates in user units are assumed by default.

The specified **-layer name** parameter determines the layer to which the specified text label is attached. The target layer for the text object must have connectivity information for the attachment to be successful. That is, the layer must appear in a **Connect** or **Sconnect** statement (but not after the BY argument), or it must be derived from such a layer by a series of node-preserving operations.

For the following discussion, assume these statements are in the rule file:

```
LAYER      diff 5
LAYER      poly 6
LAYER      metal1 10
LAYER      metal2 15
LAYER      txt 20

sd = diff NOT poly

CONNECT    diff poly metal1 BY contact
CONNECT    metal2 metal1 BY via

TEXT LAYER  txt
ATTACH     txt metal1
ATTACH     txt sd
```

The **name** parameter must be one of these layer types:

- It is the first argument to an **Attach** statement. The label is assigned to the second argument of the same Attach statement. For example:

```
qs::add_text -it $si -layer txt -label clk1 -pt {10.0 15.0}
```

This assigns the label clk1 to metal1 at the specified coordinates. The effect of this command is similar to putting a label on the text layer using the corresponding **Attach** statement in the rule file.

- It appears in a **Connect** statement, but is not a BY layer. For example:

```
qs::add_text -it $si -layer metal1 -label clk1 -pt {10.0 15.0}
qs::add_text -it $si -layer metal2 -label sig1 -pt {15.0 15.0}
```

Both of these commands have the effect of placing text objects on Connect layers.

- It is derived, has connectivity information, is not a **Connect** or **Sconnect** BY layer, and appears as the second argument to an **Attach** statement. The text is attached to the target layer of the **Attach** statement. For example:

```
qs::add_text -it $si -layer sd -label cap1 -pt {10.0 15.0}
```

This assigns the label cap1 to layer sd. The sd layer meets the four criteria.

This command is useful for debugging shorts. By placing new text objects in the *short_iterator*, this can assist in isolating the location of a short by reducing the geometric path between shorted text labels.

Examples

This example shows the creation of a short iterator and the addition of a text label to the iterator.

```
> set si [ qs::get_shorts -report_file shorts.rep ]
Short Iterator
> qs::add_text -it $si -label VSS -pt {23 42} -layer metal2
VSS 23.0 42.0 metal2
```

Related Topics

[short_db::add_text](#)

[qs::delete_text](#)

[qs::list_text](#)

[Short Isolation Commands](#)

qs::close_svdb

Administrative command.

Closes the database without quitting the Query Server.

Usage

qs::close_svdb [-force] [-help]

Arguments

- **-force**
An option that discards all changes from the current session.
- **-help**
An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Related Topics

[Administrative Commands](#)

qs::comparison_summary_is_available

LVS custom report command.

Returns a 1 if the LVS comparison summary information is available. Otherwise a 0 is returned.

Usage

qs::comparison_summary_is_available [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Integer.

[Tcl Shell Error Messages](#).

Examples

See “[Using LVS Custom Report Commands](#)” on page 207.

Related Topics

[LVS Custom Report Commands](#)

qs::copy_svdb

Administrative command.

Creates a copy of an SVDB database.

Usage

```
qs::copy_svdb -from source_path -to target_path [-help]
```

Arguments

- **-from source_path**

A required argument and pathname of an SVDB database to copy.

- **-to target_path**

A required argument and pathname of the target pathname of the copied SVDB.

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Description

Copies the specified SVDB (see [Mask SVDB Directory](#) in the *SVRF Manual* for more information) from the specified source directory to the specified target directory. This command is needed to copy an SVDB database because of its internal structure. The usual shell commands for copying or moving files do not work for copying an SVDB directory. However, the **tar** command does work properly for these directories.

Examples

```
>qs::copy_svdb -from svdb -to /user/scratch1/sandbox/new_svdb  
Read/Write db
```

Related Topics

[Administrative Commands](#)

qs::create_lvs_summary_report

LVS custom report command. The information created is identical to that created by the [LVS Summary Report](#) specification statement.

Writes an LVS summary report to an output file.

Usage

qs::create_lvs_summary_report *filename* [-help]

Arguments

- *filename*
A required pathname for the report output file.
- -help
An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Examples

See “[Using LVS Custom Report Commands](#)” on page 207.

Related Topics

[LVS Custom Report Commands](#)

[qs::delete_mask_polygon](#)

Short isolation command. Using [short_db::delete_mask_polygon](#) instead is generally preferred.
Deletes all mask polygons generated using the [qs::add_mask_polygon](#) command that exist in the specified iterator.

Usage

`qs::delete_mask_polygon -it short_iterator [-help]`

Arguments

- **-it *short_iterator***
A required argument and short iterator.
- **-help**
An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Related Topics

[short_db::delete_mask_polygon](#)

[qs::list_mask_polygon](#)

[qs::add_mask_polygon](#)

[Short Isolation Commands](#)

[qs::delete_text](#)

Short isolation command. Using [short_db::delete_text](#) instead is generally preferred.

Deletes a text label created in the short isolation session.

Usage

```
qs::delete_text -it short_iterator -label text_label
    [-pt {x y} | -top_coordinates | -cell_coordinates] [-dbu] [-help]
```

Arguments

- **-it *short_iterator***
A required argument and short iterator.
- **-label *text_label***
A required argument and string that serves as a net name. This label must have been added using the [qs::add_text](#).
- **-pt {*x* *y*}**
An optional argument and list of two coordinate values. By default, the *x* and *y* parameters are floating-point numbers that specify a top-level location in user units.
- **-top_coordinates**
An option that specifies the top-level coordinates are used. This is the default.
- **-cell_coordinates**
An option that specifies the cell-level coordinates are used. The cell of the current index location of the short iterator is used.
- **-dbu**
An option that specifies integral database units are used.
- **-help**
An option that shows the command usage.

Return Values

For successful global deletion of a label, “Deleted specified text label at all locations.” is returned. For successful deletion of a label at a location, the following is returned:

```
Deleted <x> <y>
```

For unsuccessful deletion of a label, “Failed to delete text. Text label does not exist.” is returned.

[Tcl Shell Error Messages](#).

Description

Deletes a text label that was added using [qs::add_text](#). The text label is deleted from the specified short iterator. This command does not delete text labels that originated from the layout.

If the -pt option is not used, then the command deletes all added text labels with the name specified by **-label *text_label***. If the -pt option is used, then only the added text label nearest the specified coordinates is deleted. Top-level coordinates in user units are used by default.

Text deletion actions are acknowledged by the command.

This command is useful for debugging shorts. By removing added text labels in the short iterator, this can assist in isolating the location of a short.

Related Topics

[short_db::delete_text](#)

[qs::list_text](#)

[qs::add_text](#)

[Short Isolation Commands](#)

qs::device_table

Design data query command. Corresponding standard command: [DEVICE TABLE](#). Also used in Calibre YieldServer.

Returns a table of information about all devices in the layout design.

Usage

`qs::device_table -write filename [-help]`

Arguments

- **-write *filename***
A required argument set that specifies to direct the output to the specified *filename*.
- **-help**
An option that shows the command usage.

Return Values

A table of information as described under “[Devices](#)” on page 24.

[Tcl Shell Error Messages](#).

Examples

The following is a device table with descriptive annotations.

```
Device_Table 1000          // precision is 1000 dbu/user unit
Table Count
0 0 1 Jun 1 19:35:29 2016 // timestamp
2                         // number of Device statements used for device
                          // classification
Device Entry 0            // first Device statement ($D=0 in netlist)
0 0 15 Jun 1 19:35:29 2016
ngate                      // seed layer
MOS                        // device type
mn                          // element name
(null)                     // model name (subtype)
(null)                     // Device NETLIST ELEMENT NAME specification
(null)                     // Device NETLIST MODEL NAME specification
4                           // pin count
g ngate 0                 // gate pin layer and swap list ordinal
s nsd 5                   // source pin layer and swap list ordinal
d nsd 5                   // drain pin layer and swap list ordinal
b pwell 0                 // bulk pin layer and swap list ordinal
0                           // auxiliary layer count
2                           // property count
w                           // first property name
1                           // second property name
```

Related Topics

[Device Commands](#)

qs::device_templates_used

Rule file query command. Corresponding standard command: [DEVICE TEMPLATES USED](#).

Returns lists of the instances formed during device recognition, along with associated statistics. Device types and subtypes that are not used to generate instances (as in LVS Box cells) are not output.

Usage

qs::device_templates_used [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

The following is the list format. The braces ({}) are literal.

```
{<type> [<subtype>] <$D> <HDC> <FDC>} // device type, optional subtype,  
... // DEVICE statement index number,  
... // hierarchical instance count, flat instance  
... // count.
```

[Tcl Shell Error Messages](#).

Examples

```
> qs::device_templates_used  
{mn n 0 24 195} {mp p 1 24 195}
```

Related Topics

[Device Commands](#)

[LVS Rule File Query Commands](#)

qs::device_valid

Design data query command. Also used in Calibre YieldServer. Corresponding standard command: [DEVICE VALID](#).

Indicates whether a layout or source device path is valid.

Usage

qs::device_valid *placement_path* {-layout | -source} [-cell {*name* | *iterator*}] [-help]

Arguments

- ***placement_path***

A required path of a device placement. By default, the path is relative to the top-level cell.

- **-layout**

Option that indicates a layout placement path is used. Either this option or **-source** must be specified. If this option is used, the persistent hierarchical database (PHDB) is queried first, followed by the cross-reference database (XDB) if available.

- **-source**

Option that indicates a source placement path is used. Either this option or **-layout** must be specified. If this option is used, then an XDB must exist.

- **-cell {*name* | *iterator*}**

An optional argument set that specifies the ***placement_path*** is relative to a given cell. When used with **-source**, the specified cell must have a corresponding layout cell as the query is actually performed on the layout design. The cell is specified as follows:

name — Cell name.

iterator — Cell iterator, such as is created by [dfm::get_cells](#).

- **-help**

An option that shows the command usage.

Return Values

A 1 (or true) if the path is valid and 0 (or false) otherwise.

[Tcl Shell Error Messages](#).

Examples

```
> # test using layout path from top level
> qs::device_valid x0/x0/m0 -layout
1
```

Related Topics

[Device Commands](#)

qs::extraction_summary_is_available

LVS custom report command.

Returns a 1 if connectivity extraction summary information is available and 0 otherwise.

Usage

qs::extraction_summary_is_available [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Integer.

[Tcl Shell Error Messages](#).

Examples

See “[Using LVS Custom Report Commands](#)” on page 207.

Related Topics

[LVS Custom Report Commands](#)

qs::find_net_interactions

Net command.

Writes a results database containing shapes external to a given cell that interact with a net in the cell.

Usage

```
qs::find_net_interactions -cell cell_name -net net_ID -file filename [-find_one] [-help]
```

Arguments

- **-cell *cell_name***
A required argument set that specifies the name of a layout cell.
- **-net *net_ID***
A required argument set that specifies a net in the cell corresponding to *cell_name*. The *net_ID* may be a name or node number.
- **-file *filename***
A required argument set that specifies the name of an output ASCII results database.
- **-find_one**
An option that specifies to output only the first polygon, if any, that interacts with the net.
- **-help**
An option that shows the command usage.

Return Values

Integer.

[Tcl Shell Error Messages](#).

Description

Returns a results database in ASCII RDB format that contains external shapes that interact with a given cell's net. The RDB may be loaded into Calibre RVE like a DRC results database. The number of interacting shapes is returned by the command.

This command is useful for finding shorts to nets higher up in the hierarchy than an instance of the specified cell.

Examples

Suppose your LVS Report shows there is an extra pin “2” on an instance of cell “A” in the extracted layout netlist. You can use this command to find the shape causing the short forming the extra pin:

```
qs::find_net_interactions -net 2 -cell A -file Cell_A.shorts.rdb
```

qs::find_path

Net command.

Finds a path of shapes connecting two points on different nets in a Calibre nmLVS Recon ERC run. The points are on specified layers. The path goes through all layers in a topological path connecting the points. The polygon results are output to an RDB file.

Usage

```
qs::find_path -from_layer layer_name -from_pt '{' x y '}'  
    -to_layer layer_name -to_pt '{' x y '}' -rdb filename  
    [-append] [-cell cell_name] [-dbu]  
    [-filter_devices seed_layer_list] [-filter_layers layer_list]  
    [-from_label label_name] [-to_label label_name] [-help]
```

Arguments

- **-from_layer *layer_name***

A required argument set that specifies the name of the layer on which a polygon containing the originating point of a path exists. This polygon must be on a different net than the one specified by **-to_layer**.

- **-from_pt '{' *x* *y* '}'**

A required argument set that specifies the coordinates of a point of a polygon on the **-from_layer** layer. By default, the *x* and *y* values are floating-point numbers in user units of length and in the coordinate space of the top-level cell. The coordinates must be specified as a Tcl list.

- **-to_layer *layer_name***

A required argument set that specifies the name of the layer on which a polygon containing the terminating point of a path exists. This polygon must be on a different net than the one specified by **-from_layer**.

- **-to_pt '{' *x* *y* '}'**

A required argument set that specifies the coordinates of a point of a polygon on the **-to_layer** layer. By default, the *x* and *y* values are floating-point numbers in user units of length and in the coordinate space of the top-level cell. The coordinates must be specified as a Tcl list.

- **-rdb *filename***

A required argument set that specifies the filename of an output ASCII results database.

- **-append**

An option that specifies results are appended to the file having the *filename*. If the *filename* does not exist, it is created as usual. By default, the RDB file is replaced if it exists.

- `-cell cell_name`

A optional argument set that specifies the path is found in the context of a cell having the `cell_name`. When this option is used, the `x` and `y` coordinates are in cell space. By default, the context is the top-level layout cell.

- `-dbu`

An optional argument that specifies the `x` and `y` coordinates are in database units. By default, they are in user units.

- `-filter_devices seed_layer_list`

An optional argument set that specifies a Tcl list of layers on which device seed shapes exist. A path found by the command does not contain devices on the specified seed layers.

- `-filter_layers layer_list`

An optional argument set that specifies a Tcl list of layers. A path found by the command does not contain devices on the specified layers.

- `-from_label label_name`

An optional argument set that defines a label for the start of the returned path. By default, the label for the starting point is `from_layer_NET_net_name`, where `from_layer` is the layer containing the polygon of the starting point, and `net_name` is the starting net.

- `-to_label label_name`

An optional argument set that defines a label for the end of the returned path. By default, the label for the terminating point is `to_layer_NET_net_name`, where `to_layer` is the layer containing the polygon of the terminating point, and `net_name` is the terminating net.

- `-help`

An option that shows the command usage.

Return Values

None.

Tcl Shell Error Messages

Description

This command can only be called from a calibre -recon run that uses the -erc and -exec options. See “[Calibre nmLVS Reconnaissance Command Line](#)” in the *Calibre Verification User’s Manual* for details on these run modes. The [Mask SVDB Directory](#) that is loaded during the -recon -erc run must be generated using the SI keyword.

Returns a results database in ASCII RDB format that contains shapes in a path between two specified points on two different nets. The RDB may be loaded into Calibre RVE like an ERC results database. The results are presented in a similar way as specified by the [LVS Isolate Shorts](#) BY CELL ALSO BY LAYER ALSO keywords, in addition to the primary-cell-context default.

The rule check names contain this string by default:

```
<from_layer>_Net_<id> - <to_layer>_Net_<id> in <cell>
```

where *from_layer* is the **-from_layer** argument name, *to_layer* is the **-to_layer** argument name, and *id* is a net name or node number. Here is an example rule check name:

```
metal1_Net_PWR - metal2_Net_5 in TOPCELL
```

The **-from_label** and **-to_label** options change the starting and ending label names, respectively.

If the **-from_point** and **-to_point** coordinates are on the same net, this causes an error.

Similarly, if the **-from_point** and **-to_point** coordinates do not intersect polygons on the **-from_layer** and **-to_layer**, respectively, this causes an error.

In general, there can be more than one path between the two specified endpoints. If there is more than one, then one of the paths is chosen arbitrarily for the output. The **-filter_devices** and **-filter_layers** options can exclude paths containing shapes from specified layers, thus limiting the set of possible paths for polygon selection.

If no path is found, the results database is empty.

Multiple instances of qs::find_path may access the same SVDB simultaneously.

Examples

Here is an example script called *recon_find_path*:

```
qs::find_path -from_layer metal1 -from_pt {2.2 57}\n              -to_layer    metal2 -to_pt     {41 34}\n              -rdb find_path.rdb
```

which can be executed as follows:

```
calibre -recon -erc -svdb svdb -turbo -exec recon_find_path
```

The *find_path.rdb* file can be opened in Calibre RVE as an ERC results database.

qs::get_data

Short isolation command.

Returns data from a short iterator.

Usage

```
qs::get_data -it short_iterator {-assigned_name | -cell_name | -index | -net_id | -object_type  
| -texts | -serial_number | -path_count | -path_list | -help}
```

Arguments

- **-it *iterator***
A required argument and short iterator created by [qs::get_shorts](#).
- **-assigned_name**
Returns the net name assigned to the short.
- **-cell_name**
Returns the name of the cell containing the short.
- **-index**
Returns the index number of the path of a short. Indices begin at 0.
- **-net_id**
Returns the numeric net ID of the shorted net.
- **-object_type**
Returns the type of iterator.
- **-texts**
Returns the text object names involved in the short.
- **-serial_number**
Returns the serial number of the short. Indexed from 0.
- **-path_count**
Returns the number of short polygon paths for the current iterator.
- **-path_list**
Returns the short path index numbers for the current iterator.
- **-help**
Returns the command usage.

Return Values

String related to the requested data.

[Tcl Shell Error Messages](#).

Description

Returns data associated with a short iterator. One data type request option must be specified.

Examples

This example shows the storing of short information in the iterator \$si, followed by requests for the net ID and net name assigned by the circuit extractor.

```
> set si [qs::get_shorts -report_file shorts.rep]
Short Iterator
> qs::get_data -it $si -net_id
2
> qs::get_data -it $si -assigned_name
A3
```

Related Topics

[Short Isolation Commands](#)

qs::get_devices_at_location

Device command. Corresponding standard command: [DEVICE LOCATION](#).

Returns the pathname of a device placement at a specified location. The specified query location is in top-cell space. The query includes all levels of hierarchy beneath the specified location. The cell context can be optionally changed to any cell. Devices that are queried can optionally be filtered by seed layer, device name, or device ID by specifying a filter object. Data coordinates can be transformed through the filter object also.

Usage

```
qs::get_devices_at_location x_coord y_coord [-cell {name | iterator}] [-filter filter_object]  
[-help]
```

Arguments

- ***x_coord***

A required x coordinate in user units.

- ***y_coord***

A required y coordinate in user units.

- **-cell {*name* | *iterator*}**

An optional argument set that specifies the query context is in a specified cell rather than in the top cell. When this option is used, coordinates are interpreted in the specified cell's space. The cell is specified as follows:

name — Cell name.

iterator — Cell iterator, such as is created by [dfm::get_cells](#).

- **-filter *filter_object***

An optional argument set that specifies a [dfm::create_filter](#) object. The object constrains the output to devices that meet the filter's criteria for seed layers, device names, or device IDs. The object can also contain geometric transformations, in which case the output coordinates conform to the transformations specified in the filter.

- **-help**

An option that shows the command usage.

Return Values

Tcl list.

[Tcl Shell Error Messages](#).

Examples

In this example, a qs::get_devices_at_location query without a filter gives a device path. Setting a filter object to constrain the query to MN devices causes no output, but setting the filter object to constrain the query to MP devices gives the original device path.

```
> qs::get_devices_at_location 6.1 58.4
X0/X0/M2
> set f [dfm::create_filter -device_names MN]
Filter
> qs::get_devices_at_location 6.1 58.4 -filter $f
(no output)
> set f [dfm::create_filter -device_names MP]
Filter
> qs::get_devices_at_location 6.1 58.4 -filter $f
X0/X0/M2
```

Related Topics

[Device Commands](#)

[qs::get_nets_at_location](#)

Net command. Corresponding standard command: [NET LOCATION](#).

Returns the pathname of a net placement at a specified location. The specified query location is in top-cell space. The cell context can be optionally changed to any cell. The query includes all levels of hierarchy beneath the specified location. If a net has a name, it is returned as the terminal element of the pathname; otherwise, node IDs are returned. Nets that are queried can optionally be filtered by layer by specifying a filter object. Data coordinates can be transformed through a filter object also.

Usage

```
qs::get_nets_at_location x_coord y_coord [-cell {name | iterator}] [-filter filter_object]  
[-help]
```

Arguments

- ***x_coord***

A required x coordinate in user units.

- ***y_coord***

A required y coordinate in user units.

- **-cell {*name* | *iterator*}**

An optional argument set that specifies the query context is in a specified cell rather than in the top cell. When this option is used, coordinates are interpreted in the specified cell's space. The cell is specified as follows:

name — Cell name.

iterator — Cell iterator, such as is created by [dfm::get_cells](#).

- **-filter *filter_object***

An optional argument set that specifies a [dfm::create_filter](#) object. The object can contain a layer filter, in which case the output is constrained to layers specified in the filter. The object can also contain geometric transformations, in which case the output coordinates conform to the transformations specified in the filter.

- **-help**

An option that shows the command usage.

Return Values

Tcl list.

[Tcl Shell Error Messages](#).

Examples

```
> qs::get_nets_at_location 2 84  
X0/X0/PWR PWR
```

Related Topics

[Net Commands](#)

qs::get_placements_at_location

Design data query command. Corresponding standard command: [PLACEMENT LOCATION](#).

Returns the pathname and cell name of a placement at a specified location. By default, the specified query location is in top-cell space and at the top level of the hierarchy. The cell context can be optionally changed to any cell, and the hierarchy can be treated as flattened. Data coordinates can be transformed through a filter object. Data is returned in a Tcl list.

Usage

```
qs::get_placements_at_location x_coord y_coord [-cell {name | iterator}]
                                [-filter filter_object] [-flat] [-help]
```

Arguments

- ***x_coord***

A required x coordinate in user units.

- ***y_coord***

A required y coordinate in user units.

- **-cell {*name* | *iterator*}**

An optional argument set that specifies the query context is in a specified cell rather than in the top cell. When this option is used, coordinates are interpreted in the specified cell's space. The cell is specified as follows:

name — Cell name.

iterator — Cell iterator, such as is created by [dfm::get_cells](#).

- **-filter *filter_object***

An optional argument set that specifies a filter object produced by [dfm::create_filter](#). The output coordinates conform to any geometric transformations specified in the filter.

- **-flat**

An option that specifies to return placement paths from all levels of hierarchy beneath the specified location.

- **-help**

An option that shows the command usage.

Return Values

Tcl list, or list of lists if -flat is used.

[Tcl Shell Error Messages](#).

Examples

```
> qs::get_placements_at_location 2 80
X0 CellA
> qs::get_placements_at_location 2 80 -flat
{X0 CellA} {X0/X0 nand}
```

Related Topics

[Design Data Query Commands](#)

[qs::get_run_summary_data](#)

Command type: LVS custom reports

Collects information for a custom LVS Summary Report. Gets the information to create a custom LVS Summary Report. Time data is presented in seconds since the last epoch. If data is not available, the server returns an empty string or 0.

Usage

`qs::get_run_summary_data option [-help]`

Arguments

- *option*

Choose one option from one of the following tables:

- LVS Connectivity and Device Extraction Data Options — [Table 2-16](#)
- ERC Data Retrieval Options — [Table 2-17](#) on page 79
- Circuit Extraction Referenced Files Options — [Table 2-18](#) on page 79
- LVS Comparison Data Retrieval Options — [Table 2-19](#) on page 80
- LVS Comparison Referenced Files Options — [Table 2-20](#) on page 80

Table 2-16. LVS Connectivity and Device Extraction Data Options

Option	Description
-ambiguous_name_pad_count	Number of ambiguous name pads detected.
-bad_device_count	Number of bad devices identified.
-ext_box_cell_count	Number of LVS Box cells present during extraction.
-ext_end_time	End time in seconds since the epoch.
-ext_rundir	Run directory during connectivity extraction.
-ext_start_time	Start time in seconds since the epoch.
-global_name_conflict_count	Number of global name conflicts detected.
-open_circuit_count	Number of open circuits detected.
-sconnect_conflict_count	Number of Sconnect conflicts detected.
-short_circuit_count	Number of short circuits detected.
-trivial_port_pads_reported	Number of trivial port pads reported, when LVS Report Trivial Ports NO is used.
-unattached_name_pad_count	Number of unattached texts detected.
-unattached_port_pad_count	Number of unattached ports detected.

Table 2-16. LVS Connectivity and Device Extraction Data Options (cont.)

Option	Description
-virtual_connect_count	Number of virtual connections created.

Table 2-17. ERC Data Retrieval Options

Option	Description
-dfm_rdb_results_printed	Total number of DFM RDB results printed.
-erc_flat_pathchk_polygon_result_count	Total number of Pathchk polygon results generated (flat).
-erc_nets_printed	Total number of ERC nets printed.
-erc_partial_results	Returns 1 if ERC output may be incomplete due to insufficient memory.
-erc_pathchk_count	Total number of statements executed.
-erc_pathchk_net_result_count	Total number of Pathchk net results generated.
-erc_pathchk_polygon_result_count	Total number of Pathchk polygon results generated (hierarchical).
-erc_polygons_printed	Total number of ERC polygons printed.
-erc_supply_problem	Returns 1 if ERC supply problems were encountered.
-erc_total_check_count	Number of ERC checks executed.
-erc_total_flat_result_count	Total number of ERC results generated (flat).
-erc_total_result_count	Total number of ERC results generated (hierarchical).

Table 2-18. Circuit Extraction Referenced Files Options

Option	Description
-dfm_rdb_files	File pathnames of DFM RDB results files.
-erc_database	File pathname of the ERC Results Database .
-erc_pathchk_database	File pathname of the ERC Pathchk results database.
-erc_pathchk_report	File pathname of ERC Pathchk results report.
-erc_summary	File pathname of the ERC Summary Report .
-ext_report	File pathname of connectivity extraction report.

Table 2-18. Circuit Extraction Referenced Files Options (cont.)

Option	Description
-ext_rules	File pathnames of the rule files used for extraction.
-layout_netlist	Layout netlist written during extraction.
-pathchk_net_files	File pathnames of Pathchk net results files.
-pathchk_poly_files	File pathnames of Pathchk polygon results files.
-seed_promotion_database	File pathnames of seed promotion database.
-shorts_database	File pathname of the LVS Short Isolation database.
-softchk_database	File pathname of the LVS Softchk results database.
-waiver_rdb_files	File pathnames of waiver RDB results files.

Table 2-19. LVS Comparison Data Retrieval Options

Option	Description
-lvs_compare_box_cell_count	Number of LVS Box cells used during comparison.
-lvs_compare_end_time	End time in seconds since the epoch.
-lvs_compare_error_types	Number of distinct types of LVS error statuses encountered.
-lvs_compare_input_error_count	Number of LVS input errors encountered.
-lvs_compare_input_warning_count	Number of LVS input warnings encountered.
-lvs_compare_warning_types	Number of distinct types of LVS warning statuses encountered.
-lvs_compare_start_time	Start time in seconds since the epoch.
-lvs_compare_status	LVS comparison status.
-lvs_rundir	LVS run directory.

Table 2-20. LVS Comparison Referenced Files Options

Option	Description
-lvs_report	File pathname of the LVS comparison report.
-lvs_rules	File pathname of the LVS comparison rule file.
-lvs_summary_report	File pathname of the LVS Summary Report file.
-source_netlist	File pathname of the Source netlist.

- `-help`

An option that shows the command usage.

Return Values

String.

[Tcl Shell Error Messages](#).

Examples

See “[Using LVS Custom Report Commands](#)” on page 207.

Related Topics

[LVS Custom Report Commands](#)

qs::get_shorts

Short isolation command.

Creates a short iterator used for input to other commands.

Usage

`qs::get_shorts -report_file filename [-help]`

Arguments

- **-report_file *filename***

A required argument and pathname of a report file that lists short information. This file is created when [qs::isolate_short](#) is run on an iterator generated by the qs::get_shorts command.

- **-help**

An option that shows the command usage.

Return Values

Short Iterator response to the shell and a short iterator.

[Tcl Shell Error Messages](#).

Description

Returns an iterator that contains short information. The iterator is expected to be stored in a variable. The shorts iterator can be passed to [qs::isolate_short](#). It can be stepped forward using the [qs::inc](#) command.

If you store two different iterators using this command in the same session, the first iterator is deleted and the second is maintained.

This command also reserves the name of a report file. The report file is written when [qs::isolate_short](#) is run on the shorts iterator. The format of the file is identical to the file generated by the [LVS Isolate Shorts](#) YES statement in the rule file.

Examples

This example shows the storing of short information in the iterator \$si, followed by isolation of the first short in the iterator.

```
> set si [qs::get_shorts -report_file shorts.rep]
Short Iterator
> qs::isolate_short -it $si
Short circuit (index: 0) (cell: TOP) (net_id: 1) (text: PWR).
    GND at (13.5,217)
    PWR at (442,217.5)
GLOBAL HEURISTICS: HCC=9 FCC=65 (58) HGC=954 FGC=4683
```

Related Topics

[Short Isolation Commands](#)

qs::group_softchk_results

Soft connection checking command.

Groups softchk UPPER, CONTACT, or LOWER polygons that interact with a polygon on a given layer at a specified location. These grouped polygons are output to an ASCII RDB file. This facilitates easier understanding of shapes involved in a conflict.

Usage

```
qs::group_softchk_results x y -layer layer_name -file filename [-append] [-cell name] [-dbu]
  [{-by_vertex_count [-lower] | -upper | -contact] [-all]} |
   {-by_contact_count -upper_layer upper_layer}] [-help]
```

Arguments

- ***x y***
A required pair of coordinates of a point used to select a softchk stamped layer (the second layer in Sconnect) polygon that the point intersects. This polygon is used for grouping other shapes from a softchk result. By default the coordinates are in top-level space with dimensions in user units of length.
- ***-layer layer_name***
A required argument set that specifies the layer of the polygon selected by the *x* and *y* coordinates. This layer is a softchk stamped layer.
- ***-file filename***
A required argument set that specifies the name of an ASCII results database. The database is similar to one produced by [LVS Softchk](#). See “[LVS Softchk Results Database File Format](#)” in the *Calibre Verification User’s Manual* for related information.
- ***-append***
An option that specifies results are appended to an existing results database. The default is to replace an existing results database. If the command is specified multiple times in a single run, this option is especially useful in the `qs::group_softchk_results` commands after the initial one.
- ***-cell name***
An optional argument set that specifies the cell space for the *x* and *y* coordinates. By default, the *name* is from the primary cell.
- ***-dbu***
An option that specifies the coordinates are in database units. By default, they are user units.
- ***-by_vertex_count* [-lower] | -upper | -contact] [-all]**
An optional argument set that specifies the prevailing net in a stamping conflict is selected based upon the total vertex count of polygons on a net. This is the default behavior and the same as the LVS Softchk default. May not be specified with `-by_contact_count`.

For the following definitions, the term “selected polygon” means the one under the specified *x y* coordinates. The following options control which layers are output:

- lower — Specifies that stamped layer (the second layer in Sconnect) shapes interacting with the selected polygon are output. This is the default.
- upper — Specifies that stamping layer (the first layer in Sconnect) shapes interacting with the selected polygon are output.
- contact — Specifies that contact or via layer (the Sconnect BY layer) shapes interacting with the selected polygon are output.
- all — Specifies that all nodal layers interacting with the selected polygon are eligible for output. By default, only layers from discarded nets are eligible. This option has no effect when -lower is used because a stamped polygon is already included in the output.
- -by_contact_count -upper_layer *upper_layer*
An optional argument set that specifies the command behaves like [lvs::softchk](#). The *upper_layer* is the stamping layer in an Sconnect conflict and must appear in a Connect statement, but not as a contact (BY) layer.
- -help
An option that shows the command usage.

Return Values

None.

[Tcl Shell Runtime Messages](#).

Description

This command can only be used in a Tcl script defined in the Calibre nmLVS Recon Softchk command line (-recon -softchk=*tcl_script*).

This command groups together polygons that are involved in an [Sconnect](#) conflict with a stamped layer shape intersected by the point defined by the *x* and *y* coordinates. The cell placement containing that softchk results shape is the context for the results output. The group of shapes is output to an ASCII results database for use with Calibre RVE. This facilitates easier debugging of softchk conflicts.

The *layer_name* frequently is the LOWER layer from an Sconnect operation.

The -by_vertex_count and -by_contact options have similar effects as in [qs::softchk](#).

The database format is discussed under “[LVS Softchk Results Database File Format](#)” in the *Calibre Verification User’s Manual*. The database can be used in Calibre RVE for LVS. Also see “[LVS Softchk Debug Method](#)” in the same manual.

Examples

Assume this rule file excerpt:

```
MASK SVDB DIRECTORY svdb SI  
  
CONNECT ptap metal1 BY contact  
SCONNECT ptap pwell
```

These commands perform LVS Softchk verification in a calibre -recon -softchk=*tcl_script* run on layers in the PHDB:

```
# find sconnect violations on pwell  
qs::softchk -lower_layer pwell -file pwell.softchk  
  
# append upper shapes involved with particular sconnect conflict  
qs::group_softchk_results 0.5 10 -layer pwell -append \  
-file pwell.softchk -by_vertex_count -upper
```

The soft connections are processed as with the LVS Softchk LOWER keyword, which is the default. UPPER shapes are then appended to the output in the *pwell.softchk* results database, which can be loaded into Calibre RVE for layout debugging.

Related Topics

[Soft Connection Checking Commands](#)

qs::help

Administrative command. Corresponding standard command: [HELP COMMANDS](#).

Displays a table of all Query Server Tcl shell commands.

Usage

qs::help

Arguments

None.

Return Values

List of commands and descriptions.

Examples

```
> qs::help
Query Server Commands:
  (For usage help on commands try "qs::cmd -help" or "qs::help command
  qs::cmd")
complete list of commands follows
```

Related Topics

[Administrative Commands](#)

qs::inc

Design data query command.

Increments an iterator to point to the next entry in the sequence.

Usage

`qs::inc iterator_name [-help]`

Arguments

- *iterator_name*

A required argument that must be an iterator name.

The *iterator_name* should not use the \$ character as when referencing a variable.

- -help

An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Examples

This example shows the creation of a short iterator and a loop to isolate the shorts in the iterator. The results are written to the *shorts.rep* file.

```
> set si [qs::get_shorts -report_file shorts.rep]
Short Iterator
> while {$si ne ""} {
    qs::isolate_short -it $si
    qs::inc si
}
```

Related Topics

[Short Isolation Commands](#)

qs::isolate_short

Short isolation and iterator control command. Using [short_db::isolate_short](#) instead is generally preferred.

Performs short isolation using an iterator.

Usage

```
qs::isolate_short -it short_iterator
[-by_cell {also | no | yes}]
[-by_layer {also | no | yes}]
[-flat {no | yes | if_fast}]
[[-inverse] -name "text_name ..."]
[-help]
```

Arguments

- **-it *short_iterator***

A required argument and short iterator created by [qs::get_shorts](#).

- **-by_cell { also | no | yes }**

An optional argument set that specifies whether to report the cell of origin for short polygons. When this option is specified, one of these arguments must be present:

also — Shorts are reported both fully merged and in the cell of origin for each layer comprising the short. This is the default behavior if this option is not specified.

no — Short polygons are not reported in the cell of origin.

yes — Shorts are reported in the cell of origin for each layer comprising the short.

- **-by_layer { also | no | yes }**

An optional argument set that specifies whether to report separate results per short, per layer. When this option is specified, one of these arguments must be present:

also — Shorts are reported both fully merged and are reported with separate layer information. This is the default behavior if this option is not specified.

no — Shorts are not reported with layer information.

yes — Shorts are reported with separate layer information.

- **-flat { no | yes | if_fast }**

An optional argument set that specifies whether to perform the short isolation flat. When this option is specified, one of these arguments must be present:

no — Short isolation is performed hierarchically. This is the default behavior if this option is not specified.

yes — Short isolation is performed flat.

if_fast — Short isolation is performed flat if it will be faster than performing it hierarchically.

If the “yes” or “if _fast” (and flat isolation occurs) options are used, then all -by_cell “yes” or “also” results are reported at the top level.

- **-name “*text_name* ...”**

An optional argument set that instructs the tool to isolate shorts between the specified connectivity text labels. The *text_name* specifies a text object name, which can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Specifying a single *text_name* parameter only makes sense if you use a wildcard that matches more than one net name.

The *text_name* may be specified as LVS_GROUND_NAME or LVS_POWER_NAME. These keywords correspond to the nets specified in [LVS Ground Name](#) and [LVS Power Name](#) statements in the rule file, respectively.

If you do not include the -name parameter, the default behavior is this:

```
-name "??"
```

and the tool processes all text object names.

- **-inverse**

An option specified with -name that causes only shorts containing names that are not in the *text_name* list to be selected for short isolation. Any short that includes at least one name not in the *text_name* list is selected for short isolation analysis.

- **-help**

An option that shows the command usage.

Return Values

Short isolation transcript and the file specified in the **-report_file** argument of the `qs::get_shorts` command.

The format of a returned short is this:

```
Short circuit (index: <n>) (cell: <cell>) (net_id: <n>) (text: <name> was
assigned) .
<label> at (<x>,<y>)
<label> at (<x>,<y>)
```

The coordinates are in database units.

[Tcl Shell Error Messages](#).

Description

Performs short isolation on the current short from the **short_iterator**.

The short isolation results output is controlled by the options. By default, this command performs short isolation using the same method as the rule file statement [LVS Isolate Shorts YES BY CELL ALSO BY LAYER ALSO](#). This means short isolation reports both the cell of

origin and the layer involved in the short in addition to fully-merged results. This is regardless of what appears in the rule file for LVS Isolate Shorts.

Here is a table that shows possible -by_cell and -by_layer combinations.

Table 2-21. -by_cell and -by_layer Options

by_cell	by_layer	Results Presentation
no	no	Fully merged for all layers.
yes	no	Per cell for all layers.
also	no	Both fully merged and per cell for all layers.
no	yes	Per layer.
yes	yes	Per cell, per layer.
also	yes	Both per layer and per cell, per layer.
no	also	Both fully merged and per layer.
yes	also	Both per cell for all layers and per cell, per layer.
also	also	all of these: fully merged for all layers; per cell for all layers; per layer; and per cell, per layer

A sub-directory with the path “*svdb/topcell.phdb/isi/*” is created to hold results database files, which have a *.shorts* extension. These files can be opened in Calibre RVE, just as for LVS Isolate Shorts results.

Examples

This example shows a loop to isolate the shorts in the iterator si.

```
> set si [qs::get_shorts -report_file shorts.rep]
> while { $si ne "" } {
    puts [qs::isolate_short -it $si];
    qs::inc si
}
Short circuit (index: 0) (cell: a1220) (net_id: 1) (text: VDD).
  VDD at (10,83)
  VSS at (10,1)
GLOBAL HEURISTICS: HCC=8 FCC=2(2) HGC=88 FGC=88
SHORT 0: HCC=1 FCC=1(1) HGC=27 FGC=27 VDD VSS (2/2)/1/1/1/1 CPU TIME = 0
REAL TIME = 0  LVHEAP = 2/7/7  MALLOC = 57/57/57
Short circuit (index: 0) (cell: TOP) (net_id: 1) (text: VDD).
  VDD at (441,370)
  VDD at (442,217.5)
  VSS at (13,325)
  VSS at (13.5,217)
GLOBAL HEURISTICS: HCC=8 FCC=63(56) HGC=1037 FGC=4135
SHORT 0: HCC=7 FCC=38(31) HGC=232 FGC=1139 VDD VSS (4/2)/7/0/0/0 CPU
TIME = 0  REAL TIME = 0  LVHEAP = 2/7/7  MALLOC = 57/57/57
```

Related Topics

[short_db::isolate_short](#)

[qs::inc](#)

[Short Isolation Commands](#)

qs::isolate_shorts

Short isolation command. Using [short_db::isolate_shorts](#) instead is generally preferred.

Isolates shorts from an SVDB database.

Usage

```
qs::isolate_shorts [-cell {PRIMARY | ALL}] [[-and | -or] [-inverse] -name "text_name ..."]  
[-help]
```

Arguments

- **-cell {PRIMARY | ALL}**

An optional argument set that specifies the hierarchical level where text shorts are isolated.

PRIMARY — Isolates shorts between text objects at the top level. This is the default.

ALL — Isolates shorts between text objects in all cells. Short isolation occurs in each cell, including its sub-hierarchy. This option is best used if texting of net names is consistent for all cells in the design.

- **-and**

An option that specifies both the -cell and -name criteria must be satisfied in order for short isolation to occur. Either -and or -or must be specified if -name used.

- **-or**

An option that specifies the -cell or -name criteria must be satisfied in order for short isolation to occur. Either -and or -or must be specified if -name used.

- **-inverse**

An option specified with -name that causes only shorts containing names that are not in the *text_name* list to be selected for short isolation. Any short that includes at least one name not in the *text_name* list is selected for short isolation analysis.

- **-name “text_name ...”**

An optional argument set that specifies to isolate shorts between the specified connectivity text labels. The *text_name* specifies a text object name, which can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Specifying a single *text_name* parameter only makes sense if you use a wildcard that matches more than one net name.

The *text_name* may be specified as LVS_GROUND_NAME or LVS_POWER_NAME. These keywords correspond to the nets specified in [LVS Ground Name](#) and [LVS Power Name](#) statements in the rule file, respectively.

When you specify the -name parameter, then any other options must precede the -name parameter.

If you do not specify the -name argument, then the default behavior is this:

```
<-cell specification> -and -name "?"
```

and the tool processes all text object names.

- -help

An option that shows the command usage.

Return Values

A sub-directory with the path “*svdb/topcell.phdb/isi/*” is created to hold results database files, which have a *.shorts* extension. These files can be opened in Calibre RVE, just as for LVS Isolate Shorts results. The run transcript also contains short information.

[Tcl Shell Runtime Messages](#).

Description

Performs hierarchical short isolation using the data from the SVDB database. The short isolation is performed according to the options specified with [LVS Isolate Shorts YES](#) in the rule file. If [LVS Isolate Shorts NO](#) is specified, short isolation is run as if [LVS Isolate Shorts YES](#) is specified with no other keywords.

The command returns an ASCII short isolation database as for LVS Isolate Shorts YES. The format is discussed under “[Hierarchical Short Isolation Results Database](#)” in the *Calibre Verification User’s Manual*. The database can be used in Calibre LVS RVE.

Examples

This example shows the creation of a hierarchical short isolation database using the rule file settings.

```
> qs::isolate_shorts

HIERARCHICAL SHORT ISOLATION started.
HIERARCHICAL SHORT ISOLATION in cell cellA started.
  GLOBAL HEURISTICS: HCC=8 FCC=63(56) HGC=1037 FGC=4135
  SHORT 0: HCC=8 FCC=63(56) HGC=259 FGC=1814 VDD VSS (4/2)/8/4/4/4 CPU
  TIME = 1  REAL TIME = 1  LVHEAP = 2/7/7  MALLOC = 39/39/39
HIERARCHICAL SHORT ISOLATION in cell cellA completed.  CPU TIME = 1  REAL
  TIME = 1  LVHEAP = 2/7/7  MALLOC = 39/39/39  ELAPSED TIME = 12734

HIERARCHICAL SHORT ISOLATION timing:
STEP          1          2          3
-----
EXTRACTION    0/0        0/0        0/0
GRAPH SEARCH  0/0        0/0        0/0
  DEEP SHORTS 0/0        0/0        0/0
  LEAF REMOVAL 0/0        0/0        0/0
LAYER OPS     0/0        0/0        0/0
  PROMOTION    0/0        0/0        0/0
  CIA          0/0        0/0        0/0
OUTPUT         0/0        0/0        0/0
-----
TOTAL          0/0        0/0        0/0

HIERARCHICAL SHORT ISOLATION completed.  CPU TIME = 1  REAL TIME = 1
LVHEAP = 2/7/7  MALLOC = 39/39/39  ELAPSED TIME = 12734
SHORT ISOLATION RESULTS_DATABASE = lvs_report.shortcuts
> more lvs_report.shortcuts
cellA 1000
SHORT 1.  VDD - VSS in cellA
5 5 3 Apr 16 22:02:15 2008
2 Shorted texts:
"VDD" at (441, 370) on layer "50" SN 1
"VSS" at (13, 325) on layer "50" SN 58
p 1 22
SN 51
7000 355000
47000 355000
...
p 2 4
SN 55
7500 359000
9500 359000
9500 361000
7500 361000
p 3 4
SN 4
```

This is equivalent to specifying `qs::isolate_shorts -cell PRIMARY -and -name "?"`.

Related Topics

[short_db::isolate_shorts](#)

[Short Isolation Commands](#)

[qs::list_mask_polygon](#)

Short isolation and iterator data retrieval command.

Lists the mask polygons in the system.

Usage

`qs::list_mask_polygon -it short_iterator [-top_coordinates | -cell_coordinates] [-dbu] [-help]`

Arguments

- **-it *short_iterator***
A required argument and short iterator created by [qs::get_shorts](#).
- **-top_coordinates**
An option that specifies the top-level coordinates are used. This is the default.
- **-cell_coordinates**
An option that specifies the cell-level coordinates are used. The cell of the current index location of the short iterator is used.
- **-dbu**
An option that specifies integral database units are used.
- **-help**
An option that shows the command usage.

Return Values

One or more Tcl lists. Each list is comprised of a layer and cell name, followed by one or more lists of polygon vertex coordinates.

`{<layer> <cell> {<x1> <y1> <x2> <y2> ...} ...}`

[Tcl Shell Error Messages](#).

Description

Lists the mask polygons in the specified short iterator. Mask polygons are generated using the [qs::add_mask_polygon](#) command.

Coordinates are returned in top-cell space and in user units by default. The `-top_coordinates` option explicitly specifies that top-cell space is used. The `-cell_coordinates` option specifies cell-space coordinates. The cell for cell-space coordinates is determined by the current index location in the iterator.

The `-dbu` option specifies that integral database units are used. When this option is not specified, all returned coordinates are floating-point numbers.

Examples

```
> set si [qs::get_shorts -report_file shorts.rep]
> qs::list_mask_polygon -it $si
{metal1 TOP {0.0 0.0 100.0 0.0 100.0 100.0 0.0 100.0}} {metal2 TOP {0.0
0.0 100.0 0.0 100.0 100.0 0.0 100.0}} {200.0 200.0 300.0 200.0 300.0 300.0
200.0 300.0}}
```

Related Topics

[Short Isolation Commands](#)

qs::list_shorts

Short isolation command.

Returns information about original short circuit elements as a Tcl list.

Usage

qs::list_shorts [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

The following list of lists is returned for each short:

```
{<serial_number> <index> <cell_name> <net_id> <text> {<label1> <x1> <y1>}<br/>{<label2> <x2> <y2>} ...}
```

The shorts are listed in order of serial number starting with 0. The short path index number, cell name, net ID number assigned by the circuit extractor, and assigned net name from a connectivity text object are shown next. The text labels involved in the short with their coordinates in cell space are given as sub-lists.

[Tcl Shell Error Messages](#).

Related Topics

[qs::print_shorts](#)

[Short Isolation Commands](#)

qs::list_text

Short isolation and iterator control command.

Returns a list of text labels in the system.

Usage

`qs::list_text -it short_iterator [-top_coordinates | -cell_coordinates] [-dbu] [-help]`

Arguments

- **-it *short_iterator***
A required argument and short iterator created by [qs::get_shorts](#).
- **-top_coordinates**
An option that specifies the top-level coordinates of added text objects are returned. This is the default.
- **-cell_coordinates**
An option that specifies the cell-level coordinates of added text objects are returned. The cell of the current index location of the short iterator is used.
- **-dbu**
An option that specifies integral database units are returned for coordinates.
- **-help**
An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Description

Returns a Tcl list for text labels added to an iterator by the [qs::add_text](#) command. The list is of this form:

`{<label> <x> <y> <layer>}`

The x and y coordinates are returned in top-cell space and in user units by default. The **-top_coordinates** option explicitly specifies that top-cell space is used. The **-cell_coordinates** option specifies cell-space coordinates. The cell for cell-space coordinates is determined by the current index location in the iterator.

The **-dbu** option specifies that integral database units are used. When this option is not specified, all returned coordinates are floating-point numbers.

Examples

```
> set si [qs::get_shorts -report_file shorts.rep]
> qs::list_text -it $si
{VSS 23.0 42.0 metal2}
```

Related Topics

[Short Isolation Commands](#)

qs::net_devicenames

Net command. Corresponding standard command: [NET DEVICENAMES](#)

Returns a list of instance paths of all devices on a specified layout net.

Usage

`qs::net_devicenames -net layout_net_path [-cell cell_name] [-help]`

Arguments

- `-net layout_net_path`

A required argument set that specifies a layout net name or path. The *layout_net_path* is relative to the query cell context.

- `-cell cell_name`

An optional argument set that specifies the query cell context. By default, the top level cell is assumed.

- `-help`

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Description

Returns a Tcl list of paths of device instances on the net specified by the *layout_net_path* in sub-lists. The device template number (\$D comment coded property) is also in each sub-list. The cell context is top level by default and can be changed with the -cell option.

The net specified by the *layout_net_path* is traced throughout the query cell context, not just downward from the given path.

Examples

This example shows the command output in interactive mode: Three instances are returned for the net PWR. The device template number is 1.

```
> qs::net_devicenames -net PWR
{x0/X0/M2 1} {x0/X0/M3 1} {x1/X0/M1 1}
```

Related Topics

[Net Commands](#)

qs::net_layers

Net command. Corresponding standard command: [NET LAYERS](#).

Returns a list of layers that comprise a net.

Usage

qs::net_layers -net *layout_net_path* [-cell *cell_name*] [-help]

Arguments

- **-net *layout_net_path***

A required argument set that specifies a layout net name or path. The *layout_net_path* is relative to the query cell context.

- **-cell *cell_name***

An optional argument set that specifies the query cell context. By default, the top level cell is assumed.

- **-help**

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Description

Returns a Tcl list of layer names having shapes on the net specified by the *layout_net_path*. The cell context is top level by default and can be changed with the -cell option.

The net specified by the *layout_net_path* is traced throughout the query cell context, not just downward from the given path.

Examples

This shows a command execution in interactive mode. The contents of var can be processed as a Tcl list.

```
> set var [qs::net_layers -net X0/OUT -cell CellA]
...
contact metal1 nsd psd via metal2
```

Related Topics

[Net Commands](#)

qs::net_not_connected

Net command. Corresponding standard command: [NET NOT CONNECTED](#).

Returns placements of a layout cell for which a specified net is not connected to a reference net.

Usage

```
qs::net_not_connected -ref_nets net_list -cell layout_cell -net checked_net
    [-context_cell layout_cell] [-help]
```

Arguments

- **-ref_nets *net_list***
A required argument set specifying the names of nets to which connections are checked. The *net_list* is a Tcl list of net names. A net path through the hierarchy is accepted.
- **-cell *layout_cell***
A required argument set specifying a layout cell name.
- **-net *checked_net***
A required argument set specifying a name or node ID of a net located in the *layout_cell*. This net is checked for connections to any net in the *net_list*.
- **-context_cell *layout_cell***
An optional argument set that specifies a context cell (hierarchical ancestor) in which to perform the check. By default, this is the layout primary cell.
- **-help**
An option that shows the command usage.

Return Values

Tcl list.

Description

Returns a list of all placements of the *layout_cell* for which the *checked_net* is not connected to any net in the *net_list*. If the *checked_net* is connected to any *net_list* net throughout the hierarchy (by default), an empty list is returned. The -context_cell argument can limit the checked portion of the design. Returned values are with respect to this cell.

Cell names can be original cell names or FAUX BIN cells as identified in the LVS run transcript.

The order in which the returned placements are listed is arbitrary.

A *net_list* path that extends upward in the hierarchy is searched from its highest point in the hierarchy. For example, if net X1/X3/2 is used in the *net_list*, and this net connects to the net GND in the current query cell, the commands qs::net_not_connected -ref_net "X1/X3/2" -cell

“NAND” -net “GND” and qs::net_not_connected -ref_net “GND” -cell “NAND”-net “GND” are equivalent.

Examples

```
qs::net_not_connected -ref_nets "PWR" -cell "nand" -net "1"
```

This command finds placements of nand where net 1 in that cell is not connected to the PWR net in the context of the primary cell. An example return value could be the placement path:

X0/X0

Related Topics

[Net Commands](#)

qs::net_pins

Net command. Corresponding standard command: [NET PINS](#).

Returns names and locations of intentional device pins on a layout net.

Usage

```
qs::net_pins -net layout_net_path -file filename [-cell {iterator | cell_name}]  
[-filter filter_object] [-help]
```

Arguments

- ***layout_net_path***
A required argument set that specifies a path to a layout net. If -cell is unspecified, the path is in the top-level cell context; otherwise, it is in the context of the specified cell.
- **-file *filename***
A required argument set that specifies a pathname of an output file.
- **-cell {*iterator* | *cell_name*}**
An optional argument set that defines a context cell. By default, the top-level cell is the context cell. These arguments define the cell:
 - iterator* — A cell iterator, such as is created by [dfm::get_cells](#).
 - cell_name* — A name of a cell.
- **-filter *filter_object***
Optional argument set that specifies a [dfm::create_filter](#) object, which controls the layers that are processed. Any transformations defined in the object are also observed.
- **-help**
An option that shows the command usage.

Return Values

A report is output containing the following information:

```
Net_Pins <precision> // "Net_Pins" and precision
Pins:           // "Pins:"
0 0 <n> <date>      // n lines of text follow (n pins were found)
<pin_info_1>       // pin number 1 (see description for definition of
                    // pin_info)
...
<pin_info_n>       // pin number n
Layer_name_1        // first layer on which pins are present
<k> <k> 0 <date>    // k pins on this layer
p <i> 4             // i is the number of this pin in the "Pins:" check
                    // section
...
...                // four vertices of the first square
...                // remaining m-1 pins on this layer
...                // remaining layers on which pins are present
```

Tcl Shell Error Messages.

Description

Returns the intentional device pins attached to the net associated with the *layout_net_path*. Each pin is represented as a device pathname, a device type number (that is, the 0-based index of the device in a device table as shown under “[Devices](#)” on page 24), and a marker square centered at the pin location on the pin layer. The location on which the square is centered is an arbitrary point where the pin shape touches or overlaps the device seed shape. The actual shape of the pin is not given. Vertex coordinates are given in context cell space.

The pin location information is only available if the [Mask SVDB Directory](#) PINLOC or CCI options are specified in the rule file that generated the database.

The -filter option controls the layers processed by the command, the transformations of the pin coordinates, or both.

The pins are listed in the “Pins:” check section in an arbitrary order. Each line is a white-space-separated list of the following items:

```
pin_number // number of pin in the list of pins
device_path // instance pathname ending with device instance name
device_type // index of the device in the device table
pin_index // index of the pin in the ordered list of device pins
```

The pin_number is not strictly necessary since it simply increments by one for each line, but is included for convenience.

Related Topics

[Net Commands](#)

[qs::net_text_map](#)

Net command. Corresponding standard command: [NET TEXT MAP](#).

Returns the mapping of layout net names to system-generated node numbers.

Usage

```
qs::net_text_map {cell_iterator | cell_name} [-invalid] [-help]
```

Arguments

- ***cell_iterator***

An argument that specifies a cell iterator, such as is created by [dfm::get_cells](#). Either this argument or ***cell_name*** must be specified.

- ***cell_name***

An argument that specifies a cell name. Either this argument or ***cell_iterator*** must be specified.

- -invalid

An option that specifies only the correspondence for invalid nets is returned.

- -help

An option that shows the command usage.

Return Values

Tcl list of lists of this form:

```
{<name> <number>} {<name> <number>} ...
```

[Tcl Shell Error Messages](#).

Description

Generates a Tcl list of layout net names mapped to Calibre generated net numbers for the specified cell.

Certain text names are not used for SPICE netlist net names by Calibre. These net names are not included in the output map by default. The -invalid option causes the command to return only those names.

Related Topics

[Net Commands](#)

qs::net_trace

Net command. Corresponding standard command: [NET TRACE](#)

Returns the path of the highest representative of a net and all intermediate net pathnames between the net and its highest representative. The specified net path is also returned. This command requires the layout to be geometric.

Usage

```
qs::net_trace -net layout_net_path [-cell cell_name] [-help]
```

Arguments

- **-net *layout_net_path***

A required argument set that specifies a layout net name or path. The *layout_net_path* is relative to the query cell context.

- **-cell *cell_name***

An optional argument set that specifies the query cell context. By default, the top level cell is assumed.

- **-help**

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Description

Returns the path of the highest representative of the net associated with the *layout_net_path*. By default, the root of the path is the top level. Using the -cell parameter changes the context to the specified cell. Intermediate net paths between the highest representative net and the specified net are also returned, along with the specified net itself. The output is a Tcl list.

The trace proceeds upward from the given *layout_net_path* until it reaches the highest hierarchical representative within the cell context. This does not imply that the trace does not also descend into other instances below the primary level of the context cell, or that the trace does not descend into the instances on the given path by more than one pin, or that the trace does not connect through an external pin of a specified -cell parameter into higher levels of the design.

Examples

Assume this simplified layout netlist:

```
.SUBCKT inv VSS VDD IN Y
M0 Y IN VSS VSS n
M1 Y IN VDD VDD p
.ENDS
*****
.SUBCKT CellB 1 2 IN1
X0 2 1 IN1 OUT inv
.ENDS
*****
.SUBCKT TOPCELL GND PWR 3
X0 GND 3 PWR CellA
X1 PWR GND 3 CellB
.ENDS
```

This shows an interactive Tcl shell session related to the netlist:

```
qs::net_trace -net X1/X0/VSS
GND X1/2 X1/X0/VSS

qs::net_trace -net X0/VSS -cell CellB
2 X0/VSS

qs::net_trace -net 2 -cell CellB
2
```

Related Topics

[Net Commands](#)

qs::net_valid

Net command. Also used in Calibre YieldServer. Corresponding standard command: [NET VALID](#).

Indicates whether a layout or source net path is valid.

Usage

qs::net_valid *net_path* {-layout | -source} [-cell {*name* | *iterator*}] [-help]

Arguments

- ***net_path***

A required path of a net placement. By default, the path is relative to the top-level cell.

- **-layout**

Option that indicates a layout placement path is used. Either this option or **-source** must be specified. If this option is used, the persistent hierarchical database (PHDB) is queried first, followed by the cross-reference database (XDB) if available.

- **-source**

Option that indicates a source placement path is used. Either this option or **-layout** must be specified. If this option is used, then an XDB must exist.

- **-cell {*name* | *iterator*}**

An optional argument set that specifies the ***net_path*** is relative to a given cell. When used with **-source**, the specified cell must have a corresponding layout cell as the query is actually performed on the layout design. The cell is specified as follows:

name — Cell name.

iterator — Cell iterator, such as is created by [dfm::get_cells](#).

- **-help**

An option that shows the command usage.

Return Values

1 (or true) if the path is valid and 0 (or false) otherwise.

[Tcl Shell Error Messages](#).

Examples

```
> # test using layout path from top level
> qs::net_valid X0/X0/VDD -layout
1
```

Related Topics

[Net Commands](#)

qs::open_svdb

Administrative command.

Opens an SVDB database.

Usage

```
qs::open_svdb -svdb svdb [-rev revision] [-top_cell top_cell] [-phdb | -summary] [-help]
```

Arguments

- **-svdb svdb**

A required argument and name of the [Mask SVDB Directory](#) database to open. For short isolation or soft connection applications, this database should be generated using the Mask SVDB Directory SI keyword.

- **-rev revision**

An optional argument and revision name or non-negative integer corresponding to a PHDB revision. Revisions are generated using dfm::create_rev, dfm::save_rev, and short_db::merge_short_revs.

- **-top_cell top_cell**

An optional argument and top-level cell database record to open, as specified in a rule file [Layout Primary](#) statement.

- **-phdb**

An option that reads the PHDB for the specified SVDB directory. This is the default behavior.

- **-summary**

An option that opens and reads the summary information for the specified SVDB directory. Use the -summary option in LVS summary report generation applications.

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

Description

Opens the specified SVDB database. If the -top_cell option is used, then that cell's data is accessed.

When used without the -rev option, this command opens the default SVDB revision. This is typically the “master” database revision (revision 0). Revision 0 is not editable.

When the -rev option is used, the specified database revision is opened.

By default, the PHDB is read, if it is available. The -summary option causes only the summary information needed for a custom LVS summary report to be generated. See [qs::create_lvs_summary_report](#).

[Tcl Shell Error Messages](#).

Examples

```
>qs::open_svdb -svdb svdb
Opening database "svdb/TOP.phdb/db", revision 'master'
Loading hierarchy and connectivity
...
Loading device info
>
```

Related Topics

[Administrative Commands](#)

qs::parse_path

Design data query command. Corresponding standard command: [PARSE PATH](#).

Returns cell names corresponding to instances in an instance pathname in a Tcl list of lists. For primitive devices, returns their Device type and subtype (if any). Net names that terminate a path are returned in a separate list.

Usage

```
qs::parse_path {-layout | -source} [-cell cell_name pin_count] placement_path
```

Arguments

- **-layout**

A required argument that specifies the *placement_path* is from the Layout Path design.

- **-source**

A required argument that specifies the *placement_path* is from the Source Path design.

- **-cell *cell_name pin_count***

An optional argument set that specifies a root cell for the *placement_path*. The default is the primary cell. The *cell_name* is the name of the cell, and the *pin_count* is a positive integer corresponding to the number of cell ports. A cell must match both criteria or an error is given.

- ***placement_path***

A required pathname of a cell, device, or net placement. The path is relative to the primary cell by default. The root cell of the path can be changed with the -cell option.

Return Values

For the cell instances in the path, the command returns a Tcl list of lists of the following form:

```
{ {<instance_name> <cell_name> <port_count>} ... }
```

Each sub-list corresponds to a cell instance in the pathname. For a primitive device instance path, the final sub-list has this form:

```
{<instance_name> <device_type> [(<subtype>)] <pin_count>}
```

Net names that terminate a path are returned in a separate list:

```
{ {<instance_name> <cell_name> <port_count>} ... } {<net_name>}
```

This latter form also applies if there is an unresolved portion of the path.

[Tcl Shell Error Messages](#).

Description

Decomposes the ***placement_path*** into its constituent parts, verifies any instances against the cross-reference database (XDB), and returns cell names and port counts for cells having instance references in the path. An XDB is required as part of the SVDB in order for this command to function.

For primitive device instance paths, the Device type and subtype (if any) are returned for the device instance.

For net paths, the net name is returned in a separate list. This also applies to an unresolved portion of a path.

Recall that hierarchical LVS flattens cells that are used in parameterized subcircuit calls. These cells are also flattened in the cross-reference system and in the qs::parse_path command. For this reason, instance pathnames will contain flattened instance paths to parameterized devices rather than a hierarchical structure. For example, this parameterized instance call:

```
X0 1 2 3 ARB xyz=0.3
```

will flatten the cell ARB in the cross-reference database in order to evaluate the parameter xyz=0.3 within the cell. As a result, the cell ARB is not presented in the results from the qs::parse_path command. Rather, its devices are returned, including their flattened names. So a path query for X99/X0/M0 that goes through the instance X0 in the preceding instance call will return X0/M0 as the instance name for the transistor directly in the context of the cell called at the higher level by X99.

Examples

This shows the output of cell placement path:

```
> qs::parse_path -source X0/X44/XABC
{{X0 route66 44} {X44 blockA 128} {XABC abc_cell 4}}
```

An NMOS device instance path output would look like this:

```
> qs::parse_path -source X0/X44/XABC/M0
{{X0 route66 44} {X44 blockA 128} {XABC abc_cell 4} {M0 MN(nfet) 4}}
```

A net instance path output would look like this:

```
> qs::parse_path -source X0/X44/XABC/Net[0]
{{X0 route66 44} {X44 blockA 128} {XABC abc_cell 4}} {Net[0]}
```

Related Topics

[Design Data Query Commands](#)

qs::password

Administrative command. Corresponding standard command: [PASSWORD](#).

Sets a password for the current SVDB.

Usage

qs::password [*pwtext* | -clear]

Arguments

- *pwtext*
An optional string that serves as a password. This option must be specified if setting the password in a script. Using an empty quoted string clears an existing password.
- -clear
An option that removes a password. This is equivalent to specifying an empty string as the password.

Return Values

Confirmation message corresponding to action taken.

[Tcl Shell Error Messages](#).

Description

Sets or clears a password restricting access to the current SVDB database.

If the command is invoked without *pwtext* (in an interactive session), the command prompts for a password twice. The string provided must match for both prompts in order for the password to be set successfully.

Remember that single quotes (‘’) in Tcl do not quote strings but are treated as literal, which means they are part of a password if used in the *pwtext* argument.

Related Topics

[Administrative Commands](#)

qs::perf_stats

Administrative command.

Starts the performance monitor.

Usage

`qs::perf_stats -label name [-help]`

Arguments

- **-label *name***

A required argument and label name for a statistics report. The *name* must be a string.

- **-help**

An option that shows the command usage.

Return Values

Statistics report.

[Tcl Shell Runtime Messages](#).

Description

This command starts the performance statistics monitor and outputs time and memory usage to the shell transcript. The **-label *name*** argument is a string that is prefixed to the statistics report.

Use of this command requires the following environment variable to be set:

```
# C shell
setenv COCKPIT_PERF_STATS
```

You start the output of performance statistics by setting a variable, like this:

```
set stats [qs::perf_stats -label "message"] ;
```

The “message” here could be a string or any Tcl object that returns a string. To stop the statistics report, do this:

```
set stats "";
```

The stats variable is local to the procedure scope in which it appears.

The statistics are reported as follows:

```
<label name> - <date stamp>
-----
CPU TIME = <n> REAL TIME = <n> LVHEAP = <n>/<n>/<n> MALLOC = <n>/<n>/<n>
LVHEAP_PRE = <n>/<n>/<n> MALLOC_PRE = <n>/<n>/<n>
```

To disable the reporting of the “PRE” statistics, set the following environment variable:

```
# C shell
setenv COCKPIT_PERF_STATS_DISABLE_PRE
```

Examples

After setting this variable:

```
# C shell
setenv COCKPIT_PERF_STATS
```

you can then output report statistics in a loop, like this:

```
> set si [qs::get_shorts -report_file shorts.rep]
Short Iterator
while { $si != "" } {
    set serial_number [ qs::get_data -it $si -serial_number ]
    set stats [ qs::perf_stats -label "ISOLATE_SHORT($serial_number)" ]
    ...
    qs::inc si
    set stats ""
}
```

The output for each index of the iterator would be similar to this:

```
ISOLATE _SHORT(1) - Tue Jul 1 14:22:29 2008
-----
CPU TIME = 0.03 REAL TIME = 0.02 LVHEAP = 2/7/7 MALLOC = 57/57/57
LVHEAP_PRE = 2/7/7 MALLOC_PRE = 57/57/57
```

Related Topics

[Administrative Commands](#)

qs::placement_valid

Design data query command. This is the same as the [qs::device_valid](#) command when the placement is a device. Also used in Calibre YieldServer. Corresponding standard command: [PLACEMENT VALID](#).

Indicates whether a layout or source placement path is valid.

Usage

qs::placement_valid *placement_path* {-layout | -source} [-cell {*name* | *iterator*}] [-help]

Arguments

- ***placement_path***

A required path of a placement. By default, the path is relative to the top-level cell.

- **-layout**

An argument that indicates a layout placement path is used. Either this option or **-source** must be specified. If this option is used, the persistent hierarchical database (PHDB) is queried first, followed by the cross-reference database (XDB) if available.

- **-source**

An argument that indicates a source placement path is used. Either this option or **-layout** must be specified. If this option is used, then an XDB must exist.

- **-cell {*name* | *iterator*}**

An optional argument set that specifies the ***placement_path*** is relative to a given cell. When used with **-source**, the specified cell must have a corresponding layout cell as the query is actually performed on the layout design. The cell is specified as follows:

name — Cell name.

iterator — Cell iterator, such as is created by [dfm::get_cells](#).

- **-help**

An option that shows the command usage.

Return Values

A 1 (or true) if the path is valid and 0 (or false) otherwise.

[Tcl Shell Error Messages](#).

Examples

```
> # test using layout path from top level
> qs::placement_valid X0/X0 -layout
1
```

Related Topics

[Design Data Query Commands](#)

qs::print_shorts

Short isolation command.

Returns information about original short circuit elements.

Usage

`qs::print_shorts [-help]`

Arguments

- `-help`

An option that shows the command usage.

Return Values

The following appears for each short:

```
Short circuit (serial number: <n>) (index: <n>) (cell: <name>) (net_id:<n>) (text: <assigned_name>):  
    <label> at (<x>,<y>)  
    <label> at (<x>,<y>)  
    [...]
```

The shorts are listed in order of serial number starting with 0. The cell name, net ID number assigned by the circuit extractor, and assigned net name from a connectivity text object are shown. The index number corresponds to a short path in a short (there can be more than one). The text label values involved in the short are then listed with their cell-space coordinates.

[Tcl Shell Error Messages](#).

Related Topics

[qs::list_shorts](#)

[Short Isolation Commands](#)

qs::quit

Administrative command. Corresponding standard command: [QUIT](#).

Exits the Query Server Tcl shell. The command `qs::exit` is an alias and performs the same function. Alternatively, you can type “exit”.

Usage

`qs::quit [-force] [-help]`

Arguments

- `-force`
An option that closes the database, discarding all changes to the PHDB.
- `-help`
An option that shows the command usage.

Return Values

None.

Related Topics

[Administrative Commands](#)

qs::ran_erc

LVS custom report command.

Returns a 1 if ERC has been run. Otherwise, a 0 is returned.

Usage

qs::ran_erc [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Integer.

[Tcl Shell Error Messages](#).

Examples

This shows the return of the command when entered interactively and ERC has not been run:

```
>qs::ran_erc
0
```

Related Topics

[LVS Custom Report Commands](#)

qs::ran_softchk

LVS custom report command.

Returns a 1 if LVS Softchk has been run. Otherwise a 0 is returned.

Usage

qs::ran_softchk [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Integer.

[Tcl Shell Error Messages](#).

Examples

This shows the return of the command when entered interactively and [LVS Softchk](#) has not been run:

```
> qs::ran_softchk
0
```

Related Topics

[LVS Custom Report Commands](#)

[Soft Connection Checking Commands](#)

qs::rules_connectivity

Rule file query command. Corresponding standard command: [RULES CONNECTIVITY](#).

Returns rule file connectivity statement information.

Usage

qs::rules_connectivity [-help]

Arguments

- -help

An option that shows the command usage.

Return Values

Tcl list of lists. The braces “{ }” are literal.

```
{CONNECT layer1 [layerN... [BY layerC]]}
```

[Tcl Shell Error Messages](#).

Description

Returns the [Connect](#) and [Sconnect](#) statements in the rule file, as well as layer connections made by node-preserving layer operations. Connectivity derived from [Stamp](#) operations is also returned. The connectivity statements are returned as Tcl lists.

Encrypted layers and device layers with no connectivity are not returned.

Connections established through Connect or Sconnect are reported using the word CONNECT. Connections established through layer derivation are reported using the string IMPLICIT_CONNECT.

Examples

Assume the following in the rule file:

```
CONNECT A B BY C
D = B NOT E
F = D AND G
H = I STAMP BY F
```

This is the output:

```
{CONNECT A B BY C} {IMPLICIT_CONNECT D B} {IMPLICIT_CONNECT F B}
{IMPLICIT_CONNECT H B}
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_file_name

LVS rule file query command. Corresponding standard command: [RULES FILE NAME](#).

Returns the name of the rule file used to create the SVDB database.

Usage

qs::rules_file_name [-help]

Arguments

- -help

An option that shows the command usage.

Return Values

String.

[Tcl Shell Error Messages](#).

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_layer_types

Rule file query command. Corresponding standard command: [RULES LAYER TYPES](#).

Returns a Tcl list of lists of rule file layers and their types.

Usage

qs::rules_layer_types [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Tcl list of lists of the following form:

{<layer_name> <type>} ...

Description

Returns a list of layers with their corresponding types. The type descriptions are given under [LVS SETTINGS REPORT WRITE](#).

Examples

```
> qs::rules_layer_types
{Nwell DEV_PIN} {Nwell SCONNECT} {Nwell PEX} {nmos DEV_SEED} {nmos
DEV_PIN} {nmos CONNECT} ...
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_layout_case

Rule file query command. Corresponding standard command: RULES {LAYOUT | SOURCE} CASE.

Returns the Layout Case setting in the rule file.

Usage

qs::rules_layout_case [-help]

Arguments

- -help

An option that shows the command usage.

Return Values

Integer. 0 is NO, 1 is YES.

[Tcl Shell Error Messages](#).

Examples

This response indicates [Layout Case](#) YES is set:

```
> qs::rules_layout_case
1
```

Related Topics

[LVS Rule File Query Commands](#)

[qs::rules_lvs_compare_case](#)

Rule file query command. Corresponding standard command: [RULES LVS COMPARE CASE](#). Returns the LVS Compare Case keyword settings in the rule file: NO, YES, NAMES, TYPES, SUBTYPES, or VALUES as a Tcl list.

Usage

`qs::rules_lvs_compare_case [-help]`

Arguments

- `-help`

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Examples

This response indicates [LVS Compare Case NAMES TYPES](#) is set:

```
> qs::rules_lvs_compare_case
NAMES TYPES
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_lvs_db_layer

Rule file query command. Corresponding standard command: [RULES LVS DB LAYER](#).

Returns the LVS DB Layer parameters in the rule file as a Tcl list of layer names.

Usage

qs::rules_lvs_db_layer [-help]

Arguments

- -help

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Examples

This example assumes [LVS DB Layer POLY DIFF](#) is in the rule file.

```
> qs::rules_lvs_db_layer
POLY DIFF
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_lvs_db_connectivity_layer

Rule file query command. Corresponding standard command: [RULES LVS DB CONNECTIVITY LAYER](#).

Returns the LVS DB Connectivity Layer parameters in the rule file as a Tcl list of layer names.

Usage

`qs::rules_lvs_db_connectivity_layer [-help]`

Arguments

- `-help`

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Examples

This example assumes [LVS DB Layer](#) POLY DIFF is in the rule file and that these are connectivity layers. The Mask SVDB Directory CCI option must be present in the rules.

```
> qs::rules_lvs_db_connectivity_layer
POLY DIFF
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_lvs_downcase_device

Rule file query command. Corresponding standard command: [RULES LVS DOWNCASE DEVICE](#).

Returns the LVS Downcase Device setting in the rule file.

Usage

`qs::rules_lvs_downcase_device [-help]`

Arguments

- `-help`

An option that shows the command usage.

Return Values

Integer. 0 is NO, 1 is YES.

[Tcl Shell Error Messages](#).

Examples

This response indicates [LVS Downcase Device](#) YES is set:

```
> qs::rules_lvs_downcase_device  
1
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_lvs_ground_name

Rule file query command. Corresponding standard command: RULES LVS {GROUND | POWER} NAME.

Returns the LVS Ground Name setting of the rule file.

Usage

qs::rules_lvs_ground_name [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Description

Returns the [LVS Ground Name](#) nets in the rule file as a Tcl list. If nets are declared as variables, the values of the variables are returned. If the names are declared with wildcards, the wildcards are returned, not the matching names.

Examples

This example assumes LVS Ground Name VSS GND is in the rule file:

```
> qs::rules_lvs_ground_name
VSS GND
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_lvs_power_name

Rule file query command. Corresponding standard command: RULES LVS {GROUND | POWER} NAME.

Returns the LVS Power Name setting of the rule file.

Usage

qs::rules_lvs_power_name [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

List.

[Tcl Shell Error Messages](#).

Description

Returns the [LVS Power Name](#) nets in the rule file as a Tcl list. If nets are declared as variables, the values of the variables are returned. If the names are declared with wildcards, the wildcards are returned, not the matching names.

Examples

This example assumes LVS Power Name VCC VDD is in the rule file:

```
> qs::rules_lvs_power_name
VCC VDD
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_lvs_spice_replicate_devices

Rule file query command. Corresponding standard command: RULES LVS SPICE
REPLICATE DEVICES.

Returns the LVS Spice Replicate Devices setting in the rule file.

Usage

`qs::rules_lvs_spice_replicate_devices [-help]`

Arguments

- `-help`

An option that shows the command usage.

Return Values

Integer. 0 is NO, 1 is YES.

[Tcl Shell Error Messages](#).

Examples

This response indicates LVS Spice Replicate Devices YES is set:

```
> qs::rules_lvs_spice_replicate_devices
1
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_precision

Rule file query command. Corresponding standard command: [RULES PRECISION](#).

Returns the Precision setting in the rule file as a floating-point number.

Usage

qs::rules_precision [-help]

Arguments

- -help

An option that shows the command usage.

Return Values

Floating-point number.

[Tcl Shell Error Messages](#).

Examples

This example assumes [Precision](#) 1000 is set in the rule file:

```
> qs::rules_precision
1000.0
```

Related Topics

[LVS Rule File Query Commands](#)

[qs::rules_source_case](#)

Rule file query command. Corresponding standard command: RULES {LAYOUT | SOURCE} CASE.

Returns the Source Case setting in the rule file.

Usage

`qs::rules_source_case [-help]`

Arguments

- `-help`

An option that shows the command usage.

Return Values

Integer. 0 is NO, 1 is YES.

[Tcl Shell Error Messages](#).

Examples

This response indicates [Source Case](#) YES is set:

```
> qs::rules_source_case
1
```

Related Topics

[LVS Rule File Query Commands](#)

qs::rules_unit_length

Rule file query command. Corresponding standard command: [RULES UNIT LENGTH](#).

Returns the Unit Length setting in the rule file. The return value is in meters and represented in scientific notation.

Usage

qs::rules_unit_length [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Floating-point number.

[Tcl Shell Error Messages](#).

Examples

This example assumes that [Unit Length u](#) is set in the rule file:

```
> qs::unit_length  
1e-06
```

Related Topics

[LVS Rule File Query Commands](#)

qs::softchk

Soft connection checking command.

Performs soft connection checks in a Calibre nmLVS Recon run on specified layers that exist in the PHDB.

Usage

```
qs::softchk -lower_layer layer [-append] [-file filename]
[ {-by_vertex_count [-lower | -upper | -contact] [-all]} |
 {by_contact_count -upper_layer upper_layer [-contact_layer layer] [-check_name name]}
 ]
```

Arguments

- **-lower_layer *layer***

A required argument set that specifies the second layer of an [Sconnect](#) operation. This is a stamped layer.

- **-append**

An option that specifies results are appended to an existing results database. The default is to replace an existing results database. If the command is specified multiple times in a single run, this option is especially useful in the qs::softchk commands after the initial one.

- **-file *filename***

An option that specifies the name of the results database. By default, the results database is *softchk.rdb*.

- **-by_vertex_count [-lower | -upper | -contact] [-all]**

An optional argument set that specifies the prevailing net in a stamping conflict is chosen based upon the total vertex count of polygons on a net. This is the default behavior and the same as the [LVS Softchk](#) default. May not be specified with -by_contact_count. The following options control which layers are reported:

-lower — Specifies the stamped layer is reported (the second layer in Sconnect). This is the default.

-upper — Specifies the stamping layer is reported (the first layer in Sconnect).

-contact — Specifies the contact or via layer is reported (the Sconnect BY layer).

-all — Specifies that all nodal layers involved the connection to the stamped layer are eligible for reporting. By default, only layers from discarded nets are eligible. This option has no effect when -lower is used because a stamped polygon is already included in the output.

- **-by_contact_count -upper_layer *upper_layer***

An optional argument set that specifies the command behaves like [lvs::softchk](#). The *upper_layer* is the stamping layer in an Sconnect conflict and must appear in a Connect statement but not as a contact (BY) layer.

- `-contact_layer layer`

An optional argument set specified with `-by_contact_count` that references an Sconnect BY layer, which is a contact or via layer. If this option is not specified, then a direct connection between *upper_layer* and *lower_layer* is assumed. If this option is specified, then the prevailing net in an Sconnect conflict is selected based upon the greatest number of contact *layer* shapes on the nets involved in the conflict.

- `-check_name name`

An option specified with `-by_contact_count` that defines a rule check name for the results. By default, the results are grouped with a *name* of the form `LVS_SOFTCHK_<N>_SOFTCHK`, where N is an integer.

Return Values

None.

[Tcl Shell Runtime Messages](#).

Description

This command can only be called from a Tcl script called from the calibre -recon `-softchk=tcl_script` option. See “[Calibre nmLVS Reconnaissance Command Line](#)” in the *Calibre Verification User’s Manual* for details on these run modes.

When `-by_vertex_count` (the default) is active, this command performs hierarchical LVS Softchk verification of [Mask SVDB Directory](#) SI data stored in a PHDB.

When `-by_contact_count` is specified, this command performs soft connection checking on the PHDB data in the same way as the `lvs::softchk` Tcl function (although `qs::softchk` does not have the same option set). If selection of prevailing nets in an Sconnect conflict based upon contact count is desired, then `-contact_layer` must be specified with `-by_contact_count`.

Sconnect operations that correspond to the arguments given in the `qs::softchk` command are expected to exist in the rule file that produced the SVDB; otherwise, an error is given.

By default, the command creates a results database called `softchk.rdb`. The name of the file can be changed by using the `-file` option.

The default results database format is discussed under “[LVS Softchk Results Database File Format](#)” in the *Calibre Verification User’s Manual*. If `-by_contact_count` is specified, then the database format follows the one discussed under “[Report Soft Connections Using Contact Counts](#)” in the same manual. Either database can be used in Calibre RVE for LVS. (Also see “[LVS Softchk Debug Method](#)” in the same manual.)

Examples

Assume this rule file excerpt:

```
MASK SVDB DIRECTORY svdb SI

CONNECT ptap metal1 BY contact
SCONNECT metal1 pwell BY ptap
```

These commands perform LVS Softchk verification in a calibre -recon -softchk=*tcl_script* run on layers in the PHDB:

```
# Recon Softchk Tcl script

# replace any existing softchk.rdb and output shorted pwells
qs::softchk -lower_layer pwell

# output taps from conflicting nets and the selected one
# append the results to the previous ones
qs::softchk -lower_layer pwell -by_vertex_count -contact -all -append
```

The soft connections are processed as with the LVS Softchk LOWER keyword, which is the default. They are then processed as with the CONTACT and ALL keywords. The *softchk.rdb* results database can be loaded into Calibre RVE for LVS to start the layout debugging process.

Related Topics

[Soft Connection Checking Commands](#)

qs::softchks

Soft connection checking command.

Performs soft connection checks in a Calibre nmLVS Recon run on the data in the PHDB.

Usage

```
qs::softchks {-sconnect_conflict | -config_file filename} [-append]
  [{-by_vertex_count [-lower | -upper | -contact] [-all]} | -by_contact_count] [-file filename]
  [-help]
```

Arguments

- **-sconnect_conflict**

An argument that specifies soft connections are output based upon [Sconnect](#) stamping warnings stored in the PHDB during the circuit extraction run. Either this argument or **-config_file** must be specified.

- **-config_file *filename***

An argument set that specifies a file containing instructions for how to perform soft connection checks (see the Description section). Either this argument set or **-sconnect_conflict** must be specified.

- **-append**

An option that specifies results are appended to an existing results database. The default is to replace an existing results database.

- **-by_vertex_count [-lower | -upper | -contact] [-all]**

An optional argument set that specifies the prevailing net in a stamping conflict is chosen based upon the total vertex count of polygons on a net. This is the default behavior and the same as the [LVS Softchk](#) default. May not be specified with **-by_contact_count**. The following options control which layers are reported:

-lower — Specifies the stamped layer is reported (the second layer in Sconnect). This is the default.

-upper — Specifies the stamping layer is reported (the first layer in Sconnect).

-contact — Specifies the contact or via layer is reported (the Sconnect BY layer).

-all — Specifies that all nodal layers involved the connection to the stamped layer are eligible for reporting. By default, only layers from discarded nets are eligible. This option has no effect when **-lower** is used because a stamped polygon is already included in the output.

- **-by_contact_count**

An option that specifies the prevailing net in a stamping conflict is chosen based upon the total contact count on a net. Contacts are those defined by the Sconnect BY keyword for layers involved in a conflict. May not be specified with **-by_vertex_count**.

- **-file *filename***
An option that specifies the name of the results database. By default, the results database is *softchk.rdb*.
- **-help**
An option that shows the command usage.

Return Values

None.

[Tcl Shell Runtime Messages](#).

Description

Performs hierarchical [LVS Softchk](#) verification of [Mask SVDB Directory](#) SI data stored in a PHDB. This command can only be called by a user from a Tcl script called from the calibre -recon -softchk=*tcl_script* option. See “[Calibre nmLVS Reconnaissance Command Line](#)” in the *Calibre Verification User’s Manual* for details on these run modes.

By default, soft connection checks are run in a similar manner as LVS Softchk LOWER in a circuit extraction run. The checks are run on any Sconnect stamped layers (the second layer in the Sconnect syntax) in the PHDB for which there was a stamping conflict during circuit extraction. Such conflicts are indicated in the circuit extraction report like this:

```
WARNING: Stamping conflict in SCONNECT - Multiple source nets stamp one target net.
```

If **-config_file** is used, then soft connections are conducted according to entries in a configuration file containing lines of this form:

```
<lower_layer> {LOWER | UPPER | CONTACT} [ALL]
```

The semantics of the arguments are exactly the same as for LVS Softchk.

By default, the command creates a results database called *softchk.rdb*, which is similar to the default *.softchk* results database produced by LVS Softchk. The name of the file can be changed by using the **-file** option. The database format is discussed under “[LVS Softchk Results Database File Format](#)” in the *Calibre Verification User’s Manual*. The database can be used in Calibre RVE for LVS.

Examples

If the LVS circuit extraction run shows this warning:

```
WARNING: Stamping conflict in SCONNECT - Multiple source nets stamp one target net.
```

then a “calibre -recon -softchk” run automatically applies this command:

```
qs::softchks -sconnect_conflict
```

The transcript shows this:

```
HIER LVS SOFTCHK execution started...
LVS SOFTCHK pwell LOWER.
Number of polygons produced by the current SOFTCHK operation = 1
LVS SOFTCHK TOTAL: CPU TIME = 0  REAL TIME = 0  LVHEAP = 3/5/67  MALLOC
= 77/77/77
LVS SOFTCHK execution completed. CPU TIME = 0  REAL TIME = 0  LVHEAP = 3/
5/67  MALLOC = 78/78/78 ELAPSED TIME = 1
LVS SOFTCHK RESULTS DATABASE = softchk.rdb
```

The soft connections are processed using the LVS Softchk LOWER keyword, which is the default. The *softchk.rdb* can be loaded into Calibre RVE for LVS to start the layout debugging process.

Related Topics

[Soft Connection Checking Commands](#)

[short_db::softchk](#)

qs::status

Administrative command. Corresponding standard command: [STATUS](#).

Provides the state of server settings. By default, only -phdb and -xdb option data is reported. Other settings are reported when explicitly specified.

Usage

```
qs::status [[{-file filename} | -list] {-filter filter_object [-filter_devices | -filter_layers |  
-maximum_vertex_count | -magnify_results | -reflectx_results | -rotate_results |  
-translate_results]}] | -phdb | -xdb | -help]
```

Arguments

- **-file *filename***

An optional argument set that specifies to output the status message to a file. Messages are returned to STDOUT by default. May not be specified with -list.

- **-list**

An option that species to return data in a Tcl list. Each requested data type is contained in its own sub-list. May not be specified with -file.

- **-filter *filter_object***

An optional argument set that specifies to output the settings in a [dfm::create_filter](#) object. If this argument set is specified without any of its modifier options, the status of all filters is returned. Otherwise, the response contains just the information related to the modifier option.

- **-filter_devices**

An option that specifies to return *filter_object* device type IDs. Device IDs correspond to the \$D property value in extracted SPICE netlists.

- **-filter_layers**

An option that specifies to return *filter_object* layer names.

- **-magnify_results**

An option that returns the magnification setting of the *filter_object*.

- **-maximum_vertex_count**

An option that returns the maximum vertex count setting of the *filter_object*.

The default maximum vertex count is 4096 and is not governed by a global setting in the Query Server Tcl shell or in Calibre YieldServer.

- **-phdb**

An option that specifies to return persistent hierarchical database (PHDB) information. This option may not be used with -filter.

- **-reflectx_results**
An option that returns the reflection setting of the *filter_object*.
- **-rotate_results**
An option that returns the rotation setting of the *filter_object*.
- **-translate_results**
An option that returns the translation setting of the *filter_object*.
- **-xdb**
An option that specifies to return cross-reference database (XDB) information. This option may not be used with -filter.
- **-help**
An option that shows the command usage.

Return Values

By default, a report is written in response format. If -file is used, the report is written to the specified file. If -list is used, a Tcl list of lists is returned with each sub-list being of the form: {<type> <value>}. The <value> can itself be a list, depending on the type.

Here is an example when -list is used by itself and no SVDB is loaded.

```
{phdb {NOT LOADED}} {xdb {NOT LOADED}}
```

Tcl Shell Error Messages

Examples

This is a default response given by the command in interactive mode when no SVDB is loaded:

```
> qs::status
Status 1000
Entries:
0 0 2
PHDB: NOT LOADED
XDB: NOT LOADED
```

If the PHDB or XDB is available, then the hierarchy type and Calibre version used to create the database are reported.

Related Topics

[Administrative Commands](#)

qs::write_lvs_comparison_summary_report_to_file

LVS custom report command.

Writes the summary comparison information to an output file. The created file should be opened for write access by the Tcl “open” command as a part of a summary report generation script.

Usage

`qs::write_lvs_comparison_summary_report_to_file file_channel [-help]`

Arguments

- *file_channel*
A required file output channel (sometimes called a file handle).
- -help
An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Examples

See “[Using LVS Custom Report Commands](#)” on page 207.

Related Topics

[LVS Custom Report Commands](#)

`qs::write_lvs_extraction_summary_report_to_file`

LVS custom report command.

Writes the summary extraction information to an output file. The created file should be opened for write access by the Tcl “open” command as a part of a summary report generation script.

Usage

`qs::write_lvs_extraction_summary_report_to_file` *file_channel* [-help]

Arguments

- *file_channel*
A required Tcl file output channel (sometimes called a file handle).
- -help
An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Examples

See “[Using LVS Custom Report Commands](#)” on page 207.

Related Topics

[LVS Custom Report Commands](#)

short_db::add_mask_polygon

Short repairs database command.

Suppresses short isolation on a specified layer. Also used in Calibre nmLVS Recon scripts.

Usage

```
short_db::add_mask_polygon -layer name -ec {x1 y1 x2 y2 x3 y3 ... [{x1 y1 x2 y2 x3 y3 ...}...]} -sn serial_number [-cell name] [-dbu] [-top_coordinates] [-cell_coordinates] [-help]
```

Arguments

- **-layer name**

A required argument and name (or number) of an original layer appearing in a [Connect](#) statement in the rules.

- **-ec {x1 y1 x2 y2 x3 y3... [{x1 y1 x2 y2 x3 y3 ...}...]}**

A required argument and Tcl list of lists of polygon vertex coordinates. Coordinates are floating-point numbers in user units by default. At least three non-collinear vertices must be specified. Additional vertices may be specified to make polygons of any shape. Each sub-list specifies a distinct polygon.

- **-sn serial_number**

A required argument and serial number of a short. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command.

- **-cell name**

An optional argument and name of a cell in which to isolate shorts.

- **-dbu**

An option that specifies coordinates are in database units.

- **-top_coordinates**

An option that specifies coordinates are given in top-level space. This is the default.

- **-cell_coordinates**

An option that specifies coordinates are given in cell-level space.

- **-help**

An option that shows the command usage.

Return Values

For a successful attachment, the following is returned:

```
(polygons) <layer> <cell> (vertices) <n>
```

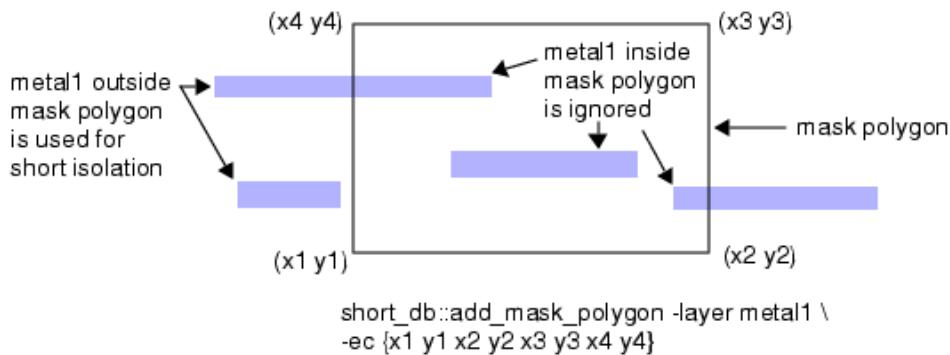
[Tcl Shell Error Messages](#).

Description

Creates mask polygons in memory that suppress short isolation on a specified layer and within the areas of the specified mask polygons.

The **-layer *name*** option specifies the layer to mask for short isolation. Any portion of a ***name*** layer that lies within the area of a mask polygon is ignored when using [short_db::isolate_short](#). The ***name*** parameter must appear in a Connect statement in the rule file for the layer to be masked for short isolation.

Figure 2-2. Masking Polygons with short_db::add_mask_polygon



The **-ec** option is required and specifies polygon vertex coordinates. Each coordinate list contains a set of x y values that specify polygons of at least three vertices.

If multiple mask polygons are specified and they touch or overlap, then they are merged into a single polygon. Subsequently, the command behaves as if the merged polygon had been specified. Specifying a layer that is not in a Connect statement is allowed but does not do anything useful.

The **-sn** option is required and specifies which serial number is used. A serial number is used to identify a group of related shorts. Shorts having the same serial number share the same electrical connection.

The **-cell** option specifies the cell to mask for short isolation. When this option is used, the list of coordinates given by **-ec** must be within the coordinate space of the cell. The masked polygon is removed within all cell instances. This parameter has no interaction with **-top_coordinates** or **-cell_coordinates**.

The **-dbu** option specifies that the given coordinates are in database units. When this option is not specified, all coordinates are floating-point numbers. When this option is specified, then all coordinates are integers.

By default coordinates are in top-cell space and in user units. The **-top_coordinates** option explicitly specifies that top-cell space is used. The **-cell_coordinates** option specifies cell-space coordinates. The cell for cell space coordinates is determined by the current short in the iterator.

Examples

This example shows coordinates in database units:

```
> short_db::add_mask_polygon -layer metall1 -ec {47000 302000 47000 \
271000 48000 271000 48000 312000} -sn 0 -dbu
polygon_p: 47000 302000 47000 271000 48000 271000 48000 312000
metall1 a1220 (vertices) 8
```

Related Topics

[short_db::delete_mask_polygon](#)

[Short Repair Database Commands](#)

`short_db::add_text`

Short repairs database command.

Adds a text object to the short isolation database. Also used in Calibre nmLVS Recon scripts.

Usage

```
short_db::add_text -label text -layer layer -pt {x y} -sn serial_number  
[-top_coordinates | -cell_coordinates] [-dbu] [-help]
```

Arguments

- **-label *text***

A required argument and text label to assign to a layer.

- **-layer *layer***

A required argument and layer (name or number). This command assigns text (label) to a connect layer. The ***layer*** must be one of these types:

- Original and is the source layer in an [Attach](#) operation. The label is assigned to the target layer of Attach.
- Original, non-contact, connected, and is not the source layer of an Attach operation. The label is assigned to the layer.
- Derived, non-contact, and is the target layer of an Attach operation. The label is assigned to layer.
- In Calibre Recon Softchk, it may be any derived layer. If the layer does not have connectivity, this can result in an unattached label warning.

- **-pt {*x y*}**

A required argument and coordinate list of a text object. The ***x y*** values are in user units in top-level space by default.

- **-sn *serial_number***

A required argument and serial number of a short. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command.

- **-top_coordinates**

An option that specifies coordinates are given in top-level space. This is the default.

- **-cell_coordinates**

An option that specifies coordinates are given in cell-level space.

- **-dbu**

An option that specifies coordinates are in database units.

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Runtime Messages](#).

Description

Adds a new text label to a short iterator at the specified location. A polygon to which the text label is attached must enclose the coordinates of the **-pt** argument list for the text attachment to be successful. If you attempt to add text to a location where there is no polygon, the command reports a failure to add the text. Top-level coordinates in user units are assumed by default.

The specified **layer** parameter determines the layer to which the specified text label is attached. The target layer for the text object must have connectivity information for the attachment to be successful. That is, the layer must appear in a Connect or Sconnect statement (but not after the BY argument), or it must be derived from such a layer by a series of node-preserving operations.

For the following discussion, assume these statements are in the rule file:

```
LAYER      diff 5
LAYER      poly 6
LAYER      metal1 10
LAYER      metal2 15
LAYER      txt 20

sd = diff NOT poly

CONNECT    diff poly metal1 BY contact
CONNECT    metal2 metal1 BY via

TEXT LAYER  txt
ATTACH     txt metal1
ATTACH     txt sd
```

The **layer** parameter must be one of these layer types:

- It is the first argument to an [Attach](#) statement. The label is assigned to the second argument of the same Attach statement. For example:

```
short_db::add_text -layer txt -label clk1 -pt {10.0 15.0} -sn 0
```

This assigns the label clk1 to metal1 at the specified coordinates. The effect of this command is similar to putting a label on the text layer using the corresponding Attach statement in the rule file.

- It appears in a Connect statement, but is not a BY layer. For example:

```
short_db::add_text -layer metal1 -label clk1 -pt {10.0 15.0} -sn 0
short_db::add_text -layer metal2 -label sig1 -pt {15.0 15.0} -sn 0
```

Both of these commands have the effect of placing text objects on Connect layers.

- It is derived, has connectivity information, is not a Connect or Sconnect BY layer, and appears as the second argument to an Attach statement. The text is attached to the target layer of the Attach statement. For example:

```
short_db::add_text -layer sd -label cap1 -pt {10.0 15.0} -sn 0
```

This assigns the label cap1 to layer sd. The sd layer meets the four criteria.

In Calibre nmLVS Recon Softchk, this command places the a text object at the specified location and performs the equivalent of [Virtual Connect Name *text*](#) to avoid producing an open circuit warning.

Examples

Example 1

This example assigns the label clk1 to metal1. The effect of this edit is similar to putting a label on the text layer using the Attach operation. Layer mark satisfies this criterion and can be assigned a label.

SVRF:

```
LAYER      metal1 10
LAYER      metal2 15
LAYER      mark   20
CONNECT    metal2 metal1 BY via
TEXT LAYER mark
ATTACH    mark metal1
```

QS Tcl Shell:

```
short_db::add_text -layer mark -label clk1 -pt {10.0 15.0} -sn 0
```

Example 2

Using the same Layer declaration statements in the previous example, layers metal1 and metal2 satisfy the attachment criterion and can be assigned a clock1 and signal label. The via layer cannot be assigned a label using short_db::add_text.

QS Tcl Shell:

```
short_db::add_text -layer metal1 -label clock1 -pt {10.0 15.0} -sn 0
short_db::add_text -layer metal2 -label signal -pt {15.0 15.0} -sn 0
```

Example 3

Using the same Layer declaration statements in the first example, this assigns the label cap1 to src_drn.

SVRF:

```
LAYER      metal1 10
LAYER      metal2 15
LAYER      mark 20
CONNECT    metal2 metal1 BY via
TEXT LAYER mark
ATTACH     mark metal1
LAYER      diff 25
TEXT LAYER diff
src_drn   = diff NOT poly
CONNECT    src_drn metal1 BY contact
ATTACH     diff src_drn
```

QS Tcl Shell:

```
short_db::add_text -layer src_drn -label cap1 -pt {10.0 15.0} -sn 0
```

Related Topics

[short_db::delete_text](#)

[Short Repair Database Commands](#)

short_db::assign_short

Short repairs database command.

Assigns or de-assigns a short vis-a-vis a user. If -sn is not specified, then the current short serial number is assumed. If -uid or -login are not specified, then the current user's ID is assumed.

Assignments can be reviewed with the command `short_db::user_info -show`. Also used in Calibre nmLVS Recon scripts.

Usage

```
short_db::assign_short [-show | -show_all] [-sn serial_number] [-r]
[-uid integer | -login string] [-help]
```

Arguments

- **-show**
An option that shows information on a user currently using the SVDB.
- **-show_all**
An option that shows information on all users currently using the SVDB.
- **-r**
An option that removes the assignment of a short to a specific user.
- **-sn *serial_number***
A optional argument and serial number of a short to assign. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command.
- **-uid *integer***
An optional argument set that specifies the user's ID number.
- **-login *string***
An optional argument set that specifies the user's login name.
- **-help**
An option that shows the command usage.

Return Values

Short assignment information.

[Tcl Shell Error Messages](#).

Examples

```
> short_db::assign_short -login jdoe
# (jdoe has to be a valid user on the system)
```

Related Topics

[short_db::comment_short](#)
[short_db::get_serial_number](#)
[short_db::user_info](#)
[short_db::get_status](#)
[short_db::set_status](#)

short_db::comment_short

Short repairs database command.

Adds or returns comments on a short. Returns comments if -text is not specified. If -sn is not specified, then the current short's serial number is used. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::comment_short [-text “*string*”] [-sn *serial_number*] [-help]

Arguments

- **-text “*string*”**
An optional argument and comment text contained in the *string*. The quotation marks are literal.
- **-sn *serial_number***
A required argument and serial number of a short. Serial numbers can be obtained from the short_db::get_serial_number or short_db::list_shorts command.
- **-help**
An option that shows the command usage.

Return Values

Comment strings.

[Tcl Shell Runtime Messages](#).

Examples

This shows assignment of comment text to the current short:

```
> short_db::comment_short -text "Assigned to jdoe."
```

Related Topics

[short_db::assign_short](#)

[short_db::get_serial_number](#)

[Short Repair Database Commands](#)

short_db::delete_mask_polygon

Short repairs database command.

Deletes all mask polygons added using the short_db::add_mask_polygon command that correspond to a serial number. Also used in Calibre nmLVS Recon scripts.

Usage

`short_db::delete_mask_polygon -sn serial_number [-help]`

Arguments

- **`-sn serial_number`**

A required argument and serial number of a short. Serial numbers can be obtained from the short_db::get_serial_number or short_db::list_shorts command.

- **`-help`**

An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Related Topics

[short_db::add_mask_polygon](#)

[Short Repair Database Commands](#)

`short_db::delete_text`

Short repairs database command.

Deletes a text object in the short isolation database. Also used in Calibre nmLVS Recon scripts.

Usage

```
short_db::delete_text -label name -sn serial_number
[-pt {x y} [-top_coordinates] [-cell_coordinates] [-dbu][-help]
```

Arguments

- **`-label name`**
A required argument and text label to delete.
- **`-sn serial_number`**
A required argument and serial number of a short. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command.
- **`-pt {x y}`**
An optional argument and coordinates of a text label. The coordinates are in user units in top-level space by default.
- **`-top_coordinates`**
An option that specifies coordinates are given in top-level space. This is the default.
- **`-cell_coordinates`**
An option that specifies coordinates are given in cell-level space.
- **`-dbu`**
An option that specifies coordinates are in database units.
- **`-help`**
An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Description

Deletes a text label added using `short_db::add_text` for the short with the *serial_number*. This command does not delete text labels that originated from the layout.

If the `-pt` option is not used, then the command deletes all added text labels with the name specified by *name*. If the `-pt` option is used, then only the added text label closest to the specified coordinates is deleted. Top-level coordinates in user units are used by default.

Text deletion actions are acknowledged by the command.

This command is useful for debugging shorts. By removing added text labels in the short iterator, this can assist in isolating the location of a short.

Examples

```
> short_db::delete_text -label VSS -pt {47.5 271.5} -sn 0
Deleted 47.5 271.5
```

Related Topics

[Short Repair Database Commands](#)

short_db::get_change_set

Short repairs database command.

Returns the name of the change set of the current short. Change sets contain records of any attempts to isolate or modify the properties of a short. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::get_change_set [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Related Topics

[short_db::list_change_sets](#)

[short_db::rename_change_set](#)

[short_db::remove_change_set](#)

[short_db::switch_change_set](#)

[short_db::get_serial_number](#)

Short repairs database command.

Returns the current short's serial number. A serial number is assigned to each short in the SVDB database, indexed from 0. Also used in Calibre nmLVS Recon scripts.

Usage

`short_db::get_serial_number [-help]`

Arguments

- `-help`

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Examples

```
> short_db::get_serial_number
0
```

Related Topics

[short_db::assign_short](#)
[short_db::get_status](#)
[short_db::set_status](#)
[short_db::comment_short](#)
[short_db::mark_fixed](#)

short_db::get_short_path_endpoints

Short repairs database command.

Returns endpoint data for a short path ID. Not used in Calibre nmLVS Recon Softchk.

Usage

short_db::get_short_path_endpoints -short_path_id *id* [-help]

Arguments

- **-short_path_id *id***

A required argument and short path ID given by the [short_db::get_short_paths](#) or [short_db::read_shorts_db](#) command.

- **-help**

An option that shows the command usage.

Return Values

A pair of strings of this form:

```
"<label_name>" at (<x>, <y>) on layer "<layer>"
```

The `label_name` is a text label value. The `x` and `y` are coordinates, and the `layer` is a layer name.

[Tcl Shell Runtime Messages](#).

Examples

This script loops over short serial numbers and outputs related short path endpoint data and short statuses:

```
set serial_number 0
while { [ catch { set ids [short_db::get_short_paths \
                     -sn $serial_number] } ] == 0 } {
    foreach id $ids {
        puts [short_db::get_short_path_endpoints -short_path_id $id]
        puts [short_db::get_status -sn $serial_number]
    }
    incr serial_number
}
```

Related Topics

[Short Repair Database Commands](#)

short_db::get_short_paths

Short repairs database command.

Returns a list of short path IDs. Not used in Calibre nmLVS Recon Softchk.

Usage

short_db::get_short_paths -sn *serial_number* [-help]

Arguments

- **-sn *serial_number***

A required argument and serial number of a short to process. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command.

- **-help**

An option that shows the command usage.

Return Values

Tcl list.

[Tcl Shell Runtime Messages](#).

Description

Returns a list of short path IDs. The list is empty if there are no shorts associated with the *serial_number*.

A short path ID is a unique identifier representing the two endpoints of a short path. The `short_db::get_short_path_endpoints` command is the downstream client for short path IDs.

Examples

This series of commands returns short path endpoint data when short 0 is non-empty.

```
set ids0 [short_db::get_short_paths -sn 0]
foreach id $ids0 {
    puts [short_db::get_short_path_endpoints -short_path_id $id]
```

Related Topics

[Short Repair Database Commands](#)

short_db::get_status

Short repairs database command.

Returns a status name of a short. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::get_status [-sn *serial_number*] [-help]

Arguments

- -sn *serial_number*

A optional argument and serial number of a short. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command.

- -help

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Description

Returns the user-defined status of a short. The status name is assigned with `short_db::set_status` `short_db::read_shorts_db`, or `short_db::mark_fixed`. When -sn is not specified, the current short serial number is used. After retrieving the status of a short, use `short_db::goto_short` to make that short the current one.

Related Topics

[short_db::assign_short](#)
[short_db::mark_fixed](#)
[short_db::comment_short](#)
[short_db::list_shorts](#)
[short_db::get_serial_number](#)

[short_db::goto_short](#)

Short repairs database command.

Makes the short with a specified serial number the current short. Also used in Calibre nmLVS Recon scripts.

Usage

`short_db::goto_short -sn serial_number [-help]`

Arguments

- **-sn *serial_number***

A required argument and serial number of a short. Serial numbers can be obtained from the `short_db::list_shorts` command.

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Examples

```
short_db::goto_short -sn 3
Current short serial number is 3
```

Related Topics

[short_db::assign_short](#)

[short_db::mark_fixed](#)

[short_db::comment_short](#)

[short_db::list_shorts](#)

short_db::init_database

Administrative command.

Initializes an interactive short repairs database. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::init_database [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

Description

Initializes the short repair database and sets the current change set to “default”. A short repairs session should begin with this command.

[Tcl Shell Error Messages.](#)

short_db::isolate_short

Short repairs database command.

Isolates a short in the shorts database. Also used in Calibre nmLVS Recon SI scripts but not in Recon Softchk.

Usage

```
short_db::isolate_short -sn serial_number
[-by_cell also | yes | no]
[-by_layer also | yes | no]
[-flat no | yes | if_fast]
[[-inverse] -name "text_name ..."]
[-path short_database_path]
[-help]
```

Arguments

- **-sn *serial_number***

A required argument and serial number of a short. Serial numbers can be obtained from the short_db::get_serial_number or short_db::list_shorts command.

- **-by_cell {also | no | yes}**

An optional argument set that specifies whether to report the cell of origin for short polygons. When this option is specified, one of these arguments must be present:

also — Shorts are reported both fully merged and in the cell of origin for each layer comprising the short. This is the default behavior if this option is not specified.

no — Short polygons are not reported in the cell of origin.

yes — Shorts are reported in the cell of origin for each layer comprising the short.

- **-by_layer {also | no | yes}**

An optional argument set that specifies whether to report separate results per short, per layer. When this option is specified, one of these arguments must be present:

also — Shorts are reported both fully merged and are reported with separate layer information. This is the default behavior if this option is not specified.

no — Shorts are not reported with layer information.

yes — Shorts are reported with separate layer information.

- **-flat { no | yes | if_fast }**

An optional argument set that specifies whether to perform the short isolation flat. When this option is specified, one of these arguments must be present:

no — Short isolation is performed hierarchically. This is the default behavior if this option is not specified.

yes — Short isolation is performed flat.

if_fast — Short isolation is performed flat if it will be faster than performing it hierarchically.

If the “yes” or “if_fast” (and flat isolation occurs) options are used, then all -by_cell “yes” or “also” results are reported at the top level.

- **-name “*text_name* ...”**

An optional argument set that specifies to isolate shorts between selected connectivity text labels. The *text_name* specifies a text object name, which can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters.

Specifying a single *text_name* parameter only makes sense if you use a wildcard that matches more than one net name.

The *text_name* may be specified as LVS_GROUND_NAME or LVS_POWER_NAME. These keywords correspond to the nets specified in [LVS Ground Name](#) and [LVS Power Name](#) statements in the rule file, respectively.

If you do not include the -name parameter, the default behavior is this:

```
-name "?"
```

and the tool processes all text object names.

- **-inverse**

An option specified with -name that causes only shorts containing names that are not in the *text_name* list to be selected for short isolation. Any short that includes at least one name not in the *text_name* list is selected for short isolation analysis.

- **-path *short_database_dir***

An optional argument set that specifies the directory for writing the short isolation database. By default, the database is written to the current working directory. These files can be opened in Calibre RVE, just as for LVS Isolate Shorts results.

- **-help**

An option that shows the command usage.

Return Values

A confirmation message of this format is returned to the Tcl shell terminal.

```
Short circuit (index: <i>) (cell: <cell>) (net_id: <n>) (text: <name> was
assigned).
<label> at (<x>,<y>)
<label> at (<x>,<y>)
```

These are the elements:

<i> — short serial number.

<cell> — name of cell in which the short occurs.

<n> — numeric net ID.
<name> — assigned text label name.
<label> — text label value of a shorted net.
<x> <y> — coordinate location of text label.

This line is also returned:

```
FOUND_SHORT {0 | 1} REPORT_PATH <lvs_report>-default-sn<n>.shorts
```

with these elements:

{0 | 1} — indicates whether a short was detected. 0 means no, 1 means yes.
<svdb> — Mask SVDB Directory.
<topcell> — primary cell name.
<n> — serial number of the short.

The output of the command by Tcl command substitution is simply the FOUND_SHORT line, which should be treated as a Tcl array to accommodate any future enhancements to this line.

Tcl Shell Error Messages.

Description

Performs short isolation on the current short.

By default, this command performs short isolation using the same method as the rule file statement **LVS Isolate Shorts YES BY CELL ALSO BY LAYER ALSO**. This means short isolation reports both the cell of origin and the layer involved in the short in addition to fully-merged results regardless of what appears in the rule file for LVS Isolate Shorts. The coordinates are in database units.

Here is a table that shows possible -by_cell and -by_layer combinations.

Table 2-22. -by_cell and -by_layer Options

by_cell	by_layer	Results Presentation
no	no	Fully merged for all layers.
yes	no	Per cell for all layers.
also	no	Both fully merged and per cell for all layers.
no	yes	Per layer.
yes	yes	Per cell, per layer.

Table 2-22. -by_cell and -by_layer Options (cont.)

by_cell	by_layer	Results Presentation
also	yes	Both per layer and per cell, per layer.
no	also	Both fully merged and per layer.
yes	also	Both per cell for all layers and per cell, per layer.
also	also	All of these: fully merged for all layers; per cell for all layers; per layer; and per cell, per layer.

The command writes an ASCII short isolation database as for LVS Isolate Shorts YES (see “Short Isolation” in the *Calibre Verification User’s Manual* for related details). The transcript reports the path to the database. If an LVS Report name is available in the rules, then the database has a name of the form *lvs_report-change_set-sn<N>.shorts*, where *lvs_report* is the name of the LVS Report, *change_set* is the name of the current short database change set, and <N> is the short serial number. If the LVS Report name is not available, then the database name is of the form *lvs.topcell.rep-change_set-sn<N>.shorts*, where *topcell* is the primary cell name.

Examples

Example 1

This is returned in an interactive shell:

```
> short_db::isolate_short -sn 0
Short circuit (index: 0) (cell: TOP) (net_id: 1) (text: PWR).
    GND at (13.5 217) (layer metal2)
    PWR at (442 217.5) (layer metal2)
FOUND_SHORT 1 REPORT_PATH lvs.report-default-sn0.shortcuts
> set s [short_db::isolate_short -sn 0]
Current short serial number is 0
    Short circuit (index: 0) (cell: TOP) (net_id: 1) (text: PWR).
        GND at (13.5 217) (layer metal2)
        PWR at (442 217.5) (layer metal2)
FOUND_SHORT 1 REPORT_PATH lvs.report-default-sn0.shortcuts
> puts $s
FOUND_SHORT 1 REPORT_PATH lvs.report-default-sn0.shortcuts
```

Example 2

This is a typical usage idiom in a script:

```
array set short_res [short_db::isolate_short -sn $id]
...
if { ! $short_res(FOUND_SHORT) } {
    puts "No short found!"
    break
}
```

Related Topics

[short_db::isolate_shorts](#)

[Short Repair Database Commands](#)

short_db::isolate_shorts

Short repairs database command.

Finds shorts in the short isolation database. Also used in Calibre nmLVS Recon SI scripts but not in Recon Softchk.

Usage

short_db::isolate_shorts

```
[-by_cell no | also | yes]
[-by_layer no | also | yes]
[-flat no | yes | if_fast]
[-cell {PRIMARY | ALL}] [[-and | -or] [-inverse] -name "text_name ..."]
[-path short_database_dir]
[-help]
```

Arguments

- **-by_cell** {no | also | yes}

An optional argument set that specifies whether to report the cell of origin for short polygons. When this option is specified, one of these arguments must be present:

no — Short polygons are not reported in the cell of origin. This is the default.
 also — Shorts are reported both fully merged and in the cell of origin for each layer comprising the short.
 yes — Shorts are reported in the cell of origin for each layer comprising the short.

- **-by_layer** {no | also | yes}

An optional argument set that specifies whether to report separate results per short, per layer. When this option is specified, one of these arguments must be present:

no — Shorts are not reported with layer information. This is the default.
 also — Shorts are reported both fully merged and are reported with separate layer information.
 yes — Shorts are reported with separate layer information.

- **-flat** {no | yes | if_fast}

An optional argument set that specifies whether to perform the short isolation flat. When this option is specified, one of these arguments must be present:

no — Short isolation is performed hierarchically. This is the default.
 yes — Short isolation is performed flat.
 if_fast — Short isolation is performed flat if it will be faster than performing it hierarchically.

If the “yes” or “if_fast” options are used (and flat isolation occurs), then all -by_cell “yes” or “also” results are reported at the top level.

- **-cell {PRIMARY | ALL}**

An optional argument set that specifies the hierarchical level where text shorts are isolated.

PRIMARY — Isolates shorts between text objects at the top level. This is the default.

ALL — Isolates shorts between text objects in all cells. Short isolation occurs in each cell, including its sub-hierarchy.

- **-and**

An option that specifies both the -cell and -name criteria must be satisfied in order for short isolation to occur. Either -and or -or must be specified if -name is used.

- **-or**

An option that specifies the -cell or -name criteria must be satisfied in order for short isolation to occur. Either -and or -or must be specified if -name is used.

- **-inverse**

An option specified with -name that causes only shorts containing names that are not in the *text_name* list to be selected for short isolation. Any short that includes at least one name not in the *text_name* list is selected for short isolation analysis.

- **-name “*text_name* ...”**

An optional argument set that specifies to isolate shorts between the specified connectivity text labels. The *text_name* specifies a text object name, which can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Specifying a single *text_name* parameter only makes sense if you use a wildcard that matches more than one net name.

The *text_name* may be specified as LVS_GROUND_NAME or LVS_POWER_NAME. These keywords correspond to the nets specified in [LVS Ground Name](#) and [LVS Power Name](#) statements in the rule file, respectively.

When you specify the -name parameter, then any other options must precede it.

If you do not specify the -name argument, then the behavior is taken from the LVS Isolate Shorts YES statement. The default is equivalent to this:

```
<-cell specification> -and -name "?"
```

and the tool processes all text object names.

- **-path *short_database_dir***

An optional argument set that specifies the directory for writing the short isolation database. By default, the database is written to the current working directory. These files can be opened in Calibre RVE, just as for LVS Isolate Shorts results.

- **-help**

An option that shows the command usage.

Return Values

A confirmation message of this format is returned to the Tcl shell terminal.

```
Short circuit (index: <i>) (cell: <cell>) (net_id: <n>) (text: <name> was
assigned).
<label> at (<x>,<y>)
<label> at (<x>,<y>)
```

These are the elements:

<i> — short serial number.

<cell> — name of cell in which the short occurs.

<n> — numeric net ID.

<name> — assigned text label name.

<label> — text label value of a shorted net.

<x> <y> — coordinate location of text label.

[Tcl Shell Runtime Messages](#).

Description

Sequentially runs hierarchical short isolation on all shorts using data from the SVDB database.

If [LVS Isolate Shorts](#) NO is specified in the rule file, then this command performs in the same way as LVS Isolate Shorts YES with no other keywords. This is the default behavior. If LVS Isolate Shorts YES is specified, then this command uses whatever short isolation options are specified in the rules.

Here is a table that shows possible -by_cell and -by_layer combinations.

Table 2-23. -by_cell and -by_layer Options

by_cell	by_layer	Results Presentation
no	no	Fully merged for all layers.
yes	no	Per cell for all layers.
also	no	Both fully merged and per cell for all layers.
no	yes	Per layer.
yes	yes	Per cell, per layer.
also	yes	Both per layer and per cell, per layer.
no	also	Both fully merged and per layer.

Table 2-23. -by_cell and -by_layer Options (cont.)

by_cell	by_layer	Results Presentation
yes	also	Both per cell for all layers and per cell, per layer.
also	also	All of these: fully merged for all layers; per cell for all layers; per layer; and per cell, per layer.

The command writes an ASCII short isolation database as for LVS Isolate Shorts YES (see “[Short Isolation](#)” in the *Calibre Verification User’s Manual* for related details). The transcript reports the path to the database. If an LVS Report name is available in the rules, then the database has a name of the form *lvs_report-change_set.shorts*, where *lvs_report* is the name of the LVS Report, and *change_set* is the name of the current short database change set. If the LVS Report name is not available, then the database name is of the form *lvs.topcell.rep-change_set.shorts*, where *topcell* is the primary cell name.

Examples

Example 1

```
> short_db::isolate_shorts
    Short circuit (index: 0) (cell: TOP) (net_id: 1) (text: PWR) .
        GND at (13.5,217)
        PWR at (442,217.5)
```

Related Topics

[short_db::isolate_short](#)

[Short Repair Database Commands](#)

short_db::list_change_sets

Short repairs database command.

Returns a list of all change sets. Change sets contain records of any attempts to isolate or modify the properties of a short. The change set must have changes in it in order to be listed. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::list_change_sets [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

List of change sets that have changes in them.

[Tcl Shell Error Messages](#).

Examples

```
> short_db::get_change_set  
default  
  
> short_db::switch_change_set -to change1  
Current change_set is "change1"  
  
> short_db::add_text -label VSS -pt {23 42} -layer metal2  
Added text on VSS 23.0 42.0 metal2  
  
> short_db::list_change_sets  
change1  
default
```

Related Topics

[short_db::remove_change_set](#)

[short_db::switch_change_set](#)

[short_db::rename_change_set](#)

short_db::list_shorts

Short repairs database command.

Returns information about original short circuit elements as a Tcl list, including short serial numbers. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::list_shorts [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

The following list of lists is returned for each short of the LVS Isolate Shorts variety:

```
{<serial_number> <index> <cell_name> <net_id> <text> {<label1> <x1> <y1>}  
<label2> <x2> <y2>} ...}
```

The shorts are listed in order of serial number starting with 0. These correspond to short circuit warnings in the circuit extraction transcript serially from start to end.

The short path index number, cell name, net ID number assigned by the circuit extractor, and assigned net name from a connectivity text object are shown next. The text labels involved in the short with their coordinates in cell space are given as sub-lists.

The following list of lists is returned for each short of the LVS Softchk variety:

```
{<serial_number> <cell_name> {<selected_net> <target_net> {<rejected_net>  
...}}}
```

The stamping conflicts are listed with a serial number starting with 0. These correspond to stamping conflict warnings in the circuit extraction transcript serially from start to end. A cell name follows.

The selected net is the net chosen for stamping the target polygon. A rejected net is a net that was not selected for stamping. A target net is a net of the target polygon at the cell level. Frequently, the target net and the selected net are the same, but the net IDs may differ due to hierarchical connectivity.

[Tcl Shell Error Messages](#).

Related Topics

[short_db::print_shorts](#)

[Short Repair Database Commands](#)

short_db::list_text

Short repairs database command.

Returns a list of text labels in the shorts database. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::list_text [-sn *serial_number*] [-top_coordinates | -cell_coordinates] [-dbu] [-help]

Arguments

- -sn *serial_number*

A optional argument and serial number of a short. Serial numbers can be obtained from short_db::get_serial_number or short_db::list_shorts. When this option is specified, the command output is limited to text objects involved in the specified short.

- -top_coordinates

An option that specifies the top-level coordinates of added text objects are returned. This is the default.

- -cell_coordinates

An option that specifies the cell-level coordinates of added text objects are returned.

- -dbu

An option that specifies integral database units are returned for coordinates.

- -help

An option that shows the command usage.

Return Values

A Tcl list of this form:

{<label> <x> <y> <layer>}

[Tcl Shell Error Messages](#).

Description

Returns a Tcl list of text labels consistent with any specified options. By default, information for all text objects in the shorts database are returned.

The x and y coordinates are returned in top-cell space and in user units by default. The -top_coordinates option explicitly specifies that top-cell space is used. The -cell_coordinates option specifies cell-space coordinates. The cell for cell-space coordinates is determined by the current index location in the iterator.

The -dbu option specifies that integral database units are used. When this option is not specified, all returned coordinates are floating-point numbers.

Examples

```
> set si [short_db::isolate_shorts]
> short_db::list_text
{TOP_CELL A2 -25.0 30.0 m9} {TOP_CELL B -20.0 30.0 m9}
```

Related Topics

[short_db::add_text](#)

[Short Repair Database Commands](#)

[short_db::mark_fixed](#)

Short repairs database command.

Marks the short associated with the serial number as “FIXED”. Also used in Calibre nmLVS Recon scripts.

Usage

[short_db::mark_fixed -sn *serial_number* \[-help\]](#)

Arguments

- **[-sn *serial_number*](#)**

A required argument and serial number of a short. Serial numbers can be obtained from the [short_db::get_serial_number](#) or [short_db::list_shorts](#) command.

- **[-help](#)**

An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Examples

> [short_db::mark_fixed -sn 0](#)

Related Topics

[short_db::assign_short](#)

[short_db::set_status](#)

[short_db::comment_short](#)

[short_db::get_serial_number](#)

short_db::merge_short_revs

Short repairs database command.

Merges database revisions containing shorts into a single revision and sets it as the default. Also used in Calibre nmLVS Recon SI scripts but not in Recon Softchk.

Usage

short_db::merge_short_revs -revs *revision_list* [-merge_into *target_revision*] [-help]

Arguments

- **-revs *revision_list***
A required argument and Tcl list of database revisions. Database revisions are generated by dfm::create_rev or short_db::merge_short_revs.
- **-merge_into *target_revision***
An optional argument set that causes revisions in the ***revision_list*** to be merged into the ***target_revision***, which then becomes the default. The ***target_revision*** must exist, and it may not be the master revision, a frozen revision, or any revision in the ***revision_list***.
- **-help**
An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Description

By default, this command collects data from each revision in the ***revision_list*** and merges the data into a new default revision. If **-merge_into** is specified, then the ***revision_list*** data is moved into the specified ***target_revision***, which becomes the default revision. The ***revision_list*** revisions that were merged are deleted.

If the data value from one revision conflicts with the data value from another revision, the data value from the higher-numbered (later) revision is chosen as the value for the merged revision.

Examples

Assume there are revision 1 and revision 2, and that in revision 1 the serial number 0 is in the fixed state, while in revision 2 the serial number 0 is in the un-fixed state. They are merged as follows:

```
short_db::merge_short_revs -revs {1 2}
```

A merge of these revisions creates a new default revision in which serial number 0 is in the un-fixed state because revision 2 is the later revision.

Related Topics

[Short Repair Database Commands](#)

short_db::print_shorts

Short repairs database command.

Returns information about original short circuit elements. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::print_shorts [-help]

Arguments

- **-help**

An option that shows the command usage.

Return Values

The following appears for each short of the LVS Isolate Shorts variety:

```
Short circuit (serial number: <n>) (index: <n>) (cell: <name>) (net_id: <n>)
  (text: <assigned_name>):
    <label> at (<x>,<y>)
    <label> at (<x>,<y>)
  [...]
```

The shorts are listed in order of serial number starting with 0. The cell name, net ID number assigned by the circuit extractor, and assigned net name from a connectivity text object are shown. The index number corresponds to a short path in a short (there can be more than one). The text label values involved in the short are then listed with their cell-space coordinates.

The following appears for each short of the LVS Softchk variety:

```
Sconnect conflict (serial number: <n>) (cell: <name>)
  Target net: <net_id>
  Selected net: <net_id>
  Rejected nets: <net_id> ...
```

The stamping conflicts are listed with a serial number starting with 0. These correspond to stamping conflict warnings in the circuit extraction transcript serially from start to end. A cell name follows.

The selected net is the net chosen for stamping the target polygon. A rejected net is a net that was not selected for stamping. A target net is a net of the target polygon at the cell level. Frequently, the target net and the selected net are the same, but the net IDs may differ due to hierarchical connectivity.

[Tcl Shell Error Messages](#).

Related Topics

[short_db::list_shorts](#)

[Short Repair Database Commands](#)

[short_db::read_shorts_db](#)

Short repairs database command.

Reads a *.shorts* results database and makes the repair status and path endpoint data available to the current session. Returns a Tcl list of short path IDs. Not used in Calibre nmLVS Recon Softchk.

Usage

[short_db::read_shorts_db](#) [-all | -sn *serial_number*] [-help]

Arguments

- **[-all](#)**
An option that causes all shorts in the database to be processed. This is the default.
- **[-sn *serial_number*](#)**
An option and serial number of a short to process. Serial numbers can be obtained from the [short_db::get_serial_number](#) or [short_db::list_shorts](#) command.
- **[-help](#)**
An option that shows the command usage.

Return Values

Tcl list.

[Tcl Shell Runtime Messages](#).

Description

Reads in a *.shorts* database. By default, all shorts are processed.

When a short is read in, the endpoint data of each short path is made available in the session through the returned list of short path IDs. A short path ID is a unique identifier representing the two endpoints of a short path. The [short_db::get_short_path_endpoints](#) command is the downstream client for short path IDs.

There are three statuses associated with shorts and short paths: FIXED, UNFIXED, and NOTRUN. FIXED means the short has been fixed, and UNFIXED means it has not. NOTRUN means short isolation has not been run in an interactive session. The command initializes shorts with the NOTRUN status. The status can be queried with the [short_db::get_status](#) command.

Examples

Assume Mask SVDB Directory has been specified in the rule with the SI keyword. This command creates the SVDB:

```
calibre -spice layout.sp rules
```

Assume a Query Server Tcl Shell script called *si.qs.tcl*:

```
# initialize short status
short_db::read_shorts_db
set status0 [short_db::get_status -sn 0]
# if short isolation has not been run, do it
if { $status0 eq "NOTRUN" } {
    short_db::isolate_short -sn 0
} else {
    puts "ERROR: shorts DB is not initialized correctly."
}
# get the status again
set status0 [short_db::get_status -sn 0]
if { $status0 eq "FIXED" }{
    puts ">>> short is fixed"
}
```

The script reads in the *.shorts* database and performs short isolation on the first short after adding a shape to the database. The script then checks the status of the short. The script can be added as follows:

```
calibre -qs -svdb svdb -exec si.qs.tcl
```

Related Topics

[Short Repair Database Commands](#)

short_db::remove_change_set

Short repairs database command.

Deletes the contents of a change set. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::remove_change_set -name *set* {-all | -sn *serial_number*} [-reset] [-help]

Arguments

- **-name *set***

A required argument and change set name to delete.

- **-all**

An option that deletes all changes associated with change set. Either this option or **-sn** must be specified.

- **-sn *serial_number***

An option and serial number of a short. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command. Either this option or **-all** must be specified.

- **-reset**

An option that restores short isolation results to the most recent revision created during batch short isolation or during a Calibre nmLVS Recon SI run.

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Runtime Messages](#).

Description

Deletes the contents of the specified change set for a short. Change sets contain records of any attempts to isolate or modify the properties of a short. By default, the changes associated with the current short are deleted. If the **-sn** option is used, then the specified short's changes are deleted. If the **-all** option is used, then the entire content of the change set is deleted.

Examples

```
> short_db::remove_change_set -name change2 -sn 0
Removed change_set group "change2"
```

Related Topics

[short_db::list_change_sets](#)

[**short_db::switch_change_set**](#)

[**short_db::rename_change_set**](#)

short_db::rename_change_set

Short repairs database command.

Renames a change set. Also used in Calibre nmLVS Recon scripts.

Usage

`short_db::rename_change_set -to new_set [-from old_set] [-help]`

Arguments

- **-to *new_set***
A required argument and new name of a change set.
- **-from *old_set***
An optional argument and name of a change set to rename. By default, the current change set is the *old_set*.
- **-help**
An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Description

Renames a change set to the ***new_set*** name. Change sets contain records of any attempts to isolate or modify the properties of a short.

By default, the current change set is renamed. The **-from** option specifies the name of another change set to rename.

After changes are made to the PHDB, switching back to a previous change set can be performed with [**short_db::switch_change_set**](#).

Examples

```
> short_db::rename_change_set -from default -to change0
Renamed change_set from "default" to "change0"
```

Related Topics

[**short_db::list_change_sets**](#)

[**short_db::switch_change_set**](#)

[**short_db::show_change_set**](#)

[**short_db::get_change_set**](#)

short_db::select_connects

Short repairs database command.

Selects Connect and Sconnect operations to be processed during stand-alone short isolation. The layer names specified in this command correspond to those in the rule file used to generate the PHDB. This command is only valid when used in Calibre nmLVS Recon Short Isolation (Calibre nmLVS Recon SI) scripts.

Usage

```
short_db::select_connects {{[-select_by_layer layer_list] ... [-select_by layer_list] ...} | -all} [-help]
```

Arguments

One of the **-select_by_layer**, **-select_by**, or **-all** options must be specified.

- **-select_by_layer *layer_list***

An argument set that specifies layers appearing in [Connect](#) or [Sconnect](#) operations.

The *layer_list* has this form:

```
{'layer ... '}
```

The braces indicate a Tcl list (which may have any appropriate Tcl form). At least one layer name that appears in some Connect or Sconnect operation must be specified in the list.

This option may be specified with **-select_by** and it may be specified more than once.

- **-select_by *layer_list***

An argument set that specifies sets of layers appearing in Connect or Sconnect operations.

The *layer_list* has either of these forms:

```
{'layer1 layer2'}
```

```
{'layer1 layer2 BY layerC'}
```

The braces indicate a Tcl list (which may have any appropriate Tcl form). The first form matches a two-layer Connect or Sconnect operation, and the second form matches an operation that has the BY keyword followed by a contact or via layer.

This option may be specified with **-select_by_layer** and it may be specified more than once.

- **-all**

An argument that selects all layers in Connect or Sconnect operations in the rule file.

- **-help**

An option that shows the command usage.

Return Values

None.

Description

Selects Connect or Sconnect operations to participate in stand-alone short isolation. This command may only be used in the Tcl script specified by the “calibre [-recon -si=tcl_script](#)” command line.

It is assumed that short isolation is performed using an appropriate Query Server command after specifying `short_db::select_connects`.

By default, layers that connect to device pins are processed in the Calibre nmLVS Recon SI flow as connectivity layers on which shorts can be detected. The `short_db::select_connects` command changes this behavior so that the Connect and Sconnect operations matching the layers specified by the command are processed. The **-select_by** and **-select_by_layer** options are generally more restrictive than the default behavior. The **-all** option is completely permissive.

The command options may be specified more than once, and the settings are aggregated from all occurrences. This behavior applies:

1. If a specified layer does not appear in the rules, then it is an error.
2. The set of all options is aggregated, and each *layer_list* is evaluated against rule file Connect and Sconnect operations.
3. If no active Connect or Sconnect operation can be mapped by a member of a specified *layer_list*, then it is an error.
4. The run proceeds with the selected mappings.

Running short isolation with a more limited set of Connect statements can improve runtime, but this can limit the number of layers upon which shorts are detected.

This command is analogous to the [LVS Recon Select Connects](#) specification statement.

When the `-recon -si` run’s rule file contains an LVS Recon Select Connects statement, then a limited set of connectivity operations may be processed. As a result, any PHDB data written may have limited connectivity information present. A warning is printed to the transcript noting this fact:

Warning: LVS SI SELECT CONNECTS is specified in batch rules.

Subsequent Calibre runs using the containing SVDB have access to the subset of connectivity operations present in the resulting SVDB directory’s rule file. This in turn limits the connectivity operations that may be selected by `short_db::select_connects`.

For example, if the following are present in the rule file:

```
CONNECT M1 M2 BY VIA1
CONNECT M2 M3 BY VIA2
CONNECT M3 M4 BY VIA3
```

and the first two operations are selected by these statements:

```
LVS RECON SELECT CONNECTS BY [M1 M2 BY VIA1]
LVS RECON SELECT CONNECTS BY [M2 M3 BY VIA2]
```

then connectivity data present in the SVDB that is generated is valid for the following layers only:

M1, M2, M3, VIA1, VIA2

Furthermore, only the previously selected Connect operations are available. Consequently any attempt to specify `short_db::select_connects -select_by_layer` with layers other than these generates an error. Similarly, any attempt to specify `-select_by` with layer lists that do not match the two Connect operations that were selected in the original run also generates an error.

Examples

Assume the rule file has these operations:

```
CONNECT m1 m2 BY via1
CONNECT m2 m3 BY via2
```

This code in a script called *leaf_shorts.qs* would match the previous Connect operations:

```
# find shorts in leaf cells
short_db::select_connects -select_by {m1 m2 BY via1} \
                           -select_by {m2 m3 BY via2}
short_db::isolate_shorts -by_layer yes -by_cell yes -cell ALL
puts "\n END OF SCRIPT \n"
```

This script can be called as follows using a [Mask SVDB Directory](#) from a previous run to isolate shorts on the specified Connect layers:

```
calibre -recon -si=leaf_shorts.qs -svdb svdb
```

Related Topics

[Short Repair Database Commands](#)

short_db::set_distribution_configuration

Short repairs database command. Used only in MTflex calibre -recon -si=*tcl_script* runs.

Configures runtime parameters for remote hosts in a stand-alone short isolation run that specifies a Query Server script. Also used in Calibre nmLVS Recon SI scripts but not in Recon Softchk.

Usage

```
short_db::set_distribution_configuration {{-by_turbo minimum_cpus} |  
{-by_memory minimum_memory} } [-min_remote_run short_count] [-help]
```

Arguments

- **-by_turbo *minimum_cpus***

An argument set that specifies a CPU count used to calculate the number of short isolation processes to run on a remote host. The ***minimum_cpus*** is a positive integer with a default value of 16, which tends to give the best performance. The units of ***minimum_cpus*** are CPUs/process (or, threads/process). May not be specified with **-by_memory**.

- **-by_memory *minimum_memory***

An argument set that specifies memory allocation used to calculate the number of short isolation processes to run on a remote host and the number of CPUs (or threads) per process. The ***minimum_memory*** is a positive integer with the units megabytes/process. By default, memory is not used as a criterion to select remote hosts. May not be specified with **-by_turbo**.

- **-min_remote_run *short_count***

An optional argument set that defines the minimum number of shorts to be processed by remote hosts. The default value is 2.

- **-help**

An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Description

Configures remote host selection and runtime processing criteria for the short isolation portion of a calibre -recon -si=*tcl_script* run. This command only applies in MTflex runs where **-remote** or **-remote_file** are specified on the command line and does not apply to the connectivity extraction portion of the run.

For this discussion, a “process” is an instance of the short isolation module. Multiple processes can run concurrently on a host, and each process can support multiple threads simultaneously.

By default, a remote host must have a minimum of 16 CPUs in order to be used, and the number of processes run on a remote host is this:

$$\text{process count} = \text{virtual core count} / 16$$

(Any remainders from the divisions shown in this section are discarded.)

The minimum number of CPUs per remote can be changed by specifying **-by_turbo** with a **minimum_cpus** value that serves as the divisor in the preceding equation. If a remote host does not have a CPU count greater than or equal to **minimum_cpus**, then this message is given:

<remote-host-name> has <n> CPUs (**minimum_cpus** CPUs are requested) - It will not be used in distributed SI.

Alternatively, the **-by_memory** option can be used to define the process count. By default, memory is not a criterion for determining a suitable remote host. If **-by_memory** is specified, then the process count per remote host is determined as follows:

$$\text{process count} = \text{remote host memory (MB)} / \text{minimum_memory}$$

Additionally, the number of CPUs (or threads) per process is calculated as follows:

$$\text{CPUs per process} = \text{virtual CPU count} / \text{process count}$$

If a remote host does not meet the minimum memory criterion, then this message is given:

<remote-host-name> has <m> MB of memory (<n>MB minimum is requested) - It will not be used in distributed SI.

The minimum number of shorts that can be processed using remote hosts is two. This can be adjusted using the **-min_remote_run** option.

Examples

Assume the following remote hosts:

host1: 64 CPUs, 64 virtual CPUs, 500GB RAM

host2: 64 CPU, 128 virtual CPUs, 500GB RAM

Assume this command line:

```
calibre -recon -si=distr.tcl -turbo -hyper -remote host1,host2 rules
```

Assume *distr.tcl* contains the following:

```
short_db::set_distribution_configuration -by_turbo 16; # default
short_db::isolate_shorts
```

The following are the process allocations:

host1: process count = 64/16, or 4 short isolation processes using 16 threads each

host2: process count = 128/16, or 8 short isolation processes using 16 threads each

Assume *distr.tcl* contains the following:

```
short_db::set_distribution_configuration -by_memory 100000; # 100GB  
short_db::isolate_shorts
```

host1: process count = 500/100, or 5 short isolation processes; CPUs per process = 64/5, so 12 threads each

host2: process count = 500/100, or 5 short isolation processes; CPUs per process = 128/5, so 25 threads each

Related Topics

[Short Repair Database Commands](#)

short_db::set_status

Short repairs database command.

Assigns or removes a status label. By default, this occurs for the current short. If the -sn option is used, then that short's status label is affected. Also used in Calibre nmLVS Recon scripts.

Usage

```
short_db::set_status -label "string" [-r] [-sn serial_number] [-help]
```

Arguments

- **-label "string"**
A required argument and status label name. The quotation marks are literal.
- **-r**
An option that removes the user defined status label for a short.
- **-sn serial_number**
An optional argument and serial number of a short. Serial numbers can be obtained from the short_db::get_serial_number or short_db::list_shorts command.
- **-help**
An option that shows the command usage.

Return Values

None.

[Tcl Shell Error Messages](#).

Examples

```
> short_db::set_status -label assigned
```

Related Topics

[short_db::assign_short](#)
[short_db::mark_fixed](#)
[short_db::comment_short](#)
[short_db::get_serial_number](#)

short_db::show_change_set

Short repairs database command.

Reports the changes to a short that are stored in a change set. Also used in Calibre nmLVS Recon scripts.

Usage

```
short_db::show_change_set {-all | -sn serial_number} [-cmdl] [-name set] [-unique] [-help]
```

Arguments

- **-all**
An option that returns all changes associated with a change set. Either this argument or the **-sn** argument set must be specified.
- **-sn *serial_number***
An option and serial number of a short. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command. Either this argument set or **-all** option must be specified.
- **-cmdl**
An option that causes the output to show coordinate values from the original input command lines rather than coordinates scaled to database units, which is the default. This option has no effect if the original commands in the change set used the `-dbu` option.
- **-name *set***
An option that specifies the name of a change set.
- **-unique**
An option that specifies to return only changes that are unique to a change set.
- **-help**
An option that shows the command usage.

Return Values

Returns a structured list of lists in this format:

```
{ serial_number si_value {change_list} } ...
```

Where *serial_number* is the integer serial number of the short and *si_value* is a Boolean value indicating whether short isolation was run on this short. Zero indicates that short isolation did not run yet. A value of 1 indicates that short isolation already ran. The *change_list* argument can be any number of user-defined Query Server Tcl shell commands.

[Tcl Shell Runtime Messages](#).

Description

Returns a list of changes associated with a change set. Change sets contain records of any attempts to isolate or modify the properties of a short. By default, the current change set is assumed. Another change set can be specified with the -name option.

By default, if user units are the inputs for commands in the change set, these are internally converted to database units, and the -dbu option is inserted into the commands stored in the change set. If the -cmdl option is used, then the original forms of the commands (with user units and no -dbu option) are output. If the original commands used the -dbu option, then there is nothing to revert, and they are output as they were originally issued.

The **-all** argument shows all changes associated with the change set. The **-sn** argument shows just the changes associated with the specified short.

Examples

```
> short_db::add_mask_polygon -layer metall1 -ec {22.5 73.0 22.5 42.0 23.5  
42.0 23.5 83.0} -sn 0  
  
# notice the change to database units by default  
> short_db::show_change_set -all  
{0 0 {short_db::add_mask_polygon -sn 0 -layer metall1 -dbu  
-cell_coordinates -cell nand -ec {22500 73000 22500 42000 23500 42000  
23500 83000 } }}  
  
# notice the original command is part of the output  
> short_db::show_change_set -all -cmdl  
{0 0 {short_db::add_mask_polygon -sn 0 -layer metall1 -cell_coordinates  
-cell nand -ec {22.5 73 22.5 42 23.5 42 23.5 83 } }}
```

Related Topics

[short_db::switch_change_set](#)
[short_db::rename_change_set](#)
[short_db::get_change_set](#)
[short_db::list_change_sets](#)

short_db::softchk

Soft connection checking command.

Performs soft connection checks in a Calibre nmLVS Recon run on a specified short. Not used in Calibre nmLVS Recon SI scripts.

Usage

```
short_db::softchk -sn serial_number [-append] [-file filename]
[{-by_vertex_count [-lower | -upper | -contact] [-all]}] | -by_contact_count
```

Arguments

- **-sn *serial_number***

A required argument and serial number of a short. Serial numbers can be obtained from the `short_db::get_serial_number` or `short_db::list_shorts` command.

- **-append**

An option that specifies results are appended to an existing results database. The default is to replace an existing results database. If the command is specified multiple times in a single run, this option is especially useful in the `short_db::softchk` commands after the initial one.

- **-file *filename***

An option that specifies the name of the results database. By default, the results database is `softchk.rdb`.

- **-by_vertex_count [-lower | -upper | -contact] [-all]**

An optional argument set that specifies the prevailing net in a stamping conflict is chosen based upon the total vertex count of polygons on a net. This is the default behavior and the same as the [LVS Softchk](#) default. May not be specified with `-by_contact_count`. The following options control which layers are reported:

-lower — Specifies the stamped layer is reported (the second layer in Sconnect). This is the default.

-upper — Specifies the stamping layer is reported (the first layer in Sconnect).

-contact — Specifies the contact or via layer is reported (the Sconnect BY layer).

-all — Specifies that all nodal layers involved the connection to the stamped layer are eligible for reporting. By default, only layers from discarded nets are eligible. This option has no effect when `-lower` is used because a stamped polygon is already included in the output.

- **-by_contact_count**

An optional argument set that specifies the prevailing net in stamping conflict is based upon maximum contact count. Contacts are defined by the Sconnect BY keyword.

Return Values

None.

Tcl Shell Runtime Messages.

Description

This command can only be called from a calibre -recon run that uses the `-softchk=tcl_script` option. See “[Calibre nmLVS Reconnaissance Command Line](#)” in the *Calibre Verification User’s Manual* for details on these run modes.

When `-by_vertex_count` (the default) is active, this command performs hierarchical LVS Softchk verification of [Mask SVDB Directory](#) SI data stored in a PHDB.

When `-by_contact_count` is specified, this command performs soft connection checking on the PHDB data using maximum contact count as the criterion for determining the prevailing net.

By default, the command creates a results database called `softchk.rdb`. The name of the file can be changed by using the `-file` option.

The default results database format is discussed under “[LVS Softchk Results Database File Format](#)” in the *Calibre Verification User’s Manual*. If `-by_contact_count` is specified, then the database format follows the one discussed under “[Report Soft Connections Using Contact Counts](#)” in the same manual. Either database can be used in Calibre RVE for LVS. (Also see “[LVS Softchk Debug Method](#)” in the same manual.)

Examples

Assume this rule file excerpt:

```
MASK SVDB DIRECTORY svdb SI  
  
CONNECT ptap metal1 BY contact  
SCONNECT metal1 pwell BY ptap
```

These commands perform LVS Softchk verification in a calibre -recon `-softchk=tcl_script` run on layers in the PHDB:

```
# for transcript details  
short_db::print_shorts  
# write each short to its own database.  
# include stamped and stamping shapes. output all stamping shapes.  
set sn [expr [llength [short_db::list_shorts]] - 1]  
while $sn >= 0 {  
    short_db::softchk -sn $sn -file short_${sn}.rdb  
    short_db::softchk -sn $sn -file short_${sn}.rdb -append \  
        -by_vertex_count upper -all  
    incr sn -1  
}
```

The soft connections are processed as with the LVS Softchk LOWER keyword, which is the default. They are then processed as with the UPPER and ALL keywords. The each `short_<sn>.rdb` database can be loaded into Calibre RVE for LVS to start the layout debugging process.

Related Topics

[qs::softchk](#)

[Short Repair Database Commands](#)

short_db::switch_change_set

Short repairs database command.

Defines a new change set. Also used in Calibre nmLVS Recon scripts.

Usage

short_db::switch_change_set -to *new_set* [-help]

Arguments

- **-to *new_set***

A required argument and change set name.

- **-help**

An option that shows the command usage.

Return Values

Confirmation message.

[Tcl Shell Error Messages](#).

Description

Defines a new change set if *new_set* does not exist and makes it the current one. Otherwise, it makes the (existing) *new_set* the current one. Change sets contain records of any attempts to isolate or modify the properties of a short.

Examples

```
> short_db::get_change_set  
Current change_set is "change0"  
  
> short_db::switch_change_set -to change2  
Current change_set is "change2"
```

Related Topics

[short_db::list_change_sets](#)

[short_db::show_change_set](#)

[short_db::rename_change_set](#)

[short_db::get_change_set](#)

short_db::user_info

Short repairs database command.

Shows or modifies user information for a short. Also used in Calibre nmLVS Recon scripts.

Usage

```
short_db::user_info [-show | -show_all] [-login string] [-uid integer]  
[-fname string] [-email string] [-phone string] [-comment “string”] [-help]
```

Arguments

- **-show**
An option that shows information about the current user.
- **-show_all**
An option that shows information on all users.
- **-login *string***
An optional argument set that specifies a user’s last name.
- **-uid *integer***
An optional argument set that specifies a numeric user account ID.
- **-fname *string***
An optional argument set that adds a user’s first name.
- **-email *string***
An optional argument set that adds an email address.
- **-phone *string***
An optional argument set that adds a user’s phone number.
- **-comment “*string*”**
An optional argument set that adds a user’s comment. Double quotation marks are used to enclose the comment.
- **-help**
An option that shows the command usage.

Return Values

If no options are specified, the command returns nothing. For -show or -show_all, user information is returned.

[Tcl Shell Error Messages](#).

Description

Shows or adds user information, depending on the options specified. The -show and -show_all options act independently of other options for this command. If one of these options is specified together with the adding options, the specified information is added first, then shown as appropriate.

If neither the -login nor the -uid option is supplied, the present user's login credentials are assumed for the options that add information. Otherwise, added information is for the user specified by -login or -uid. If both options are specified, the -login option has priority and is searched for first, followed by the -uid option. If the login or uid is found, information is added accordingly. When using any of the adding options, existing records are overwritten if they already exist.

Examples

```
> short_db::user_info -show
userid                      30007
lastname
firstname
login                      jdoe
phone
email
comment
timestamp                  2009-01-19 22:40:25
short_sn                    0
```

The timestamp is self-explanatory. The short_sn is the short serial number currently assigned to the user.

Related Topics

[short_db::assign_short](#)
[short_db::set_status](#)
[short_db::get_status](#)
[short_db::get_serial_number](#)

Using Short Database Commands

This procedure takes you through the process of running various short database commands. They store the current state of short isolation information in a persistent database.

Using short database commands often requires detailed knowledge of the layout, and such is the case here. Frequently, this type of analysis would be performed interactively in Calibre RVE and your layout editor, but it is possible to use the Query Server commands also.

Different versions of the shorts database can be initialized and stored. These can be accessed later to continue short isolation troubleshooting.

In the Query Server Tcl shell, the short_db:: commands should generally be considered before the qs:: command equivalents. The short_db:: interface is generally simpler to use and offers certain extra features.

The Calibre nmLVS Recon Short Isolation (Calibre nmLVS Recon SI) and Softchk (Calibre nmLVS Recon Softchk) flows support the use of short database commands through the `-recon -si=tcl_script` or `-softchk=tcl_script` command line options. The short database commands appear in the Tcl scripts.

Prerequisites

- A hierarchical Calibre nmLVS connectivity extraction (calibre -spice) run has been performed.
- An SVDB generated using [Mask SVDB Directory](#) SI.
- A Calibre Query Server license. In the Calibre nmLVS Recon SI flow, a Calibre nmLVS Recon SI license and a hierarchical LVS license pair are also needed.

Procedure

1. Start the Query Server Tcl shell:

```
calibre -qs -turbo -svdb svdb
```

Session data is stored in a DFM database revision. This revision can be created explicitly using [dfm::create_rev](#) or it is done implicitly when the session is saved.

2. Execute this command:

```
> short_db::init_database
Current change_set is "default"
```

3. Print information about the shorts in the SVDB database:

```
> short_db::print_shorts
Short circuit (serial number: 0) (index: 0) (cell: TOP) (net_id: 1)
  (text: PWR)
    IO at (13.5,217)
    PWR at (442,217.5)
  ...

```

A serial number tracks a short in the Query Server Tcl shell and is used in many short_db:: commands as an argument.

4. Use the [short_db::isolate_short](#) command to run short isolation on the short with serial number 0:

```
> short_db::isolate_short -sn 0
Current short serial number is 0
...
Short circuit (index: 0) (cell: TOP) (net_id: 1) (text: PWR).
  IO at (13.5 217) (layer metal2)
  PWR at (442 217.5) (layer metal2)
```

5. Use a [short_db::add_mask_polygon](#) command to delete in memory portions of a layer involved in the short. (This would require specific knowledge of the layout to be effective.)

```
> short_db::add_mask_polygon -sn 0 -layer metal1 -ec {542 70441 542  
72802 2099 72802 2099 70441} -dbu -cell a1620 -cell_coordinates  
Added mask polygon on metal1  
> short_db::add_mask_polygon -sn 0 -layer metal1 -ec {420160 151635  
420160 154995 422805 154995 422805 151635}  
Added mask polygon on metal1
```

6. Re-run isolate short with the [short_db::isolate_short](#) command:

```
> short_db::isolate_short -sn 0  
Current short serial number is 0  
(Masked out 1 original layer(s) ...)  
Short circuit (index: 0) (cell: TOP) (net_id: 1) (text: PWR).  
    IO at (13.5 217) (layer metal2)  
    PWR at (442 217.5) (layer metal2)  
  
NOTE: No shorts were isolated in this cell.  
  
(Restored 1 original layer(s) ...)
```

The masking polygons removed part of metal1 in memory, which causes the short to disappear.

7. Display the current change_set name with the [short_db::get_change_set](#) command:

```
> short_db::get_change_set  
default
```

A unique change set is always active at any point in the session. The first change set is always “default”. All short repairs database commands that affect the state of a short (that is, adding and removing text or polygons) are associated with a change set.

8. Review the changes affecting the state of short serial number 0 within the change set “default” with the [short_db::show_change_set](#) command:

```
> short_db::show_change_set -name default -sn 0  
{0 1 {short_db::add_mask_polygon -sn 0 -layer metal1 -dbu  
-cell_coordinates -cell a1620 -ec {542 72802 2099 72802 2099 70441  
542 70441 } }} {0 1 {short_db::add_mask_polygon -sn 0 -layer metal1  
-dbu -cell_coordinates -cell TOP -ec {420160000 154995000 422805000  
154995000 422805000 151635000 420160000 151635000 } }}
```

The changes are returned as a Tcl list of lists.

9. Use [short_db::rename_change_set](#) to preserve all changes in this change set:

```
> short_db::rename_change_set -to IOfixed -from default  
Renamed change_set from "default" to "IOfixed"
```

10. Save the changes made to the current DFM database with the [dfm::save_rev](#) command:

```
> dfm::save_rev

Creating revision 1

Saving revision 1
```

11. Close the current SVDB with the [qs::close_svdb](#) command:

```
> qs::close_svdb
```

12. The next portion of the example shows how to trace work that was done on the PHDB in a previous revision. Open SVDB revision 1 with the [qs::open_svdb](#) command:

```
> qs::open_svdb -svdb svdb -rev 1
Opening database "svdb/TOP.phdb/db", revision 1
```

13. Set the `change_set` to “PGfixed” with the [short_db::switch_change_set](#) command to access the revisions performed on serial number 0:

```
> short_db::switch_change_set -to IOfixed
IOfixed
```

14. Use [short_db::show_change_set](#) to display the saved commands that get applied when you move to a different short:

```
> short_db::show_change_set -all -name IOfixed
{0 1 {short_db::add_mask_polygon -sn 0 -layer metal1 -dbu -
cell_coordinates -cell a1620 -ec {542 72802 2099 72802 2099 70441
542 70441 } } } {0 1 {short_db::add_mask_polygon -sn 0 -layer metal1
-dbucell_coordinates -cell TOP -ec {420160000 154995000 422805000
154995000 422805000 151635000 420160000 151635000 } } }
```

The changes are as when the default change set was in effect.

15. Use [short_db::get_status](#) to recall the status of the short:

```
short_db::get_status
FIXED
```

You could now run [short_db::isolate_shorts](#) to determine if there are more shorts to fix.

16. Quit the session.

```
> qs::quit
```

Using LVS Custom Report Commands

This procedure shows how to create a custom LVS report.

With an [LVS Summary Report](#) statement and a TVF script, you can use [LVS Custom Report Commands](#) to create a custom report.

Prerequisites

- A complete Calibre nmLVS rule file.
- A Calibre Query Server license.

Procedure

1. Add this block to your rule file:

```
LVS SUMMARY REPORT lvs.summary
/*
set spc {
}
# User customized status messages
proc my_print_happy_face { filehandle } {
    global spc
    puts $filehandle "
    puts $filehandle "$spc      #
    puts $filehandle "$spc      #
    puts $filehandle "$spc #   #
    puts $filehandle "$spc # #
    puts $filehandle "$spc   #
    puts $filehandle "
}

proc my_print_sad_face { filehandle } {
    global spc
    puts $filehandle "
    puts $filehandle "$spc #   #
    puts $filehandle "$spc  # #
    puts $filehandle "$spc   #
    puts $filehandle "$spc # #
    puts $filehandle "$spc #   #
    puts $filehandle "
}

# main report generation
proc my_summary_report { arg } {
    if { $arg == "-help" } {
        puts "my_summary_report { -help | report_file }"
        return
    }
    set report_name $arg
    if [ catch { open $report_name w } repfile ] {
        puts "Cannot open $report_name."
        return
    }
}
```

```

puts $repfile ""
puts $repfile ""
puts $repfile "=====
puts $repfile "==== XYZ Corp. GENERATED CUSTOM SUMMARY REPORT"
puts $repfile "===="
puts $repfile ""

set overall_status 0
# Start out with overall status based on CORRECT compare status:
# Note that CORRECT statuses include 'CORRECT' and 'CORRECT ...'
# with some additional overall status info.

if { ! [ string match "CORRECT*" \
      [ qs::get_run_summary_data -lvs_compare_status ] ] } {
    set overall_status 1
};

# Custom override of overall status based on comparison results
# and presence of any shorts in extraction:
# Protect against dry-run
if { [ qs::extraction_summary_is_available ] } {
    if { [ qs::get_run_summary_data -short_circuit_count ] } {
# Override overall status if any shorts are found
        puts -nonewline $repfile "Shorts in the Design: "
        puts -nonewline $repfile \
            "[ qs::get_run_summary_data -short_circuit_count ]"
        puts $repfile " shorts found."
        set overall_status 1
    }
}

set printed_something 0
# Protect against dry-run
if { [ qs::extraction_summary_is_available ] } {
    if { \
        [ qs::write_lvs_extraction_summary_report_to_file $repfile ] } {
            incr printed_something
    }
}

# enable as desired
if { $overall_status == 0 } {
    my_print_happy_face $repfile
} else {
    my_print_sad_face $repfile
}

# Protect against dry-run
if { [ qs::comparison_summary_is_available ] } {
    if { \
        [ qs::write_lvs_comparison_summary_report_to_file $repfile ] } {
            incr printed_something
    }
}
if { ! $printed_something } {
    puts $repfile "No LVS Run information is available."
}

```

```
        close $repfile
        return $printed_something
    }

# Run the report - use the summary interface to get the name of the
# LVS SUMMARY file we are writing to:

my_summary_report [ qs:::get_run_summary_data -lvs_summary_report ]

*/]
```

2. Run Calibre nmLVS with the complete Calibre nmLVS rule file.

Results

Results in the generated report appear similar to this:

```
=====
== XYZ Corp. GENERATED CUSTOM SUMMARY REPORT
==

Extraction Summary
-----
Extraction Start Time: Thu Sep 29 14:10:33 2016
Extraction Finish Time: Thu Sep 29 14:10:34 2016
Circuit Extraction Report: lvs.report.ext
Circuit Extraction SPICE Netlist: svdb/top.sp

          #
          #
          #   #
          #   #
          #   #
          #

Comparison Summary
-----
Comparison Start Time: Thu Sep 29 14:10:35 2016
Comparison Finish Time: Thu Sep 29 14:10:35 2016
LVS Comparison Report: lvs.report
LVS Comparison Status: CORRECT.

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

Chapter 3

Hcell and Hierarchical Analysis

Hcells define correspondences between layout and source cells in hierarchical LVS runs. Calibre nmLVS-H cannot generally determine these correspondences if the subcircuit names are different in the two designs, so a user-defined hcell list is needed in that case. The Query Server can assist in finding a prospective set of hcells that act as design partitions to improve comparison efficiency and to make debugging simpler.

The Query Server Hcell analysis interface (also called the hcell analyzer) helps to identify a useful set of hcells to test. Commands are provided to read input data, identify hcells through various manual or automatic means, and to produce reports detailing cells, effectiveness of hcells, statistics on contained instances, and so forth.

Note

 Any hcell list generated by the Query Server should be regarded as provisional and should be thoroughly tested to ensure validity and performance.

The functionality in the Tcl shell based Query Server (calibre -qs) supersedes similar functionality in the command-line based Query Server (calibre -query). The historical method for doing Hcell analysis using the non-Tcl Query Server is discussed under “[Hcell List Management Using Standard Commands](#).”

The [LVS Hcell Report](#) specification statement has related functionality and can be a useful adjunct to the Query Server hcell interface.

See “[Hcells](#)” in the *Calibre Verification User’s Manual* for more information about hcells.

Current Hcell List	212
Evaluation of Hcells for LVS Comparison	213
Evaluation of Hcells for Netlist Extraction	216
Hcell Selection Limitations	217
Netlist Hcell Analysis and Reporting Flow	217
Tcl Shell Hcell Analysis Commands.....	219
Hcell Evaluation Report Format	240
Hcell Analysis and Hierarchy Tree Report Format.....	242
Unbalanced Hcell Reporting.....	246
Cell Matches Using Placement Matching	246

Current Hcell List

The Query Server maintains an internal list of hcells called the current hcell list. Hcells are added to this list by reading the rule file, by reading an hcell list file, by specifying individual hcell pairs in Query Server commands, or by adding automatically matched hcells. Hcells can be removed from the current list either globally or by specifying individual pairs to be excluded. The current hcell list is used for Query Server analysis and reporting functions, as well as to select a list of hcells to be used for testing in LVS verification.

To establish the current hcell list for the tool, it is typical to first read the rule file using the [hcells::read](#) command. The netlists referenced in the rules are read in based on your rule file settings and the hcells::read command options. (If the Layout System is geometric, then only the Source Path netlist is read.)

Hcells that are explicitly or automatically identified can become part of the current hcell list. Hcells are added to the current hcell list in the following ways:

- [Hcell](#) rule file statement. (The [LVS Exclude Hcell](#) statement filters out hcells.)
- [hcells::hcells](#) command together with an hcell file used with the -hcell command line option.
- [hcells::hcell](#) command for specifying individual cells.
- [hcells::add_matching_hcells](#) or [hcells::select](#) command together with [hcells::automatch](#) or [hcells::placementmatch](#) commands.

The hcells::hcells and hcells::hcell commands are explicit ways of adding hcells to the current hcell list. The hcells::automatch and hcells::placementmatch commands cause cells to be matched automatically; however, cells identified by these commands are *not immediately added to the current hcell list*. The hcells::add_matching_hcells or hcells::select command adds automatically selected cell pairs to the current hcell list.

You can remove cells from the current hcell list with the following commands:

- [hcells::clear_hcell](#) command removes an individual hcell pair.
- [hcells::clear_hcells](#) command removes all hcells.

You can select the cells to be included in an output hcell list using [hcells::select](#) (selected cell pairs then comprise the current hcell list). You can then print the current hcell list using [hcells::print_hcells](#).

The [hcells::status](#) command reports current settings of various hcell analysis commands.

Evaluation of Hcells for LVS Comparison

Hcells in the current hcell list should be evaluated for their effectiveness in LVS comparison. This can be done initially in the Query Server Tcl shell. Ultimately, every hcell list used for LVS comparison should be tested in Calibre nmLVS-H.

The [hcells::select](#) command measures the effectiveness of cells at reducing the *total hierarchical instance count* (THIC) of the design, and the proportion of internal instance counts within cells compared to the total instance count of the design. (For the remainder of this discussion, reviewing the “[Hcell Evaluation Report Format](#)” on page 240 can be helpful.) THIC is only applied as a selection criterion when hcells::select is used.

THIC is defined as the number of primitive devices and hcell placements in the design after it is flattened based upon rule file settings and internal heuristics (including unbalanced hcell expansion, flattening inside cells, cell copies made for one-to-many hcells, and so forth), but before any device reductions occur. Device filtering is accounted for, but unused device filtering is not.

THIC varies with the set of hcells. The improvement percentage in THIC is the primary measure for an hcell’s efficiency. This percentage gives the factor by which THIC can be reduced by the addition of the given hcell, assuming all previously listed hcells were in place prior to the addition.

The remaining potential improvement is an estimate of how much more improvement is possible, based on comparison with the lesser of the THIC with all candidate cells added as hcells, or the least THIC calculated to that point.

Candidate cells come from automatic matching (automatching, or automatch) if enabled, placement matching if enabled, and existing hcells in the hcell list if evaluation of current hcells is enabled. If evaluation of current hcells is disabled, all candidates are added as hcells that were in the current hcell list prior to invoking hcells::select. Then the remainder of the selection process starts.

The hcells::select command sets the thresholds for measuring effectiveness of cells in the current hcell list. Effectiveness thresholds ensure that the most effective hcells are selected. More specifically, these are the cases:

- Threshold evaluation of current hcells is enabled ([hcells::evaluate_current_hcells -yes](#), the default). In this case, the candidate set of hcells will include all hcells in the current hcell list, plus any hcells found by automatching or placement matching (if enabled). Before selection begins, the current hcell list is cleared. Then hcells taken from the candidate list are added to the list until the hcells::select thresholds are met. The result is the set of selected hcells. Cells are picked from the candidate list in order of estimated effectiveness at reducing the THIC.

- Threshold evaluation of current hcells is disabled. In this case, the candidate set of hcells will only include hcells found by automatching or placement matching (if enabled). The current hcell list is not cleared before beginning.

To illustrate the need for effectiveness measurement, `hcells::automatch` selects all cells that have the same name in layout and source. Many of those are small cells (like vias), or cells that appear only a few times in the design and have few internal instances. These cells contribute virtually nothing to reducing data size or execution time, and sometimes may even degrade performance if used as hcells. (Automatched cells may also result in discrepancies, for example, when the layout version of the cell does not closely correspond to the source.) Using `hcells::automatch -strict` with a reasonable `hcell::select` effectiveness threshold can result in a better preliminary hcell list.

The `hcells::select` command iteratively identifies hcells based upon their potential to reduce the THIC, given the existing set of hcells (including those chosen on previous iterations). By default, the savings threshold is set so that when the potential remaining THIC savings falls below 30 percent, further hcells will not be selected unless they meet the instance count criterion.

The `hcells::select` command also considers the instance count in a cell (at its top level) and compares this to the total flat instance count of the design. By default, if a cell's internal instance count represents one percent of the total flat instance count of the design, that cell is selected as an hcell, even if it contributes nothing to THIC savings. This instance count threshold ensures large cells with many instances can also serve as hcells. The instance counts are shown in the Instance Count In This Cell column of the hcell evaluation report.

When `hcells::evaluate_current_hcells -no` is set, current hcells are always kept in the current hcell list regardless of whether they actually meet the evaluation thresholds.

Following is an example of hcell report generation where both layout and source are assumed to be SPICE.

```
$ calibre -qs
> hcells::automatch -strict
> hcells::read -rules lvs.rules
> hcells::select -file cells.report
> hcells::print_hcells -file hcells
```

This script identifies hcell candidates based upon the strict automatch algorithm, which is discussed in the `hcells::automatch` documentation. Additionally, any `Hcell` specification statements in the rule file are read. The entire collection of hcell candidates are then evaluated against the thresholds (which are defaults in this case).

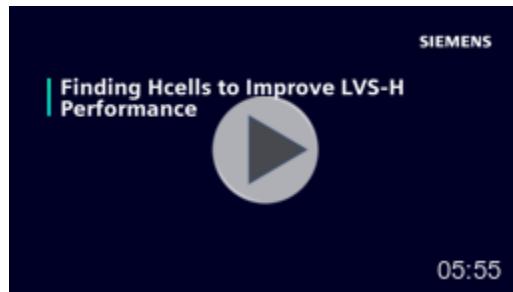
Note

 If the layout is geometric, then only the source netlist is evaluated, and `hcells::select` does not evaluate hcell effectiveness thresholds.

The contents of the *cells.report* file are the same as the ones generated by the [NETLIST SELECT HCELLS](#) command.

The `hcells::print_hierarchy` command prints a hierarchy report that can be useful when adding or removing hcells from the current list. See “[Netlist Hcell Analysis and Reporting Flow](#)” on page 217. You can use `hcells::hcell` to add hcells to the current list and `hcells::clear_hcell` to remove hcells from the current list.

The final generated hcell list should be thoroughly tested in LVS runs to ensure it performs well and produces valid results.



Hcell Cycles and Conflicts

An hcell cycle occurs when there is a circular chain of references among hcells. This is an error and is discussed in detail under “[Hierarchical Cells Forming a Cycle](#)” in the *Calibre Verification User’s Manual*.

An hcell conflict is a pair of cells that should correspond, except that this would result in a one-to-many correspondence where some cell on the “many” side is the ancestor of another cell on the “many” side. In that case, if these cells were made to correspond, there would be a cycle in the merged hierarchy, thus giving an error.

Every conflict implies a cycle could be formed, but not every cycle arises from a conflict. If a conflict is detected, cells are made not to correspond as hcells so the cycle is averted.

The processing order of cells in hierarchical LVS comparison is different from the hcell selection algorithm in the Query Server, so it is possible there is a difference in which hcells involved in a conflict are discarded. As a result, there can be cases in LVS comparison where conflicts are detected where the hcell list came from the Query Server. If this occurs, the following warning is given:

WARNING: Cell <name> cannot be made to correspond with cell <name> due to a hierarchical conflict.

If this occurs, you should examine the hierarchy of your designs and the hcell list to determine how to modify the hcell list to avoid the conflict.

Evaluation of Hcells for Netlist Extraction

The hcell analysis feature is typically used for LVS comparison when layout and source are both SPICE netlists. However, the hcell analysis can be used when the layout is geometric. This method has the disadvantage that the devices and connectivity from the layout have not yet been extracted, so comparison performance estimations are done entirely with the source netlist.

The `hcells::read` command scans the source netlist specified in the rule file in order to obtain source cell names. Then it reads the layout geometry database specified in the rule file to assess matching names and cells that are likely to cause performance problems during layout extraction. Finally, the source netlist is read again to perform hcell efficiency analysis using device counts in the source netlist.

Following is an example that assumes a rule file specifies a geometric [Layout System](#) and [Source System](#) SPICE.

```
$ calibre -qs
> hcells::hcells -file hcells
> hcells::automatch -on
> hcells::read -rules extract.rules -auto_expand PRESET
> hcells::add_matching_hcells
> hcells::print_hcells -file hcells.extract
```

This sequence of commands does the following things, in order:

- Reads in an hcell correspondence file called *hcells*. If no hcell file exists, you can omit `hcells::hcells`.
- Designates that cells having the same names in layout and source as potential hcells.
- Reads hcells based on the rule file and uses the internal automatic hcell expansion setting as a filter. If this filtering is too restrictive (too many hcells are being expanded, so your hcell list is too small), you can relax the `-auto_expand` setting.
- Adds cells to the current hcell list where layout and source names match, excluding layout cells that have been automatically expanded as high-cost.

If you want to filter the list further for the LVS comparison stage, use `hcells::clear_hcell` in a subsequent step.

- Prints the selected hcells to the file *hcells.extract*.

A similar flow is used by default through the `-genhcells` command line option. The `-genhcells` option uses the Query Server Tcl shell during the run to generate an hcell list that is stored in the Mask SVDB Directory. See “[Calibre nmLVS and Calibre nmLVS-H Command Line](#)” in the *Calibre Verification User’s Manual* for more details.

Hcell Selection Limitations

Hcell selection is useful feature when given a design with no hcell list or with a poor hcell list. However, limitations exist in automatic hcell selection.

These limitations should be noted:

- The Query Server does not consistently match cells having unlike names in layout and source. Hence, human intervention is required to ensure matches between cells with unlike names are correct and complete.
- The Query Server cannot match one cell in the source to many cells with different names in the layout.
- The hcell selection flow executes after netlist extraction where undesirable cell expansion can occur, so hcell candidates can be subsequently missed.
- The Query Server identifies hcell correspondence in an arbitrary order that can differ from how LVS comparison determines cell correspondence. This can lead to potential recursive hcell cycles being detected in LVS comparison, which results in warnings in the transcript. If the LVS transcript produces warnings like this:

WARNING: Cell <name> cannot be made to correspond with cell <name> due to a hierarchical conflict.

you should review the hcell list and remove the conflicting cells. Then try to determine if some other correspondence could be added to the list that does not result in a cycle conflict.

Related Topics

[Hcell and Hierarchical Analysis](#)

Netlist Hcell Analysis and Reporting Flow

The hcell analysis and hierarchy tree reports are useful for finding hcells for hierarchical LVS comparison. The report ranks cells in order of potential value as hcells (assuming that a matching hcell can be found).

The [hcells::print_hierarchy](#) command produces this report. This report is identical to the report produced by the Query Server [NETLIST REPORT HIERARCHY](#) command.

Hierarchy Report Output Generation

When identifying potential hcells, generate hierarchy reports for both the source and the layout if possible. (If both designs are not available, put together a rule file that compares the available design with itself.) The flow is basically as follows.

Given a rule file with the following lines:

```
LAYOUT PATH "lay.net"
LAYOUT PRIMARY "chip"
LAYOUT SYSTEM SPICE

SOURCE PATH "src.net"
SOURCE PRIMARY "chip"
SOURCE SYSTEM SPICE
```

you can execute the following in a terminal:

```
% calibre -qs
```

Enter the following commands interactively:

```
> hcells::read -rules rules
> hcells::clear_hcells
> hcells::print_hierarchy -file src.rep -source
> hcells::print_hierarchy -file lay.rep -layout
```

Read the reports and select an hcell pair (possibly more than one). Add each pair using the hcells::hcell command and regenerate the reports:

```
> hcells::hcell -layout layout_cell -source source_cell
> hcells::print_hierarchy -file src.new.rep -source
> hcells::print_hierarchy -file lay.new.rep -layout
```

Do this iteratively as necessary.

Regenerating the report identifies the next highest potential hcell candidates taking into account the new hcell. (Note that specifying additional hcells decreases the potential importance of other cells that share hierarchy above and below that hcell.)

See “[Hcell Analysis and Hierarchy Tree Report Format](#)” on page 242.

Tcl Shell Hcell Analysis Commands

Hcell analysis commands provide information for constructing preliminary hcell correspondence lists.

Table 3-1 shows the summaries for each supported command. The `hcells::` family of commands has equivalent Query Server commands from the non-Tcl interface. Whenever possible, the `hcells::` family of commands should be used in place of the historical equivalents. A legend for error codes is presented under “[Query Server Runtime Messages](#)” on page 631.

Table 3-1. Hcell Analysis Commands

Commands	Description
<code>hcells::add_matching_hcells</code>	Adds cells identified with the <code>hcells::automatch</code> or <code>hcells::placementmatch</code> commands to the current hcell list.
<code>hcells::automatch</code>	Identifies layout and source cells that match by name.
<code>hcells::clear_hcell</code>	Removes a corresponding cell pair from the current hcell list.
<code>hcells::clear_hcells</code>	Removes all corresponding cell pairs from the current hcell list.
<code>hcells::evaluate_current_hcells</code>	Controls hcell effectiveness evaluation.
<code>hcells::expand_unbalanced</code>	Controls unbalanced hcell expansion.
<code>hcells::hcell</code>	Adds a single hcell pair to the current hcell list.
<code>hcells::hcells</code>	Creates a current hcell list from an hcell file.
<code>hcells::placementmatch</code>	Identifies corresponding layout and source cells that match by placement count.
<code>hcells::print_hcells</code>	Writes the current list of hcells.
<code>hcells::print_hierarchy</code>	Writes a hierarchy tree report of cell statistics.
<code>hcells::read</code>	Reads in the layout and source netlists, along with any <code>Hcell</code> statements that may be in the rule file.
<code>hcells::select</code>	Selects hcells in the current hcell list based upon effectiveness thresholds and generates a report of the current hcell statistics.
<code>hcells::status</code>	Displays the settings of the hcell evaluation system.

hcells::add_matching_hcells

Hcell analysis command. Corresponding NETLIST command: **NETLIST ADD MATCHING HCELLS**.

Adds cells identified with the hcells::automatch or hcells::placementmatch commands to the current hcell list. Without this command, automatically matched hcells are not considered when producing the hcell analysis and hierarchy tree report.

Usage

hcells::add_matching_hcells [-help]

Arguments

- **-help**

Optional argument that shows the command usage.

Return Values

ERROR(134), ERROR(207)

[Query Server Runtime Messages](#).

Description

Adds automatically matched hcells to the current hcell list from either of the automatic matching commands.

When **hcells::automatch -strict** has been specified and only a single netlist is read (a single design), then ERROR(207) is given because two input designs are required for strict automatching.

Examples

This example assumes Layout System and Source System SPICE. This generates hierarchy reports for both designs based upon the current hcell list, which is comprised of any automatically matched hcells or hcells specified in the rules.

```
# automatically match hcells by name and port count
hcells::automatch -strict
# read netlists and any hcells from the rules
hcells::read -rules lvs.rules
# add automatched cells to the current hcell list
hcells::add_matching_hcells
# generate source hierarchy report
hcells::print_hierarchy -file source.cell.rpt -layout
# generate source hierarchy report
hcells::print_hierarchy -file source.cell.rpt -source
```

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

Current Hcell List

[hcells::automatch](#)

[hcells::placementmatch](#)

hcells::automatch

Hcell analysis command. Corresponding NETLIST command: [NETLIST AUTOMATCH](#).

Identifies layout and source cells that match by name.

Usage

hcells::automatch {-off | -on | -strict} [-help]

Arguments

- **-off**
Argument that specifies cells with names that match between source and layout are not used as hcells. This is the default.
- **-on**
Argument that specifies cells with names that match between source and layout are used as hcells. This setting should only be used when it is certain that source subcircuits match corresponding layout cells of the same name. The tool does not check that matched cells of the same name are also topologically equivalent.
- **-strict**
Argument that specifies cells are matched as with **-on** but must also have the same number of ports.
- **-help**
Optional argument that shows the command usage.

Return Values

ERROR(134)

[Query Server Runtime Messages](#).

Description

Causes cells with the same name in layout and source to be identified as candidates for the current hcell list. Using **-on** is similar to the **-automatch** command line option in LVS-H. Cell names that are the same are paired as hcells. Cell names that begin with ICV_[nnn] are reserved for internal use and are not used as hcells.

Cells matched by this command are added to the current hcell list using [hcells::add_matching_hcells](#) or [hcells::select](#). When [hcells::read](#) is used, both input designs must be SPICE for [hcells::select](#) to evaluate hcell efficiency.

The **-strict** option may not be used with [hcells::select](#) when the layout is geometric (two netlists are required as inputs), and it may not be used with [hcells::add_matching_hcells](#) when only one netlist is read (a single design).

When the `-strict` option is used prior to reading the layout and source designs, the Layout System is not SPICE, and `hcells::add_matching_hcells` has added any matched cells to the current hcell list, then the hcell pairs written by `hcells::print_hcells` receive a `STRICT` keyword appended to their line. These cells are treated as hcells during hierarchical LVS comparison if their port counts match.

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

hcells::clear_hcell

Hcell analysis command. Corresponding NETLIST command: [NETLIST CLEAR HCELL](#).

Removes a corresponding cell pair from the current hcell list.

Usage

hcells::clear_hcell -layout *layout_name* -source *source_name* [-help]

Arguments

- **-layout *layout_name***

Required argument and layout cell name to delete from the hcell list.

- **-source *source_name***

Required argument and source cell name to delete from the hcell list.

- **-help**

Optional argument that shows the command usage.

Return Values

NOK(49)

[Query Server Runtime Messages](#).

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

hcells::clear_hcells

Hcell analysis command. Corresponding NETLIST command: [NETLIST CLEAR HCELLS](#).

Removes all corresponding cell pairs from the current hcell list.

Usage

hcells::clear_hcells [-help]

Arguments

- **-help**

Optional argument that shows the command usage.

Return Values

ERROR(134)

[Query Server Runtime Messages](#).

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

hcells::evaluate_current_hcells

Hcell analysis command. Corresponding NETLIST command: **NETLIST EVALUATE CURRENT HCELLS**.

Controls hcell effectiveness evaluation.

Usage

hcells::evaluate_current_hcells {-yes | -no} [-help]

Arguments

- **-yes**
Required argument that specifies all cells in the current hcell list are evaluated for effectiveness along with cells selected by automatic means ([hcells::automatch](#) and [hcells::placementmatch](#)). This is the default.
- **-no**
Required argument that specifies all cells in the current hcell list remain in the list after evaluation.
- **-help**
Optional argument that shows the command usage.

Return Values

ERROR(148)

[Query Server Runtime Messages](#).

Description

Controls whether cells in the current hcell list undergo threshold effectiveness evaluation by [hcells::select](#).

When **-yes** is used, cells in the current hcell list are removed from the list if they do not meet the [hcells::select](#) evaluation thresholds. The **-no** option is useful when you know the current hcell list contains cells that you want to remain in the hcell list.

Related Topics

[Evaluation of Hcells for LVS Comparison](#)

[Evaluation of Hcells for Netlist Extraction](#)

[Tcl Shell Hcell Analysis Commands](#)

hcells::expand_unbalanced

Hcell analysis command. Corresponding NETLIST command: [NETLIST EXPAND UNBALANCED](#).

Controls unbalanced hcell expansion.

Usage

hcells::expand_unbalanced {-rules | -no | -yes} [-help]

Arguments

- **-rules**
Argument that causes unbalanced cell expansion to follow the [LVS Expand Unbalanced Cells](#) specification statement in the rule file (the default rule file setting is YES). This is the default.
- **-no**
Argument that prevents expansion of unbalanced cells regardless of settings in the rule file.
- **-yes**
Argument that causes expansion of unbalanced cells regardless of settings in the rule file.
- **-help**
Optional argument that shows the command usage.

Return Values

[ERROR\(134\)](#)

[Query Server Runtime Messages](#).

Related Topics

[Unbalanced Hcell Reporting](#)

[Tcl Shell Hcell Analysis Commands](#)

hcells::hcell

Hcell analysis command. Corresponding NETLIST command: [NETLIST HCELL](#).

Adds a single hcell pair to the current hcell list. The cells should be topologically identical, otherwise unpredictable problems can occur in LVS comparison.

Usage

hcells::hcell -source *source_name* -layout *layout_name* [-help]

Arguments

- **-source *source_name***
Required argument and source cell name to add to the hcell list.
- **-layout *layout_name***
Required argument and layout cell name to add to the hcell list.
- **-help**
Optional argument that shows the command usage.

Return Values

ERROR(134)

[Query Server Runtime Messages](#).

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

hcells::hcells

Hcell analysis command. Corresponding NETLIST command: [NETLIST HCELLS](#).

Creates a current hcell list from an hcell file. The current hcell list, if any, is cleared first.

Usage

hcells::hcells -file *filename* [-help]

Arguments

- **-file *filename***
Required argument and pathname of an hcell file.
- **-help**
Optional argument that shows the command usage.

Return Values

ERROR(101), ERROR(134), ERROR(146)

[Query Server Runtime Messages](#).

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

hcells::placementmatch

Hcell analysis command. Corresponding NETLIST command: [NETLIST PLACEMENTMATCH](#).

Identifies corresponding layout and source cells by placement count.

Usage

hcells::placementmatch {-off | -on | -loose} [-help]

Arguments

- **-off**
Argument that specifies placement matching does not occur. This is the default.
- **-on**
Argument that specifies placement matching occurs and pin counts must be the same to get a match.
- **-loose**
Argument that specifies placement matching occurs regardless of pin count.
- **-help**
Optional argument that shows the command usage.

Return Values

ERROR(134)

[Query Server Runtime Messages](#).

Description

Identifies hcell candidates for addition to the current hcell list if the number of times the cells appear in layout and source are identical and the number is unique. Cells matched by this command are added to the current hcell list using [hcells::add_matching_hcells](#) or [hcells::select](#).

Be aware this command can identify invalid hcell pairings, so its output should be carefully verified. If invalid pairings are made for an LVS run, high memory consumption, long run times, and false errors are possible.

See “[Cell Matches Using Placement Matching](#)” on page 246.

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

hcells::print_hcells

Hcell analysis command. Corresponding NETLIST command: [NETLIST REPORT HCELLS](#).

Writes the current list of hcells.

Usage

`hcells::print_hcells [-file filename] [-help]`

Arguments

- `-file filename`

Optional argument and output filename of an hcell list formatted in two columns: source and layout. If this option is omitted, the output is written as a string.

- `-help`

Optional argument that shows the command usage.

Return Values

`ERROR(101)`

[Query Server Runtime Messages](#).

Examples

This shows an interactive session that lists hcells identified from evaluating the rules and the source and layout netlists.

```
% calibre -qs
...
> hcells::read -rules rules
--- CALIBRE::HIERARCHICAL LAYOUT ANALYZER - Thu Feb 14 14:31:59 2019
...
> set hcells [hcells::print_hcells]
nand      nand
inv       inv
CellA    CellA
CellB    CellB
>
```

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Evaluation of Hcells for LVS Comparison](#)

[Evaluation of Hcells for Netlist Extraction](#)

hcells::print_hierarchy

Hcell analysis command. Corresponding NETLIST command: [NETLIST REPORT HIERARCHY](#).

Writes a hierarchy tree report of cell statistics.

Usage

hcells::print_hierarchy -file *filename* {-source | -layout} [-help]

Arguments

- **-file *filename***
Required argument that specifies the output file name of the hierarchy tree report.
- **-source**
Argument that creates a source netlist report.
- **-layout**
Argument that creates a layout netlist report.
- **-help**
Optional argument that shows the command usage.

Return Values

ERROR(101), ERROR(136), ERROR(143), ERROR(144)

[Query Server Runtime Messages](#).

Description

Writes a hierarchy tree report for the source or layout netlist to the *filename*. Either **-source** or **-layout** must be specified. The report statistics are based upon the current hcell list, if any.

Any hcell candidates identified by `hcells::automatch` or `hcells::placementmatch` do not appear in the report unless they have been added to the current hcell list using `hcells::add_matching_hcells` or `hcells::select`.

See “[Hcell Analysis and Hierarchy Tree Report Format](#)” on page 242 for details about the report.

Examples

```
# automatically match hcells by name and port count
hcells::automatch -strict
# read netlists and any hcells from the rules
hcells::read -rules lvs.rules
# add automatched cells to the current hcell list
hcells::add_matching_hcells
# generate a source hierarchy report
# a layout report can also be generated if the LAYOUT SYSTEM SPICE is used
hcells::print_hierarchy -file source.cell.rpt -source
```

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Evaluation of Hcells for LVS Comparison](#)

[Evaluation of Hcells for Netlist Extraction](#)

hcells::read

Hcell analysis command. Corresponding NETLIST command: [NETLIST READ](#).

Reads in the layout and source netlists, along with any Hcell statements that may be in the rule file.

Usage

```
hcells::read -rules rulefile [-source | -layout]
[-auto_expand {n / NONE | ALL | PRESET} [FILL]]
[-report {n | PRESET | ALL | NONE} [FILL]] [-help]
```

Arguments

- **-rules rulefile**

Required argument that specifies the LVS rule file.

- **-source**

Optional argument that specifies to read only the source netlist. The source netlist is read by default.

- **-layout**

Optional argument that specifies to read only the layout netlist. This option is only effective if the Layout System is SPICE. If the Layout System is SPICE, then the Layout Path is read by default.

- **-auto_expand {n | NONE | ALL | PRESET}**

Optional argument set that controls expansion of HIGH COST hcells in the layout geometry database. This option functions similarly to the [LVS Auto Expand Hcells](#) rule file statement and overrides that statement when present in the rule file. This option applies only when reading a layout geometry database.

NONE is the default and causes no hcells to be expanded. ALL expands all hcells that are subject to expansion. PRESET uses internal heuristics to determine the best hcells to expand, if any. The *n* parameter is a positive integer less than 100 that specifies a threshold value as a percent. Hcells that exceed the threshold are expanded.

- **-report {n | PRESET | ALL | NONE}**

Optional argument set that identifies HIGH COST hcells in the layout geometry database and reports them in the transcript. This option functions similarly to the LVS Auto Expand Hcells rule file statement REPORT keyword and overrides that statement when present in the rule file. This option applies only when reading a layout geometry database.

PRESET is the default and uses internal heuristics to determine the hcells to report, if any. NONE specifies no hcells are reported. ALL reports all hcells that are subject to reporting. The *n* parameter is a positive integer less than 100 that specifies a threshold value as a percent. Hcells that exceed the threshold are reported.

- **FILL**

Optional keyword that causes cells interacting with fill shapes, multi-patterning “color” shapes, and similar geometry, and that could adversely affect circuit extraction performance, not to serve as hcells. This option should be used if the aforementioned geometry is used as an essentially flat overlay of the original design hierarchy. This keyword applies to circuit extraction only.

When used with the -report option, cells meeting the aforementioned criteria are reported in the transcript.

- **-help**

Optional argument that shows the command usage.

Return Values

ERROR(134), ERROR(135), ERROR(145)

[Query Server Runtime Messages](#).

Description

Causes the source and layout netlists specified in the **rulefile** to be read into memory for analysis. Additionally, any **Hcell** statements in the rule file are read and added to the current hcell list.

When neither -source nor -layout is specified, this command reads the source netlist and the layout netlist if Layout System SPICE is used (if not, layout cells are not evaluated for reporting purposes). Hcell analysis finds corresponding cells based upon the matching conditions specified: automatch, placementmatch, or explicit cell matching list.

The -source and -layout options cause this command to read only the source or layout netlist, respectively, and use it as both source and layout. The hcell analysis is based only upon the specified netlist.

When the [Layout System](#) is geometric and automatching is requested, the layout database is examined for names that appear in the source netlist so that automatch hcell candidates can be appropriately identified.

The -auto_expand and -report options function similarly to the [LVS Auto Expand Hcells](#) specification statement options and are used for hcell expansion reporting and control.

The FILL keyword is useful for large designs containing an essentially flat geometry overlay of huge numbers of shapes on top of the original layout design. Using the FILL option in such situations can improve circuit extraction performance dramatically.

Examples

This is an invocation from the Tcl shell. Both layout and source are SPICE.

```
> hcells::read -rules calibreLVS.rul
Initializing LVS ...
READING layout ...
READING source ...
Identifying CORRESPONDING cells ...
CORRESPONDING Cells Identified.
Adding GLOBAL elements ...
Resolving DEEP SHORTS ...
Resolving HIGH SHORTS ...
    High Shorted pins VINP VSS of layout cell opamp2.
    High Shorted pins OUTB VSS of layout cell Switch_new.
    High Shorted pins OUTB VSS of source cell Switch_new.
    High Shorted pins VINP VSS of source cell opamp2.
HIGH SHORTS resolved.
Deleting TRIVIAL PINS ...
    Removed pin COM of layout cell Switch_new.
TRIVIAL PINS deleted.
```

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

[Evaluation of Hcells for LVS Comparison](#)

[Evaluation of Hcells for Netlist Extraction](#)

hcells::select

Hcell analysis command. Corresponding NETLIST commands: [NETLIST SELECT HCELLS](#) and [NETLIST EVALUATION THRESHOLD](#).

Selects hcells in the current hcell list and any automatically matched cells. Generates a report of the current hcell statistics.

Usage

```
hcells::select -file filename [-threshold THIC_threshold [flat_threshold]] [-help]
```

Arguments

- **-file *filename***
Required argument and evaluation report pathname.
- **-threshold *THIC_threshold* [*flat_threshold*]**
Optional argument and evaluation thresholds expressed as percentage values from 0 to 100. The *THIC_threshold* represents the computational savings based upon total hierarchical instance count (THIC) reduction. The default *THIC_threshold* is 30 (percent). The *flat_threshold* represents the ratio of a cell's internal instance count (at the top level of the cell) to the entire design's flat instance count, in percent. The default *flat_threshold* is 1 (percent).
- **-help**
Optional argument that shows the command usage.

Return Values

ERROR(134), ERROR(136), ERROR(148), ERROR(149), ERROR(206)

[Query Server Runtime Messages](#).

Description

Generates an hcell evaluation report and sets the current hcell list based upon the -threshold settings. The -threshold settings are only used as hcell selection criteria if hcell::select is used and [hcells::evaluate_current_hcells -yes](#) is enabled, which it is by default.

Candidate cell pairs identified by [hcells::automatch](#) and [hcells::placementmatch](#) are added by hcells::select to the current hcell list. When hcells::read is used, input designs must be SPICE or hcells::select cannot evaluate hcell efficiency.

If hcells::automatch -strict is active, and [hcells::read -layout](#) or [-source](#) is used or the Layout System is not SPICE, then ERROR(206) is given because hcell selection depends upon strict automatching occurring first, and this can only be performed with two input netlists.

When -threshold is unspecified, cells are added to the current hcell list until, collectively, the remaining potential THIC savings is 30 percent. Also, cells that contain at least 1 percent of the

total flat instance count of the design are selected as hcells regardless of any THIC considerations. Increasing either the *THIC_threshold* or *flat_threshold* values from their defaults tends to reduce the number of selected hcells.

The report is written to the *filename*. See “[Hcell Evaluation Report Format](#)” on page 240. If *hcells::evaluate_current_hcells -no* is set, then *hcells::select* does not adjust the current hcell list, but the report is generated, as usual.

Examples

This script may be used when the layout is SPICE and there is no pre-existing hcell list.

```
hcells::automatch -on
hcells::read -rules rules
hcells::print_hierarchy -layout -file auto_layout.report
hcells::print_hierarchy -source -file auto_source.report
hcells::select -file auto_select.report
hcells::print_hcells -file auto_hcells
qs::quit
```

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Evaluation of Hcells for LVS Comparison](#)

[Evaluation of Hcells for Netlist Extraction](#)

hcells::status

Hcell analysis command. Corresponding NETLIST command: [NETLIST STATUS](#).

Displays the settings of the hcell evaluation system.

Usage

hcells::status

Arguments

None.

Return Values

ERROR(134), ERROR(135), ERROR(145)

[Query Server Runtime Messages](#).

Examples

```
> hcells::status
automatch:          on
placementmatch:    on
THIC threshold:   30%
Flat instance count threshold 1%
evaluate current hcells: yes
hcell file:        cells.list
expand unbalanced: from rules
read from rule file:  calibreLVS.rul
```

Related Topics

[Tcl Shell Hcell Analysis Commands](#)

[Current Hcell List](#)

Hcell Evaluation Report Format

Output for: [hcells::select](#) or [NETLIST SELECT HCELLS](#).

Hcell analysis report showing the effectiveness of hcells.

Format

Column entries as follows:

Total Hier. Instance Count Layout / Source — The numbers of primitive devices and cell placements in the layout or source design cells after the cells are flattened based upon rule file settings and internal heuristics, assuming the hcell correspondences include the listed cell or pair and all previously listed cells or pairs. Therefore, the numbers decrease moving down a column. If only the source is analyzed, then the Layout column is empty.

Instance Count In This Cell Layout / Source — The counts of instances at the primary level of each cell in the layout or source, as determined by internal heuristics. By default, when this number is at least one percent of the total flat instance count of the design, the cell is selected as an hcell regardless of the contribution the cell might make to THIC savings.

Saved By Cell — Percentage of Total Hierarchical Instance Count (THIC) saved by the addition of the candidate hcell(s), assuming all previously listed hcells were in place prior to the addition.

This calculation is made for rows 2 and greater by taking the sum of the values from columns 1 and 2 of the current row, subtracting that from the sum of the corresponding values in the preceding row, dividing by the latter sum, and multiplying by 100.

For example, see the fifth data row of the table in the Examples section that follows. The fifth row's value for column 2 is 66820 (column 1 is empty in this case), and the preceding row's value is 88302. So the corresponding value for Saved By This cell is:

$$(88302 - 66820) / 88302 * 100 = 24\%$$

Total Savings So Far — Percentage of THIC saved by hcells preceding and including the current row in the report table.

This calculation is made for rows 2 and greater by taking the sum of the values from columns 1 and 2 of the current row, subtracting that from the sum of the corresponding values in the first row, dividing by the latter sum, and multiplying by 100.

Again, refer to the fifth data row of the table in the Examples section that follows. The fifth row's value for column 2 is 66820, and the first row's value is 212990. So the corresponding value for Total Saving So Far is: $(212990 - 66820) / 212990 * 100 = 69\%$.

Potential Remaining Savings — Percentage of THIC that could be saved by adding other hcells. This is an internally calculated estimate of possible THIC savings based upon what has

been saved by hcells preceding and including the current row. By default, when this value reaches 30% or less, no additional hcells are selected.

Layout / Source Cell Name — Cell names of corresponding cells in layout and source. If only the source is analyzed the Layout Cell column is empty.

Parameters

None.

Examples

This example shows an analysis of the source netlist alone. The cell name columns on the right side of the report have been removed to conserve space.

C A L I B R E L V S HCELL EVALUATION REPORT						
Total Hier.	Instance Count	Instance Count	Saved	Total	Potential	
Layout	Source	Layout	In This Cell	By This	Savings	Remaining
-	212990	-	-	0.0%	0.0%	92%
-	135524	-	17	36%	36%	87%
-	111494	-	8	18%	48%	84%
-	88302	-	28	21%	59%	80%
-	66820	-	22	24%	69%	73%
-	60306	-	6	9.7%	72%	70%
-	55510	-	4	8%	74%	68%
-	50786	-	65	8.5%	76%	65%
-	46092	-	6	9.2%	78%	61%
...						
-	28596	-	6	5.4%	87%	37%
-	27346	-	4	4.4%	87%	34%
-	26330	-	10	3.7%	88%	32%
-	25332	-	28	3.8%	88%	29%

If the Saved By This Cell column value is null or zero, then a cell was selected as an hcell because its internal instance count (column 3 or 4) met or exceeded the threshold percentage of the total flat instance count in its design. By default, this threshold is one percent.

Related Topics

[Netlist Hcell Analysis and Reporting Flow](#)

[Hcell List Management Using Standard Commands](#)

Hcell Analysis and Hierarchy Tree Report Format

Output for: [hcells::print_hierarchy](#) and [NETLIST REPORT HIERARCHY](#)

Hcell analysis statistics report used for selection of hcell list pairs. It provides an estimate of the percentage in memory savings if a cell is used as an hcell, which is a useful evaluation criterion.

Format

The report is divided into three major sections: the header, the hierarchy table, and the hierarchy tree.

The header of the file contains a description of the file's columns. This shows a header for a layout hierarchy report. The numbered sections correspond to column numbers in the hierarchy table. A source hierarchy report contains a subset of the information under item 10 under Column Definitions.

```
=====
      C A L I B R E   L V S
      HCELL ANALYSIS AND HIERARCHY TREE REPORT
=====

Top Level Data
-----
Netlist file: layout.sp
Automatch Hcells is on
Total Flat Device Count (TFDC)      =      248921 (count of all devices
represented flat)
Total Hierarchical Instance Count (THIC) =      18941 (count of all
devices expanded to hcells)

Potential Hcell Analysis
-----
This report presents information useful for selecting LVS hcells for a
netlist.
Cells are presented in order of potential memory savings if the cell is
used as an hcell.
```

Column Definitions

Flat - The following columns present statistics concerning the flattened design.

(1) Instances of this Cell

The number of times the cell is instantiated throughout the entire flattened design.

(2) Devices in this Cell (FDC)

The number of devices in this cell when its entire contents are flattened (Flat Device Count).

(3) Total Device Contrib. (1)x(2)

The number of devices this cell contributes to the total flat device count.

(4) % Total Device Contrib. ((3)/TFDC)*100

Column (3) represented as a percentage of total flat device count.

With Hcells - the following columns present statistics taking into account the current hcells and automatch setting.

(5) Instances of this Cell

The number of times the cell is instantiated within all existing hcells (always 1 for an hcell).

(6) Instances in this Cell (HIC)

The number of instances that would be in this cell if all non-hcells inside it were expanded (Hierarchical Instance Count).

(7) Total Instance Contrib. (5)x(6)

The number of instances this cell contributes to the total hierarchical instance count.

(8) % Total Instance Contrib. ((7)/THIC)*100

Column (7) represented as a percentage of total hierarchical instance count.

(9) % Memory Savings

Expected memory savings if this cell is used as an hcell.

(10) Cell Name

Cell names.

* designates leaf cells (all contents are devices).

+ designates current hcells.

= designates cells with the same name in layout and source.

The hierarchy table immediately follows the header. An example is shown later in this section.

The hierarchy tree follows the hierarchy table. It follows this general form:

```

Hierarchy Tree
-----
Devices  Cell Name
in this
Cell
=====
<count> <topcell> (level=0)
<count> . <cell> (x<instance count>) (level=1)
<count> . . <cell> (x<instance count>) (level=2)
...

```

Parameters

None.

Examples

The following is an excerpt of a hierarchy table from an actual report. It is shown divided into two parts in order to fit the page width.

<----- Flat ----->			
(1)	(2)	(3)	(4)
Instances of this Cell	Devices in this Cell	Total Device Contrib. (FDC)	% Total Device Contrib. (1)x(2) /TFDC
624	2	1248	0.6
256	16	4096	1.9
112	4	448	0.2
768	4	3072	1.4
12288	6	73728	35
34	2	68	0.0
256	2	512	0.2
32	4	128	0.1
1	8	8	0.0
91	6	546	0.3

<----- With Hcells ----->						
(5) Instances of this Cell	(6) Instances in this Cell (HIC)	(7) Total Instance Contrib. (5)x(6)	(8) % Total Instance Contrib. (7)/THIC	(9) Memory Savings	(10) Cell Name * (leaf cell) + (hcell) = (same name)	
312	2	624	3.5	1.7	*	AMcell
16	16	256	1.4	1.2	+	BIcell
56	4	224	1.2	0.9	*	NMcell
48	4	192	1.1	0.8	*	tribuf
16	6	96	0.5	0.4	*	BMcell
17	2	34	0.2	0.1	*	inv_addr
16	2	32	0.2	0.1	*	inv_wda
2	4	8	0.0	0.0	*	nand_CI
1	8	8	0.0	0.0	++	BUF4
1	6	6	0.0	0.0	++	NAND3X1

Column 9 always shows 0% savings for cells that are in the current hcell list.

Here is an excerpt of the hierarchy tree:

Hierarchy Tree	
Devices in this Cell	Cell Name
212990	top (level=0)
1942	. ADC_5bit_sc_5 (x1) (level=1)
1	. . MN (x1) (level=2)
1	. . MP (x1) (level=2)
215	. . ADC_5b_Tencode_sc (x1) (level=2)
1	. . . MN (x105) (level=3)
1	. . . MP (x110) (level=3)
1110	. . ADC_5b_thermo (x1) (level=2)
28	. . . DFFNX1 (x33) (level=3)
1 MN (x14) (level=4)
1 MP (x14) (level=4)
6	. . . nand3_1 (x31) (level=3)
1 MN (x3) (level=4)
1 MP (x3) (level=4)

In some cases the following line may appear due to unbalanced hcell placement counts in the two designs.

* expanded due to unbalanced hcell placements.

Related Topics

[Netlist Hcell Analysis and Reporting Flow](#)

[Hcell List Management Using Standard Commands](#)

Unbalanced Hcell Reporting

Hcells are said to be unbalanced when the cell instance counts differ in layout and source for a given hcell.

By default, Calibre nmLVS-H expands unbalanced hcells (see [LVS Expand Unbalanced Cells](#) in the *SVRF Manual*). The Query Server follows the rule file by default with regard to handling unbalanced hcells. However, the rule file can be overridden by [hcells::expand_unbalanced](#) and [NETLIST EXPAND UNBALANCED](#).

The [hcells::print_hierarchy](#) and [NETLIST REPORT HIERARCHY](#) commands take unbalanced cell expansion into account when reporting instance and device counts, and when calculating memory savings. The counts from unbalanced hcells are reported in any cell that has expanded placements of hcells. In addition, hcells that have all instances expanded appear with 0 in columns (5) and (7) of the Hierarchy Analysis report (the “Instances of this Cell” column and the “Total Instance Contrib.” column in the “With Hcells” section). Such cells are not marked as hcells in the report. In the Hierarchy Tree section, expanded placements of hcells are indicated with an asterisk (*).

The [hcells::select](#) and [NETLIST SELECT HCELLS](#) command takes unbalanced cell expansion into account when ranking hcells, calculating memory savings, and calculating instance counts.

Cell Matches Using Placement Matching

The Query Server Hcell Analysis functionality includes the ability to create hcell matches between cells having a similar number of placements in the layout and source.

The heuristic is called *placementmatch* in the [hcells::status](#) and [NETLIST STATUS](#) command outputs and is controlled by the [hcells::placementmatch](#) and [NETLIST PLACEMENTMATCH](#) commands.

Placement matching works as follows: If cell A in the layout is placed exactly 1000 times, and cell B in the source is placed exactly 1000 times, and no other cell in layout or source is placed exactly 1000 times, then the cells A and B are selected as hcells for analysis. By default, cells matched using this feature must have the same number of pins in layout and source. The count used is the flat instance count, column (1), in the hcell analysis report. The search for matched hcells occurs among remaining non-hcells after any explicit hcells and automatch hcells are identified.

Chapter 4

Key Concepts for Standard Mode Operation

The standard mode interface (non-Tcl) is the one used since the inception of the Query Server. In general, the Tcl shell interface is preferred because it is programmable, multithreaded, and receives greater attention for new features. Standard mode is maintained mostly for backward compatibility.

There are a number of foundational ideas you should know when using the Query Server in standard mode. You should familiarize yourself with them before using the tool.

calibre -query	248
Acknowledgment Messages.....	252
Responses and Response Files	253
Command Context in the Standard Query Server.....	255
Client Contexts.....	257

calibre -query

For the Tcl shell, see “[calibre -qs](#)” on page 30.

Calibre Query Server standard command line.

Usage

```
calibre [-cb] [-nowait | -wait n] [-lmconfig licensing_config_filename]
        [-query_input query_file] -query [svdb_directory] [layout_primary]
```

Arguments

The order of the final three command line arguments is important to prevent confusion of the parameter names with other specified options.

- **-query**

A required argument that starts the Query Server. This argument should follow all other options except *svdb_directory* and *layout_primary*.

- *svdb_directory*

An optional argument that specifies the pathname to the SVDB directory. This directory is generated by the [Mask SVDB Directory](#) rule file statement during a Calibre tool run.

- *layout_primary*

An optional argument that specifies the top-cell name as specified by the [Layout Primary](#) specification statement in the rule file. You must specify this cell name when the *svdb_directory* contains data from several different primary cells. When you do not specify *layout_primary*, the Query Server seeks a directory name ending with phdb in the *svdb_directory*.

The following optional arguments should appear before **-query**.

- **-query_input query_file**

An optional argument that specifies the pathname to a text file containing a Query Server command script.

- **-cb**

An optional argument that specifies to use the Calibre Cell/Block license. Refer to the [Calibre Administrator's Guide](#) for license information.

- **-nowait**

An optional argument that causes the tool to exit if a license is not available.

- **-wait n**

An optional argument that places a limit on the total time in minutes that Calibre queues for a license. For example, this command:

```
calibre -wait 5 -query svdb
```

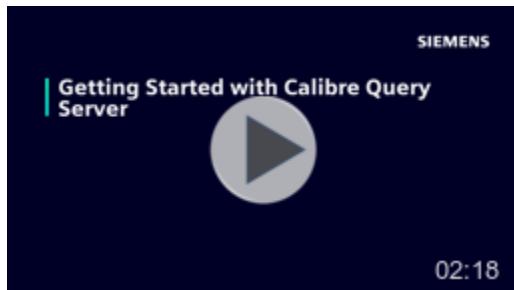
queues on a license for five minutes and exits if a license is not available in that time frame.

- `-lmconfig licensing_config_filename`

An optional argument that enables a loop search for default or substitute licenses. The loop search is used rather than the default license acquisition behavior. The *licensing_config_filename* argument specifies a configuration file that controls how the license search occurs. Refer to the *Calibre Administrator's Guide* for more information.

Description

This command starts the Calibre Query Server in standard mode. The server accesses the PHDB (persistent hierarchical database) and XDB (cross-reference database) in an SVDB (standard verification database) generated by the **Mask SVDB Directory** specification statement in the rule file. The QUERY and CCI options of the Mask SVDB Directory statement are typically used.



The Query Server commands do the following general things:

- Control connections to the client.
- Establish the contexts for reporting.
- Request design and run information from the PHDB.
- Control where output is directed.

By default, the server communicates with the client application by reading STDIN¹ to receive commands, but the server can also be queried from a script. In either mode, commands are executed immediately as they are issued to the server; hence, command scripts are executed from top to bottom unless otherwise indicated.

Query commands are issued in the following modes.

Commands from a script — In this mode, the tool is controlled by a text file. For example:

```
calibre -query_input server_script -query svdb
```

1. The Query Server standard mode does not have an application programming interface of its own. However, the Query Server can be accessed through Perl scripts or other scripting languages. In particular, Perl provides standard input/output support through its IPC::Open2 library (see <http://www.cpan.org>). The Query Server flushes all output at the end of each command response in order to facilitate communication with scripting languages.

where *server_script* contains valid Query Server commands.

You can also use a “here” document with Query Server commands, like this:

```
calibre -query svdb <<!
cells layout
cells source
!
```

Interactive commands from a user — In this mode, commands are issued by the user to the Query Server from the command line to examine a PHDB. For example, you can enter the following:

```
calibre -query svdb
```

The server responds:

```
INITIATING HDB QUERY SERVER:
-----
OK: Ready to serve.
```

At this point, commands may be entered interactively in the shell.

Commands from an interface — In this mode, a client interface other than a script interacts with the Query Server. The client must be designed to accept the output of the Query Server and to issue commands in the forms the Query Server expects.

The Query Server generates three types of outputs:

Acknowledgment Messages — A string returned by the Query Server to indicate that the command has been received and to communicate about the success or failure of command processing.

Responses — Formatted information returned by the server. Responses are written to STDOUT immediately following the acknowledgment by default, or they can be written to response files from which the client application can read. Not all commands return responses. See “[Responses and Response Files](#)” on page 253.

CCI Files — Output files used to pass connectivity data to downstream tools. For more information on CCI files, refer to “[Calibre Connectivity Interface](#).”

Examples

Example 1

This excerpt shows an invocation of the Query Server from the command line, followed by the issuing of the NET NAMES, NET LAYERS, and TERMINATE commands.

```
% calibre -query svdb
...
-----
----- CALIBRE::HDB QUERY SERVER - EXECUTIVE MODULE -----
-----
INITIATING HDB QUERY SERVER:
-----
    OK: Ready to serve.
NET NAMES
Net_Names 1000
Nets:
0 0 4 Jun 1 13:10:30 2012
1
GND
PWR
4
END OF RESPONSE
0 0 0 Jun 1 13:10:30 2012
    OK.
NET LAYERS PWR
Net_Layers 1000
Layers:
0 0 8 Jun 1 13:11:38 2012
contact
ntap
nwell
metal1
nsd
psd
via
metal2
END OF RESPONSE
0 0 0 Jun 1 13:11:38 2012
    OK.
TERMINATE
    OK: Terminating.

HDB QUERY SERVER terminated: CPU TIME = 0  REAL TIME = 97  LVHEAP = 0/5/69
MALLOC = 93/93/93
```

Example 2

Given that your rule file contains this:

```
LAYOUT PRIMARY top
MASK SVDB DIRECTORY svdb QUERY
```

The following commands provide layout and source information.

To run connectivity extraction and hierarchical comparison in one step, do this:

```
calibre -spice lay.net -turbo -lvs -hier -hcell cells rules
```

To run connectivity extraction and Calibre nmLVS-H in two steps:

```
calibre -spice lay.net -turbo rules.extract
calibre -lvs -hier -hcell cells -layout lay.net rules.compare
```

Running connectivity extraction as a separate step allows you to determine if there are connectivity issues that need to be addressed before going to comparison.

To start the Query Server, do this:

```
calibre -query svdb topcell
```

If you start the Query Server and the comparison step was not run when the SVDB was generated, you will see this message:

NOTE: Cross reference commands will be unavailable since a valid cross reference (XDB) database was not found.

You will also see this message if you use a [Mask SVDB Directory](#) statement with an option that does not generate an XDB, such as the PHDB option.

Acknowledgment Messages

Standard Query Server commands elicit an acknowledgment message upon execution. An acknowledgment is a single string of text beginning with three spaces and terminated by a carriage return character. Acknowledgments are documented for all commands that produce them in a subsection of the same name in the command reference pages.

Since some acknowledgment lines contain pathnames and other design information, there is no set limit on the length of an acknowledgment line. The client must be prepared to accept acknowledgments of arbitrary length and to reformat them for display if necessary. Here are some examples (the // comments are not part of the acknowledgment):

```
OK. // command succeeded
OK: 7 // command succeeded; returned value 7
NOK(1): There are no pins on layout net VCC. // info not available
ERROR(108): Client id 0 can never be disconnected. // command failed
```

The four previous examples illustrate the four types of acknowledgment:

- **OK.** — This acknowledges that the command succeeded. Any additional information generated by the command is returned in a response.
Responses are generated by a command that returns design data. Responses are different from acknowledgments. A command that generates responses does so only if its acknowledgment contains OK. See “[Responses and Response Files](#)” on page 253.

- **OK: <string>** — This acknowledges that the command succeeded and returns some information produced by the command. This form of acknowledgment is used by commands that generate a small amount of information on a single line.
- **NOK(<number>): <reason>** — This acknowledges that the command failed to produce the requested information. The <number> is an integer and the <reason> is a sentence explaining the reason for failure. The <number> is provided for a program to identify the message without parsing the <reason>. This type of acknowledgment is only used for queries that request design information. See “[Failure Messages](#)” on page 638.
- **ERROR(<number>): <reason>** — This acknowledges when a problem arises such that the command cannot be performed. The <number> is an integer and the <reason> is a sentence explaining the reason for the error. The <number> is provided for a program to identify the message without parsing the <reason>. Errors may represent a failure of the environment (such as the inability to open or write to the response file) or misuse of the communication channel by the application (such as an unknown command or a command that is inappropriate and could have been avoided). See “[Error and Warning Messages](#)” on page 631.

Responses and Response Files

A response is an output from a standard command that contains an extended form of design data. A response file is an ASCII file containing responses that can be read by a client.

A response contains the design information returned by a query command. Not all standard commands produce them. For the ones that do, a Response subsection appears on the command’s reference page.

Certain responses are similar to ASCII DRC results database format as found under “[ASCII nmDRC Results Database Format](#)” in the *Calibre Verification User’s Manual*. The coordinate system used in a response is determined by the cell context in which the command was invoked.

Responses are formatted as follows:

- Header line followed by one or more check sections. The header line contains a cell name followed by the precision of the design.
- Check sections beginning with a check title line containing the name of the check, which may contain embedded white space.
- Count line containing the number of shapes currently in the check, the number of shapes originally in the check, the number of text lines in the check, and the date. The two geometry counts are always equal. However, response readers should ignore the original geometry count, which is for internal use.
- Zero or more text lines, the number of lines having been given in the text line count field of the count line.

- Zero or more polygon references, the number of polygons having been given in the current geometry count field of the count line. Each polygon reference consists of a polygon header line containing the character p, a polygon number (in responses the polygons are numbered sequentially starting from 1), and the vertex count of the polygon.
- Vertex lines contain the vertices of a polygon in counterclockwise order, one vertex per line, with each line containing the x followed by the y coordinates of the vertex.
- End of response check title and count lines. Because it is always present and does not change, the END OF RESPONSE line may be omitted from the documented command descriptions.
- Coordinates are in database units as represented in the Calibre databases (not necessarily the same as the physical layout). When used, magnification factors in the rule file are applied to the coordinates.

The response format is shown next. Items in angle brackets are separated by spaces and contain no white space, except for <check_name> and <text_line>. The comments following the // characters are for information only. They do not appear in the actual responses.

```
<response_name> <precision>                                // leading line
<check_name>                                                 // first check title line
<curr_count> <orig_count> <text_count> <date>           // count line
<text_line>                                                 // first text line
...                                                       // more text lines
p <poly_number> <vertex_count>                           // polygon header line
<x> <y>                                                 // first vertex line
...
...
...
END OF RESPONSE                                         // end of check data
0 0 0 <date>                                              // terminating line
```

A response file is an alternate destination for responses. You configure the Query Server to direct output to a response file (instead of STDOUT) through the [RESPONSE FILE](#) command.

Each time a response is generated, the server opens the current response file, writes the response into the file, and closes the file. This has several implications. First, any previous contents of the file are overwritten by the new response. Second, after receiving the “OK” acknowledgment, the application can rename or delete the file. The server uses the file again for the next response. The application can establish a new response file at any point. Thus, it can have each response directed to a different response file or can have all responses directed to the same file. In the latter case, new responses overwrite previous responses unless the application renames the file between responses.

[Acknowledgment Messages](#) are not sent to response files.

Command Context in the Standard Query Server

Because of the hierarchical nature of the data on which the Query Server typically operates, the client application must provide the Query Server with a context within which to interpret command arguments.

The context defines such things as these:

- The cell (top cell in flat applications) about which queries are currently being made is the *query cell*. You define the query cell using the **CONTEXT** command.
- The cell having the coordinate system in which the results are returned is called the *viewing cell*. The viewing cell must either be the query cell or a cell containing an instance of the query cell. By default, the query cell is the viewing cell.
- The interpretation of pathnames.
- The coordinate space used for reporting results.
- The output file location.

The distinction between query cell and viewing cell is necessary because of the hierarchical nature of the data on which the Query Server operates. In commands and Query Server responses, pathnames are always relative to the query cell. For example, if the query cell is ADDER2 and the command contains the net path X2/CLK, then the net being specified is the net named CLK in the cell instance named X2 in the cell named ADDER2.

Geometric Coordinates

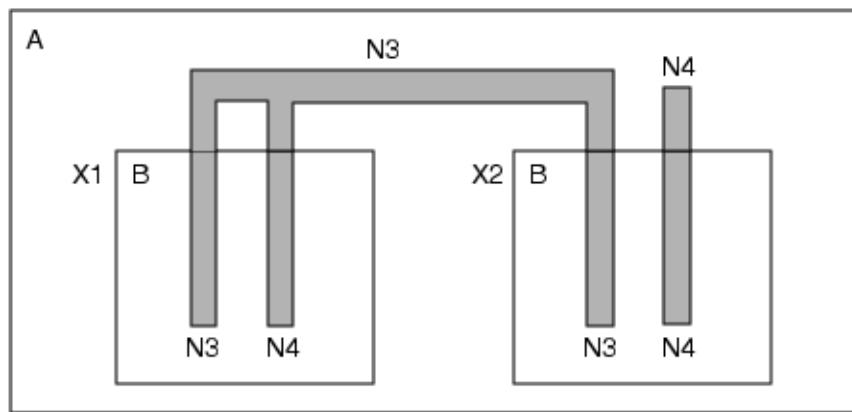
In commands that require the specification of a geometric location using (x,y) coordinates, the coordinates are those of the viewing cell and are specified in user-units.

When a cell contains instances of lower-level cells, the parent cell and each of the lower-level cells each have their own coordinate spaces. Thus, geometric results can be returned in one of the following coordinate systems:

- The coordinate system of the query cell. (The viewing cell is the same as the query cell.)
- The coordinate system of a cell containing an instance of the query cell. (The viewing cell contains an instance of the query cell.)

Figures 4-1 through 4-4 illustrate these concepts. The top-level cell is A. It contains two instances, X1 and X2, of cell B. Both cells contain nets N3 and N4 as shown in the figures.

Figure 4-1. Top Cell is A

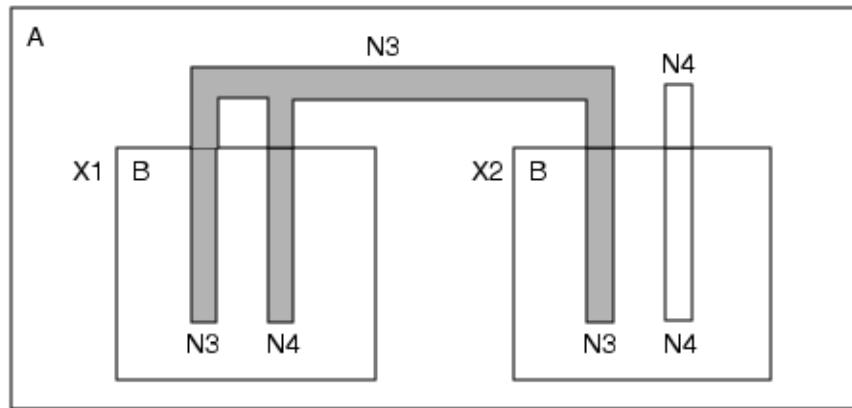


Consider the query **NET SHAPES N3** in the following three examples.

Example 1

The viewing and query cells are both A (see [Figure 4-2](#)). In this case, you are viewing cell A and asking about objects relative to cell A. Thus, N3 in the query refers to net N3 of cell A. The results are mapped into cell A. The Query Server returns the coordinates of the gray shapes in the coordinate system of cell A.

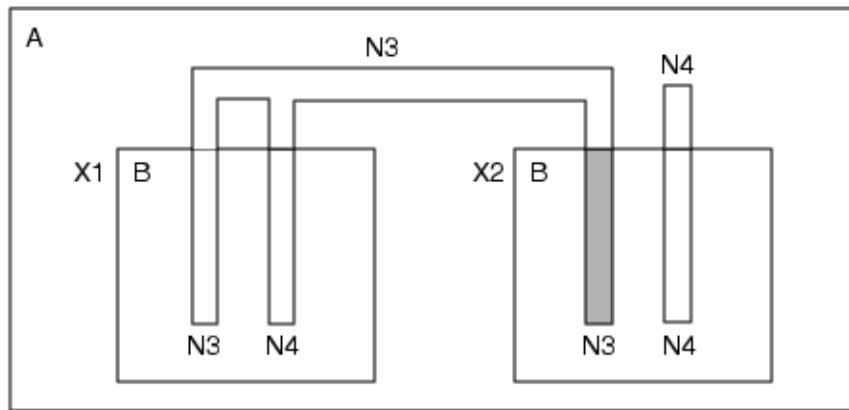
Figure 4-2. Viewing and Query Cells are Both A



Example 2

The viewing cell is A, the query cell is B and the query instance is X2 (see [Figure 4-3](#)). In this case, you are viewing cell A but making queries relative to cell B, and you expect the results to be mapped from instance X2. Thus, N3 in the query refers to net N3 of cell B. The Query Server returns the coordinates of the gray shape in the coordinate system of cell A.

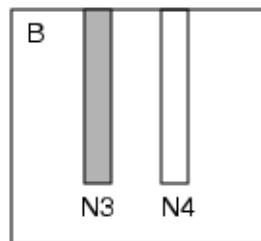
Figure 4-3. Viewing Cell is A, Query Cell is B, Query Instance is X2



Example 3

The query and viewing cells are both B (see [Figure 4-4](#)). In this case, you are viewing cell B and asking about objects relative to cell B. Thus, N3 in the query refers to net N3 of cell B. The results are to be mapped into cell B. The Query Server returns the coordinates of the gray shape in the coordinate system of cell B.

Figure 4-4. Viewing and Query Cells are Both B



Client Contexts

The Query Server standard mode is designed for multiple client contexts. From the server's point of view, client contexts are created, go through phases of being active and inactive, and eventually are deleted. Exactly one client context is active at any given time and is called the current client.

Note

- In most cases, you need not be concerned with client context. This topic would concern you if you are building your own interface to the Query Server. This is rare. In most cases, a script passed with the -query_input option is all you need.

Definition: The *current client* is the client application that is active at a specific time.

The server maintains a table of information about each client context. Commands exist which allow a client to modify its client table information.

The following table describes the data stored as the current context for each client:

Table 4-1. Client Table

Entry	Description
ID	Client ID number of this client context. See CONNECT .
Query Cell	The name of a cell (not a cell instance) about which queries are currently being made. The default value is the name of the top cell of the design. See CONTEXT .
Viewing Cell	The name of a cell (not an instance) with coordinate space used for the return of geometric results. The value must either be the query cell or a cell containing an instance (possibly separated by several levels of hierarchy) of the query cell. See CONTEXT .
Query Instance	If the viewing and query cells are the same, this value is NULL. Otherwise, it is the pathname relative to the viewing cell of an instance of the query cell.
Response Mode	The method in which the server responds: direct (STDOUT) or indirect (response file). The default value is NULL. The value is NULL if this client is receiving direct responses, otherwise the pathname of the user-specified response file for this client. See RESPONSE DIRECT and RESPONSE FILE .
MARKER SIZE	The width in user-defined units of the geometric squares used to mark pin and seed shape locations in responses. The default value is 0.25.
FILTER DISTANCE	The distance from the point of a location-based query beyond which a port, device, or net shape is ignored. The default value is 0, which means the port or device coordinate must exactly match the location point, and the edge of a net shape must pass through the location point. In normal use, this value should be set to a value greater than 0.
FILTER LAYERS	A list of layer names or numbers. In NET LOCATION or PORT LOCATION queries, ports or net shapes not on one of these layers are ignored. This filter has no effect on any other queries. The default value is ALL, which represents all possible layers.
FILTER DEVICES	A list of device type numbers. In DEVICE LOCATION queries, devices whose types are not present on the list are ignored. The default value is ALL, which represents all possible device types. See “ Devices ” on page 24 for more details.
FILTER WINDOW	A rectilinear area beyond which location-based query responses are ignored. Setting filter windows can decrease response time of a command.
FILTER CULL	The (x, y) dimensions specifying the minimum extent dimensions required for a polygon to be reported in results sets. If a polygon is not either wider than x, or taller than y, it is not reported.

Commands are provided for the application to create new client contexts, specify which client context is currently active, and delete client contexts. The server treats all queries as having come from the currently-active client context, and there is always exactly one active client.

The server assigns client ID numbers (non-negative integers) to the clients. Client ID 0 represents the application itself as a client context. It initially exists, is active, can become inactive, but can never be deleted. A client context can only be inactivated by activating another client context. Only an inactive client context can be deleted.

Because the Query Server can support multiple clients (only one is active at any time), it creates a separate response file for each client. When a response file is established for a client ID, it is used for all responses until another response file is established or the client context's response mode is set to direct. It is the client application's responsibility to delete these files when they are no longer needed.

Chapter 5

Standard Query Commands

The standard mode Query Server responds to commands that query SVDB databases and the server environment. For commands that have counterparts in the Query Server Tcl shell, the Tcl command equivalents are shown in the header of each reference page. The Tcl shell is the preferred interface.

Command Format	261
Communication and Control Commands	263
Query Setup Commands	276
Cell Query Commands	301
Browse Deviceless or Pseudo Cells Commands	309
Device Query Commands	315
Net Query Commands.....	330
Placement Query Commands.....	363
Port Query Commands	380
Rule File Query Commands	386
Hcell Analysis Commands.....	410

Command Format

Each standard query command is a single string of text terminated by a return character. Commands may be issued directly in the server shell, or they may appear in a script. The words chosen to represent a command are a compromise between brevity and ease of human recognition. Later sections cover each command in detail.

Blank lines and lines beginning with the pound character (#) are ignored and may be used in test scripts or for documentation. All other lines are treated as command lines. Here are some examples:

```
NET SHAPES X12/X45/CLOCK1
# app asks for shapes of a net relative to the current query cell
DEVICE INFO X95/X4/X312/C4
# app asks for information about a device relative to the current
# query cell
ACTIVATE 12
# app asks server to deactivate the current context and activate
# client id 12
```

Unless specified otherwise, commands and their arguments pertain to the current query cell, and pathnames are relative to the current query cell. Nets in a command can be referred to by name or by number. Nets returned in responses are always returned by name if possible, otherwise by number.

All commands return a brief acknowledgment message. More verbose responses are shown for commands that issue response messages. If a command does not issue a response, none is indicated in the command descriptions.

Although they are shown in uppercase in this document, the server is not sensitive to the text case of the command words.

Commands that access a layout file generally match the case sensitivity constraints for GDSII and OASIS®¹ layout formats. Cell names are first searched using the exact case specified in the command. If no results are generated, a second search is performed without regard to case. Layout placement names and device names are always considered case-insensitive. You can control how case sensitivity is considered for layout net names using [Layout Preserve Case](#).

Case-sensitivity for XDB (cross-reference database) commands is affected by the [Layout Case](#) and [Source Case](#) statements in the rule 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.

Communication and Control Commands

This set of commands is used for communication and control between the Query Server and the client.

Note

 In most cases, you do not need to use the ACTIVATE, CONNECT, or DISCONNECT commands.

Table 5-1. Communication and Control Commands

Command	Description
ACTIVATE	Activates a server client.
CONNECT	Connects a new client to the server.
DISCONNECT	Disconnects a client from the server.
ECHO	Controls repetition of inputs in the output.
HELP COMMANDS	Lists the Query Server commands.
PASSWORD	Sets a password for an SVDB database.
QUIT	Exits the Query Server.
RESPONSE DIRECT	Sets the Query Server response channel to STDOUT.
RESPONSE FILE	Specifies an output file for server responses.
STOP ON	Specifies error and NOK message handling behavior.
TERMINATE	Exits the Query Server.

ACTIVATE

Query Server command.

Activates a server client.

Usage

ACTIVATE *client_id*

Arguments

- *client_id*

A required positive integer.

Description

For most users, this command is not needed.

Deactivates a current client (if one exists) and activates the client (in its current context) indicated by the *client_id*.

In case of an error, the current client remains active.

Acknowledgments

ERROR(106), ERROR(107)

Related Topics

[CONNECT](#)

[DISCONNECT](#)

[Communication and Control Commands](#)

CONNECT

Query Server command.

Connects a new client to the server.

Usage

CONNECT

Arguments

None.

Description

For most users, this command is not needed.

Deactivates the current client context, creates an unused client ID number, initializes the client table entries for that context to their default values, makes the new client context the active context, and returns the new client ID. Client IDs are non-negative integers.

There is no reasonable limit to the number of client contexts that may be defined at any given time, but only one may be the current active client context. When the server is initiated, it creates client ID 0 and activates it. This client context can never be deleted. It should be reserved for use by the application acting on its own behalf when multiple client contexts are being maintained. For example, some application implementations may want to cache information such as the device table so they can use it directly without requesting it from the server for each operation.

Acknowledgments

OK: *new_client_id*

Related Topics

[ACTIVATE](#)

[DISCONNECT](#)

[Communication and Control Commands](#)

DISCONNECT

Query Server command.

Disconnects a client from the server.

Usage

DISCONNECT *client_id*

Arguments

- *client_id*

A required positive integer.

Description

For most users, this command is not needed.

Deletes the stored current context of an inactive client associated with the *client_id*, which makes the *client_id* undefined for the rest of the Query Server session. The *client_id* is a positive integer.

Acknowledgments

OK., ERROR(106), ERROR(107), ERROR(108), ERROR(109)

Related Topics

[ACTIVATE](#)

[CONNECT](#)

[Communication and Control Commands](#)

ECHO

Query Server command.

Controls repetition of inputs in the output.

Usage

ECHO

Arguments

None.

Description

Toggles the STDIN echo state. By default, echoing of input is off. If the echo state is on, all input is echoed to output.

When testing the server with a script, it is convenient to see the echo from STDIN so that the output contains a complete transcript of the session, including comments and blank lines in the input.

The acknowledgment shows the new echo state. The echo state applies to all clients.

Acknowledgments

OK., ECHO ON or ECHO OFF

Related Topics

[Communication and Control Commands](#)

HELP COMMANDS

Query Server command. Corresponding Tcl shell command: [qs::help](#).

Lists the Query Server commands.

Usage

HELP COMMANDS

Arguments

None.

Description

Returns a list of the valid Query Server commands and arguments, including the Calibre Connectivity Interface. Each command is listed together with its arguments on a separate line. Alternate forms are listed on separate lines. The list is intended to serve as a reminder for people already familiar with the command set. The returned text does not explain the commands.

Response

```
Help_Commands <precision>          // "Help_Commands" and precision
Commands:                           // "Commands:"
0 0 <n> <date>                   // n lines of text follow (no shapes)
protocol version                   // first command description
help commands                      // next command description
...
device info <layout_device_path> // last command description
```

Acknowledgments

OK., ERROR(1), ERROR(101)

Related Topics

[Communication and Control Commands](#)

[RESPONSE FILE](#)

PASSWORD

Query Server command. Corresponding Tcl shell command: [qs::password](#).

Sets a password for the current SVDB.

Usage

PASSWORD [*pwttext*]

Arguments

- *pwttext*

An optional, case-sensitive string to serve as a password. If the string is non-empty, it must be composed of any combination of letters, numbers, and symbols. Using no argument removes the password of a database. This string must be specified if the password is set in a script.

Description

Sets or resets a password restricting access to the current SVDB database.

If the command is invoked without *pwttext* in an interactive session, the command prompts for a password twice. The string provided must match for both prompts in order for the password to be set successfully.

Acknowledgments

OK., NOK(54)

Related Topics

[Communication and Control Commands](#)

QUIT

Query Server command. Corresponding Tcl shell command: [qs::quit](#).

Exits the Query Server. Performs identically to the TERMINATE command.

Usage

QUIT

Arguments

None.

Acknowledgments

OK: Terminating.

Related Topics

[Communication and Control Commands](#)

[TERMINATE](#)

RESPONSE DIRECT

Query Server command. Corresponding Tcl shell command: puts.

Sets the Query Server response message channel to STDOUT, which is the default. If a response file exists at the time this command is processed, it is retained.

Usage

RESPONSE DIRECT

Arguments

None.

Acknowledgments

OK.

Examples

This shows a typical command sequence in a script. The response from DEVICE TABLE is sent to the *device_table* file. Then the response output is restored to STDOUT.

```
RESPONSE FILE device_table
DEVICE TABLE
RESPONSE DIRECT
```

Related Topics

[RESPONSE FILE](#)

[Responses and Response Files](#)

[Communication and Control Commands](#)

RESPONSE FILE

Query Server command. Corresponding Tcl shell command: puts.

Specifies an output file for server responses.

Usage

RESPONSE [MULTIPLE] FILE *filename*

Arguments

- *filename*

A required pathname of an output file. Directories in a pathname are created if they do not exist.

- MULTIPLE

An optional keyword that specifies multiple sequential responses are all appended to the response file. Normal response header and trailer lines are not issued for subsequent commands.

Description

Establishes a file as the response output location for the current client context. For commands that issue responses (not all do), this file becomes the output destination. The default output location is STDOUT. This command issues no response data. It simply acknowledges OK for successful response file creation. The response file may already exist, in which case any response is written into it at the front of the file and overwriting any existing content.

The MULTIPLE setting is useful for viewing responses to several commands in succession. When this keyword is used, the response file's top-level cell name field is "Multiple_Response_File" rather than the usual layout cell name.

If this command fails with an error, the response mode is set as if the command had been **RESPONSE DIRECT**. This is because any current response file may no longer be appropriate. That is, the client may have deleted or renamed that file before issuing this command. The client must watch for this error since it changes the acknowledgment channel.

Once the *filename* has been validated, the server attempts to use it for all following responses until another RESPONSE DIRECT or RESPONSE FILE command is issued. If the server is no longer able to write to this file path, each command that attempts to generate a response fails with this message:

ERROR(101): File <response_file_path> could not be opened for writing.

Once the RESPONSE FILE command has accepted the path, the only way to change it is with a RESPONSE DIRECT or RESPONSE FILE command.

Acknowledgments

OK., ERROR(101), ERROR(102)

Examples

This shows a typical command sequence in a script. The response from DEVICE TABLE is sent to the *device_table* file. Then the response output is restored to STDOUT.

```
RESPONSE FILE device_table
DEVICE TABLE
RESPONSE DIRECT
```

Related Topics

[Responses and Response Files](#)

[Communication and Control Commands](#)

STOP ON

Query Server command.

Specifies error and NOK message handling behavior.

Usage

STOP ON {ERROR | NOK} {NO | YES}

Arguments

- **NO**
A keyword that specifies that the server does not stop on the associated acknowledgment message type. This is the default.
- **YES**
A keyword that specifies the server stops on the associated acknowledgment message type.
- **ERROR**
A keyword that specifies the command acts on error messages. This is the default. May not be explicitly specified with **NOK**.
- **NOK**
A keyword that specifies the command acts on NOK messages. This is the default. May not be explicitly specified with **ERROR**.

Description

Controls how the Query Server handles ERROR and NOK acknowledgments. By default, ERROR and NOK messages do not cause the server to terminate.

If **NO** or **YES** is explicitly specified, then one of **ERROR** or **NOK** must also be specified. This command may be specified more than once to cover complementary conditions.

Acknowledgments

OK.

Related Topics

[Communication and Control Commands](#)

TERMINATE

Query Server command. Corresponding Tcl shell command: [qs::quit](#).

Exits the Query Server.

Usage

TERMINATE

Arguments

None.

Description

Causes the Query Server to exit. This is the same as the [QUIT](#) command.

An end of file (EOF) also causes the Query Server to terminate. This typically occurs when Query Server commands are input using redirection from STDIN to a file.

Acknowledgments

OK, Terminating.

Related Topics

[Communication and Control Commands](#)

Interrupts

Certain commands (generally those that take a while to complete) terminate in response to the SIGINT signal. When the Query Server is run using a shell command line, this signal can be sent by the ^C keystroke. This signal can be sent to the process using the kill system call on Unix-type systems.

The Query Server can also be suspended with the SIGSTOP (^Z in shells with job control) and resumed with SIGCONT. See shell documentation on job control for more information on SIGSTOP and SIGCONT.

Related Topics

[Communication and Control Commands](#)

Query Setup Commands

This set of commands defines various parameters that affect the results of queries or request information about the run status.

COMPARISON RESULT	277
CONTEXT	278
FILTER CULL	280
FILTER DEVICES	281
FILTER DEVICELAYERS	283
FILTER DISTANCE	284
FILTER LAYERS	285
FILTER {IN OUT} LAYERS	286
FILTER WINDOW	288
MARKER SIZE	289
MAXIMUM VERTEX COUNT	290
STATUS	291
Results Transformation Commands	294
MAGNIFY RESULTS	296
REFLECTX RESULTS	298
ROTATE RESULTS	299
TRANSLATE RESULTS.....	300

COMPARISON RESULT

Query Server command. Corresponding Tcl shell command:
[qs::get_run_summary_data -lvs_compare_status](#).

Returns the LVS comparison result.

Usage

COMPARISON RESULT

Arguments

None.

Description

Returns the result of the LVS comparison (CORRECT, INCORRECT, or NOT COMPARED). An INCORRECT result means some things could not be matched between layout and source. Such objects result in certain information not being retrievable by the Query Server.

Acknowledgments

OK: *result*, NOK(53), ERROR(150)

Related Topics

[Query Setup Commands](#)

CONTEXT

Query Server command. Nearest Tcl shell analogue: [dfm::get_cells](#).

Sets the cell context for queries.

Usage

CONTEXT *view_cell* [*query_instance_path*]

CONTEXT *view_cell* *layout_pin_count* *source_cell* *source_pin_count*
[*query_instance_path*]

Arguments

- ***view_cell***

A required argument that specifies the name of the layout viewing cell. The *view_cell* names can be original cell names or FAUX BIN cells as identified in the LVS run transcript.

- ***layout_pin_count***

An integer that specifies a layout cell pin count. This argument is used when there is a many-to-one correspondence between a *source_cell* and layout cells.

- ***source_cell***

The name of a source cell that corresponds to the *view_cell*. This argument is used when there is a many-to-one correspondence between a *source_cell* and layout cells.

- ***source_pin_count***

An integer that specifies a source cell pin count. This argument is used when there is a many-to-one correspondence between a *source_cell* and layout cells.

- ***query_instance_path***

An optional pathname of a query instance. The *query_instance_path* must be of the form X1/X2 /.../X(k-1)/Xk, where integers are used in the instance names. The path is relative to the *view_cell*.

Description

Establishes the query cell and optional path to an instance being queried. See “[Command Context in the Standard Query Server](#)” on page 255 for related definitions. This command deletes any existing filter windows (see [FILTER WINDOW](#)). It also changes the client table cell context entries for the current client (see [ACTIVATE](#)).

If OK is returned, the following changes are made to the client table for the current client:

- The viewing cell entry is set to *view_cell*. All results are returned in the context of this cell’s coordinate space.
- If *query_instance_path* is absent, the query cell entry is set to *view_cell* and the query instance entry is set to NULL; otherwise, the query instance entry is set to

query_instance_path, and the query cell entry is set to the name of the corresponding cell of that instance.

If an error is returned, the current context remains unchanged.

If your design contains many source cells that correspond to one layout cell, you can specify the layout (*view_cell*) and source (*source_cell*) names, along with the respective pin count parameters to establish a cross-reference context for an hcell.

The acknowledgment returns the name of the query cell so that the client may confirm the *query_instance_path* leads to the desired type of cell, or, in the case where a user issued the query path through an interface, the interface can determine the newly established query cell.

Acknowledgments

OK: Query cell: *query_cell*

Note

 If multiple layout cells are chosen with the name *view_cell* and the same *pin_count* parameters, these additional errors may appear: ERROR(103), ERROR(104), ERROR(120), ERROR(125).

Related Topics

[Query Setup Commands](#)

FILTER CULL

Query Server command.

Filters out result shapes that do not have a minimum width and height. Marker squares produced by Query Server commands are never filtered.

Usage

FILTER CULL *width height*

Arguments

- ***width***

A required floating-point width measured parallel the x-axis in user units. The default is 0.

- ***height***

A required floating-point height measured parallel to the y-axis in user units. The default is 0.

Acknowledgments

OK., ERROR(117)

Related Topics

[Query Setup Commands](#)

FILTER DEVICES

Query Server command. Corresponding Tcl shell command: [dfm::create_filter](#) -device_ids or -device_names.

Specifies which devices get queried.

Usage

Syntax 1

FILTER DEVICES {ALL | *index_list*}

Syntax 2

FILTER DEVICENAMES {*element_name* [‘(‘*model_name* ‘)’]} ...

Arguments

- **ALL**

A keyword that specifies all devices are queried. This is the default behavior regardless of the syntax line. Specified only in Syntax 1.

- ***index_list***

A whitespace-delimited list of device index numbers. Valid numbers range from 0 to one less than the number of [Device](#) statements in the rule file. The index can be determined from the device table. See “[Devices](#)” on page 24 for more details. Specified only in Syntax 1.

- ***element_name***

A required SPICE device element name in Syntax 2. More than one name may be specified, with optional *model_name* parameters.

- **(*model_name*)**

An optional SPICE model (subtype) name specified with an *element_name* in Syntax 2.

Description

Controls which devices are queried. Devices that are in the current filter set can be queried. When a command is issued, the new set of devices is effective with the next response from the server.

There are two syntactical forms, each with its own set of parameters. FILTER DEVICES ALL is set by default.

If an invalid command or list is given, the former list is retained.

Devices that are not members of an existing filter are omitted from a netlist written by [LAYOUT NETLIST WRITE](#). Any nets associated solely with pins of such removed devices are also removed from the cell in which a removed device would otherwise reside. Any cell ports (external pins) associated solely with such removed internal nets are also removed from the subcircuit definition, and the “EP=*n*” annotations in the netlist reflect reduced pin counts.

Accordingly, placements of such cells also have their corresponding placement pins removed. Flat device count ($FDC=n$) annotations reflect a reduced device count. Instance references to cells that are empty due to omitted devices likewise are omitted from the output netlist. Omitting devices and associated objects in this way can result in improved performance of downstream tools due to reduced netlist content.

Acknowledgments

OK., ERROR(115), ERROR(126)

Related Topics

[FILTER DEVICE LAYERS](#)

[Query Setup Commands](#)

FILTER DEVICELAYERS

Query Server command. Corresponding Tcl shell command: [dfm::create_filter -layers](#).

Specifies which device seed layers get queried.

Usage

FILTER DEVICELAYERS *layer* [*layer* ...]

Arguments

- *layer*

A required seed layer name specified in a [Device](#) statement in the rule file. More than one layer may be specified.

Description

Controls which device seed layers are queried. All devices that match one of the seed layer names are included in the filtered list. By default, all seed layers are in the filter list.

Acknowledgments

OK., ERROR(158)

Related Topics

[FILTER DEVICES](#)

[FILTER LAYERS](#)

[Query Setup Commands](#)

FILTER DISTANCE

Query Server command.

Specifies a distance from a query location in which to search for objects.

Usage

FILTER DISTANCE *distance*

Arguments

- *distance*

A required non-negative floating-point search distance in user units. The default is 0.

Description

Sets the distance to search for objects to the ***distance*** for the current client. The value is not checked for a reasonable maximum size, and unnecessarily large values may significantly delay the server response. A new distance is effective beginning with the next response. If an invalid distance is given, the former value is retained.

When the default is used, the hierarchy is searched only for placements that overlap a query point directly (not placements that are nearby).

Acknowledgments

OK., ERROR(113)

Related Topics

[Query Setup Commands](#)

FILTER LAYERS

Query Server command. Corresponding Tcl shell command: [dfm::create_filter -layers](#).

Specifies which PHDB layers get queried.

Usage

FILTER LAYERS {ALL | *layer* [*layer* ...]}

Arguments

- **ALL**

A keyword that specifies all layers in the PHDB are queried. This is the default.

- ***layer***

A name or number of a PHDB layer. Layer names are taken from the rule file. More than one layer may be specified.

Description

Controls which layers are queried. A new filter list is effective beginning with the next response.

Only layers in the persistent hierarchical database (PHDB) are in the filter list. If an invalid list is given, the former list is retained.

Layers in a list that are not relevant to some subsequent command are ignored. For example, the [NET LOCATION](#) command ignores any non-connectivity layers even if they are in the filter list.

The [FILTER {IN | OUT} LAYERS](#) command modifies an existing layer list.

FILTER LAYERS acts independently from [AGF FILTER BOX LAYERS](#) and [AGF FILTER {IN | OUT} BOX LAYERS](#) in the Calibre Connectivity Interface.

Acknowledgments

OK., ERROR(111), ERROR(114)

Related Topics

[FILTER DEVICELAYERS](#)

[Query Setup Commands](#)

FILTER {IN | OUT} LAYERS

Query Server command.

Adds or removes layers in the current layer query list.

Usage

FILTER IN LAYERS {ALL | *layer* [*layer* ...]}

FILTER OUT LAYERS {ALL | *layer* [*layer* ...]}

Arguments

- **IN**

A keyword that specifies to include layers in queries. By default, all layers in the FILTER LAYERS list are included.

- **OUT**

A keyword that specifies to exclude layers from queries.

- **ALL**

A keyword that specifies all devices in the FILTER LAYERS list are acted upon. This is the default.

- ***layer***

A name or number of a PHDB layer. Layer names are taken from the rule file. More than one layer may be specified.

Description

Adds or removes layers from an existing layer list created with **FILTER LAYERS**. Either **IN** or **OUT** must be specified, along with the argument that specifies the layers to act upon.

If an invalid layer list is given, the former list is retained.

Layers in a list that are not relevant to some subsequent command are ignored. For example, the **NET LOCATION** command ignores any non-connectivity layers even if they are in the filter list.

Acknowledgments

OK., ERROR(111), ERROR(114)

Examples

This example shows successive executions of filtering commands and the responses from the server.

```
FILTER LAYERS cnt metal1 metal2
    OK.
STATUS FILTER LAYERS
    OK: Filter layers: cnt metal1 metal2
FILTER IN LAYERS via
    OK.
STATUS FILTER LAYERS
    OK: Filter layers: cnt metal1 metal2 via
FILTER OUT LAYERS metal1
    OK.
STATUS FILTER LAYERS
    OK: Filter layers: cnt metal2 via
FILTER OUT LAYERS ALL
    OK.
STATUS FILTER LAYERS
    OK: Filter layers: (none)
```

Related Topics

[FILTER DEVICELAYERS](#)

[Query Setup Commands](#)

FILTER WINDOW

Query Server command.

Specifies regions that are queried.

Usage

FILTER WINDOW {NONE | INCLUDE $x1\ y1\ x2\ y2$ | EXCLUDE $x1\ y1\ x2\ y2$ }

Arguments

- **NONE**

A keyword that specifies no area filtering is performed.

- **INCLUDE**

A keyword that specifies a region that is included in queries.

- **EXCLUDE**

A keyword that specifies a region that is excluded from queries.

- **$x1\ y1\ x2\ y2$**

When **INCLUDE** or **EXCLUDE** is specified, four required floating-point numbers in user units defining the lower-left and upper-right corners of a rectangular region. The coordinates are referenced to the viewing cell.

Description

Controls which regions of the layout can be queried.

The **NONE** keyword clears all current filter windows and returns to the default state (results from the entire set of coordinates are presented). The **CONTEXT** command clears all windows.

Successive calls to this command with the **INCLUDE** and **EXCLUDE** keywords build up complex viewing areas consisting of multiple rectangular areas that are either included or excluded.

Acknowledgments

OK., ERROR(117)

Related Topics

[Query Setup Commands](#)

[Results Transformation Commands](#)

MARKER SIZE

Query Server command.

Specifies a size of marker square to be used by commands that set markers. A new size is effective beginning with the next server response.

Usage

MARKER SIZE *size*

Arguments

- *size*

A required, positive, floating-point number in user units, which is the side length of a square. The default is 0.25.

Acknowledgments

OK., ERROR(110)

Related Topics

[Query Setup Commands](#)

MAXIMUM VERTEX COUNT

Query Server command. Corresponding Tcl shell command:

`dfm::create_filter -maximum_vertex_count`.

Sets the maximum vertex count for results in all clients. Output polygons having more than the specified number of vertices are partitioned into polygons having fewer vertices.

Usage

MAXIMUM VERTEX COUNT {*value* | RESET}

Arguments

- ***value***
An integer greater than or equal to 4. The default is 4096.
- **RESET**
A keyword that specifies to set the maximum vertex count to 4096.

Acknowledgments

OK., ERROR(119)

Related Topics

[Query Setup Commands](#)

STATUS

Query Server command. Corresponding Tcl shell command: [qs::status](#).

Provides the state of server settings.

Usage

```
STATUS [CLIENT | FILTER CULL | FILTER DEVICES | FILTER DISTANCE |
        FILTER LAYERS | FILTER WINDOWS | MAGNIFY RESULTS | MARKER SIZE |
        MAXIMUM VERTEX COUNT | PHDB | XDB | QUERY CELL | QUERY INSTANCE |
        REFLECTX RESULTS | RESPONSE MODE | ROTATE RESULTS | TRANSLATE
        RESULTS | VIEW CELL]
```

Arguments

- **CLIENT**
An optional keyword that returns the current client number set with the [ACTIVATE](#) or [CONNECT](#) commands.
- **FILTER CULL**
An optional keyword that returns the width and length of filtered polygons set with the [FILTER CULL](#) command.
- **FILTER DEVICES**
An optional keyword that returns the [FILTER DEVICES](#) setting.
- **FILTER DISTANCE**
An optional keyword that returns the [FILTER DISTANCE](#) setting.
- **FILTER LAYERS**
An optional keyword that returns the [FILTER LAYERS](#) setting.
- **FILTER WINDOWS**
An optional keyword that returns the [FILTER WINDOW](#) settings.
- **MAGNIFY RESULTS**
An optional keyword that returns the [MAGNIFY RESULTS](#) setting.
- **MARKER SIZE**
An optional keyword that returns the [MARKER SIZE](#) setting.
- **MAXIMUM VERTEX COUNT**
An optional keyword that returns the [MAXIMUM VERTEX COUNT](#) setting.
- **PHDB**
An optional keyword that returns statistics of the currently-loaded [PHDB](#), if any.
- **XDB**
An optional keyword that returns statistics of the currently-loaded [XDB](#), if any.

- **QUERY CELL**
An optional keyword that returns the query cell name associated with the query instance. This is controlled by the [CONTEXT](#) command.
- **QUERY INSTANCE**
An optional keyword that returns the query instance pathname set with the [CONTEXT](#) command.
- **REFLECTX RESULTS**
An optional keyword that returns the [REFLECTX RESULTS](#) setting.
- **RESPONSE MODE**
An optional keyword that returns the response mode set with the [RESPONSE DIRECT](#) and [RESPONSE FILE](#) commands.
- **ROTATE RESULTS**
An optional keyword that returns the [ROTATE RESULTS](#) setting.
- **TRANSLATE RESULTS**
An optional keyword that returns the [TRANSLATE RESULTS](#) setting.
- **VIEW CELL**
An optional keyword that returns the current viewing cell name set with the [CONTEXT](#) command.

Description

Returns the status of various elements in the system given by the optional keywords.

If STATUS alone is issued, values for all the arguments in the client's current context are returned, and the acknowledgment is "OK."

The order in which the keyword: value pairs appear in the response is not guaranteed. Program clients that parse this response should do so on the basis of the value of *keyword*, not on an assumed position of a given item in the list.

If the view and query cells are the same, the value of STATUS QUERY INSTANCE is (null).

If the [FILTER LAYERS](#) list contains all layers, the return value is (all). If the [FILTER DEVICES](#) list contains all device types, the return value is (all).

[MAXIMUM VERTEX COUNT](#) value is global, not context-specific.

The STATUS, STATUS PHDB, and STATUS XDB commands respond with the Calibre version number used to create the respective databases. This can be used to determine if an older database is incompatible with a newer Calibre version. The XDB and PHDB keywords also report the database type (flat or hierarchical), as well as if the database contains no XDB or PHDB information, respectively.

Output can be sent to a [RESPONSE FILE](#).

Response

If a keyword is given, there is no response, just an acknowledgment with a value. Otherwise the response is the following:

```
Status <precision>      // ''Status'' and precision
Entries:                 // Entries:
0 0 <n> <date>        // n lines of text follow (no shapes)
<keyword_1>:<value_1> // first keyword name and value
...
<keyword_n>: <value_n> // intermediate names and values
<keyword_n>: <value_n> // last keyword name and value
```

Acknowledgments

OK.

Related Topics

[Query Setup Commands](#)

Results Transformation Commands

This set of commands transforms output geometric coordinates from other commands.

Table 5-2. Results Transformation Commands

Command	Description
MAGNIFY RESULTS	Multiplies output coordinates by a positive factor.
REFLECTX RESULTS	Reflects output y coordinates across the x axis.
ROTATE RESULTS	Rotates output coordinates about the database origin by an integral multiple of 90 degrees.
TRANSLATE RESULTS	Displaces output coordinates by specified values.

These commands can be issued in any order to the Query Server and acknowledgments are given as for other commands. However, they are applied to coordinates in this order:

REFLECTX RESULTS

ROTATE RESULTS

MAGNIFY RESULTS

TRANSLATE RESULTS

The current settings of these commands can be queried with the [STATUS](#) command.

The following commands observe transformations.

Table 5-3. Commands Observing Results Transformations

CELL EXTENTS WRITE	LAYOUT NETLIST WRITE
DEVICE BAD	LAYOUT SEPARATED PROPERTIES WRITE
DEVICE INFO	NET BROWSE SHAPES
DEVICE LOCATION	NET EXTERNAL SHAPES
DEVICE PINS	NET LOCATION
DEVICE TABLE ¹	NET PORTS
EXTENT	NET SHAPES
FILTER WINDOW	PLACEMENT INFO
AGF MAP ¹	PLACEMENT LOCATION (without <i>cell_name</i> option)

Table 5-3. Commands Observing Results Transformations (cont.)

AGF MAGNIFY USER UNITS PLACEMENT TRANSFORM
(controls whether magnification occurs
in CCI)

AGF WRITE PORT INFO
 PORT TABLE WRITE

1. Observes MAGNIFY RESULTS factor only, which is used for the precision value of the response.

The Tcl shell analogues to these commands are [dfm::create_filter](#) options.

MAGNIFY RESULTS

Query Server setup command. Corresponding Tcl shell command: [dfm::create_filter -magnify](#).

Multiplies output coordinates by a positive factor. The MAGNIFY RESULTS transformation is applied after any REFLECTX RESULTS or ROTATE RESULTS transformations (in that order), regardless of the order in which they are issued to the server.

Usage

MAGNIFY RESULTS {*factor* | AUTO}

Arguments

- ***factor***

A positive floating-point number. The default is 1.0. If 1.0 is not specified, the value must be the ratio of the [Layout Precision](#) to the [Precision](#) in the rule file, where the Precision is greater than the Layout Precision value, or a warning is given and the currently active value is used.

- **AUTO**

A keyword that specifies to use the ratio of the [Layout Precision](#) to [Precision](#) settings in the rule file, where the Precision is greater than the Layout Precision value.

Description

Scales the output of commands returning geometric coordinates by multiplying all coordinate values by a magnification factor. This command allows circuit extraction to be performed using a Calibre database precision that is higher than the input layout database precision while allowing retrieval of information using permitted Query Server commands at the precision originally present in the layout input database.

Specifying **AUTO** allows automatic magnification to be applied according to rule file settings. Specifying ***factor*** enforces a given magnification value.

When this command is used, [Layout Magnify AUTO](#), [Layout Precision](#), and [Precision](#) must be explicitly specified in the rule file. The Layout Precision must correspond to the physical precision of the layout database (for example, if the physical precision is 0.001, then Layout Precision must be 1000). The Precision value must be greater than the Layout Precision value. If these requirements are not met, errors are issued depending on the discrepancy.

If MAGNIFY RESULTS changes the magnification factor from 1.0, then the rule file Precision is multiplied by that factor and returned by the outputs of [RULES PRECISION](#) and [LVS SETTINGS REPORT WRITE](#).

If other commands are used than those listed under “[Commands Observing Results Transformations](#)” while a non-default ***factor*** is in effect, then [ERROR\(171\)](#) is issued. Specifying a ***factor*** of 1 restores the default.

Transformation command settings are reported by the [STATUS](#) command.

Application of magnification settings in the Calibre Connectivity Interface is controlled by [AGF MAGNIFY USER UNITS](#).

Acknowledgments

OK., ERROR(170), ERROR(171), ERROR(172), ERROR(173), ERROR(174)

Examples

Example 1

Assume the rule file has the following when the SVDB is generated:

```
LAYOUT PRECISION 1000
PRECISION 10000
LAYOUT MAGNIFY AUTO
```

Then this command in the Query Server is allowed:

```
MAGNIFY RESULTS 0.1
```

Any other magnification value causes a warning and the existing magnification factor (1.0 by default) is used. Commands that observe this setting may now be used.

Example 2

Assume these rule file settings:

```
LAYOUT PRECISION 1000
PRECISION 10000
LAYOUT MAGNIFY AUTO
```

The RULES PRECISION command returns the following:

```
RULES PRECISION OK: 10000
```

Changing the magnification value changes the precision:

```
MAGNIFY RESULTS AUTO
NOTE: <magnification_factor> automatically set to 0.1
RULES PRECISION
OK: 1000
```

A similar effect is seen in the PRECISION entry of the report written by LVS SETTINGS REPORT WRITE.

Related Topics

[Results Transformation Commands](#)

REFLECTX RESULTS

Query Server setup command. Corresponding Tcl shell command: `dfm::create_filter -reflect_x`.
Reflects output y coordinates across the x axis. If YES is specified, the reflection is performed before any other transformations.

Usage

REFLECTX RESULTS {NO | YES}

Arguments

- **NO**
A keyword that specifies result coordinates are not reflected. This is the default.
- **YES**
A keyword that specifies y coordinate values are transformed to their additive inverses.

Examples

This shows how to get reflection across the y axis.

```
REFLECTX RESULTS YES
ROTATE RESULTS 180
```

Related Topics

[Results Transformation Commands](#)

ROTATE RESULTS

Query Server command. Corresponding Tcl shell command: [dfm::create_filter -rotate](#).

Rotates output coordinates about the database origin by an integral multiple of 90 degrees. The positive direction is counter-clockwise. The ROTATE RESULTS transformation is applied after any REFLECTX RESULTS transformations, regardless of the order in which they are issued to the server.

Usage

ROTATE RESULTS *angle*

Arguments

- *angle*

A required integer from the set {-270, -180, -90, 0, 90, 180, 270} in degrees. The default is 0.

Related Topics

[Results Transformation Commands](#)

TRANSLATE RESULTS

Query Server setup command. Corresponding Tcl shell command: `dfm::create_filter -translate`. Displaces output coordinates by specified values. The TRANSLATE RESULTS transformation is applied after any REFLECTX RESULTS, ROTATE RESULTS, or MAGNIFY RESULTS transformations (in that order), regardless of the order in which they are issued to the server.

Usage

TRANSLATE RESULTS {*x_offset* | *y_offset*}

Arguments

- ***x_offset***
An integer in database units that is added to all x coordinates. The default is 0.
- ***y_offset***
An integer in database units that is added to all y coordinates. The default is 0.

Related Topics

[Results Transformation Commands](#)

Cell Query Commands

This set of commands returns cell information.

Table 5-4. Cell Query Commands

Command	Description
CELLS CORRESPONDING	Returns a list of matched cells.
CELL CORRESPONDING LAYOUT	Returns the name of the corresponding layout cell for the specified source cell.
CELL CORRESPONDING SOURCE	Returns the name of the corresponding source cell for the specified layout cell.
CELL TOP	Returns the name of the top-level cell.
CELLS LAYOUT	Returns layout cell names from an LVS comparison
CELLS SOURCE	Returns source cell names from an LVS comparison.
EXTENT	Returns the extent of the layout design in top-level space.

For commands that issue responses, output can be redirected to a [RESPONSE FILE](#).

CELLS CORRESPONDING

Query Server command. Corresponding Tcl shell commands:

[dfm::get_xref_cells](#) -corresponding and [dfm::get_xref_cell_data](#).

Returns a list of matched cells.

Usage

CELLS CORRESPONDING

Arguments

None.

Description

Returns a list of corresponding cell pairs as determined by Calibre nmLVS-H. Each pair consists of a layout cell name and the corresponding source cell name.

Cells are listed in top-to-bottom order. That is, if cell A directly or indirectly contains an instance of cell B, then cell A appears earlier in the list than cell B. Thus the first cell in the list is the top cell of the design. This list contains only the cells that correspond between layout and source. The complete list of layout cells or source cells can be obtained using the [CELLS LAYOUT](#) or [CELLS SOURCE](#) commands.

On account of the -automatch option for hierarchical LVS, this list may be more comprehensive than an hcell list because -automatch also includes all cells matched by name.

See “[Hcell Analysis Commands](#)” on page 410 for commands that optimize an hcell list.

Response

```
Corresponding_Cells <precision> // "Corresponding_Cells" and precision
Correspondences:           // Correspondences:
 0 0 <n> <date>          // n lines of text follow (no shapes)
  <layout_cell_name_1> <source_cell_name_1>
  ...
  <layout_cell_name_n> <source_cell_name_n>
```

Acknowledgments

OK., NOK(21), ERROR(101), ERROR(102)

Related Topics

[CELL CORRESPONDING LAYOUT](#)

[CELL CORRESPONDING SOURCE](#)

[RESPONSE FILE](#)

[Cell Query Commands](#)

CELL CORRESPONDING LAYOUT

Query Server command. Corresponding Tcl shell command: [dfm::get_layout_name -cell_name](#). Returns the name of the corresponding layout cell for the specified source cell. The correspondence is the same as what is returned in the CELLS CORRESPONDING response.

Usage

CELL CORRESPONDING LAYOUT *source_cell_name*

Arguments

- *source_cell_name*

A required name of a source netlist subcircuit.

Acknowledgments

OK: layout_cell_name, NOK(21), NOK(26)

Related Topics

[CELL CORRESPONDING SOURCE](#)

[Cell Query Commands](#)

CELL CORRESPONDING SOURCE

Query Server command. Corresponding Tcl shell command:

`dfm::get_source_name -cell_name`.

Returns the name of the corresponding source cell for the specified layout cell. The correspondence is the same as what is returned in the CELLS CORRESPONDING response.

Usage

CELL CORRESPONDING SOURCE *layout_cell_name*

Arguments

- *layout_cell_name*

A required name of a layout netlist subcircuit.

Acknowledgments

OK: source_cell_name, NOK(2), NOK(21)

Related Topics

[CELL CORRESPONDING LAYOUT](#)

[Cell Query Commands](#)

CELL TOP

Query Server command. Corresponding Tcl shell command: [dfm::get_top_cell](#).

Returns the name of the top-level cell.

Usage

CELL TOP

Arguments

None.

Acknowledgments

OK: top_cell_name, NOK(51)

Related Topics

[Cell Query Commands](#)

CELLS LAYOUT

Query Server command. Corresponding Tcl shell commands: `dfm::get_xref_cells -layout` and `dfm::get_xref_cell_data`.

Returns layout cell names from an LVS comparison.

Usage

CELLS LAYOUT

Arguments

None.

Description

Responds with a list of all layout cell names.

Cells are listed in top-to-bottom order according to hierarchy. That is, if cell A contains an instance of cell B, then cell A appears earlier in the list than cell B. Thus the first cell in the list is the top cell of the design. The list of cell names contains all cells in the layout, not just those that correspond to source cells. The list returned can be used by the client to set up a scan over all cells in the layout design.

Response

```
Cells_Layout <precision> // "Cells_Layout" and precision
Cells: // Cells:
0 0 <n> <date> // n lines of text follow (no shapes)
<cell_name_1> // top cell in layout design
...
<cell_name_n> // intermediate cells in layout design
// last cell in layout design
```

Acknowledgments

OK., ERROR(101), ERROR(102)

Related Topics

[CELLS SOURCE](#)

[CELL CORRESPONDING SOURCE](#)

[CELLS CORRESPONDING](#)

[RESPONSE FILE](#)

[Cell Query Commands](#)

CELLS SOURCE

Query Server command. Corresponding Tcl shell commands: [dfm::get_xref_cells -source](#) and [dfm::get_xref_cell_data](#).

Returns source cell names from an LVS comparison.

Usage

CELLS SOURCE

Arguments

None.

Description

Returns a list of all source cell names in a design.

Cells are listed in top-to-bottom order. That is, if cell A contains an instance of cell B, then cell A appears earlier in the list than cell B. Thus, the first cell in the list is the top cell of the design. The list of cell names contains all cells in the source, not just those that correspond to layout cells. The list returned can be used by the client to set up a scan over all cells in the source design.

Response

```
Cells_Source <precision> // "Cells_Source" and precision
Cells:                                // Cells:
0 0 <n> <date>                      // n lines of text follow (no shapes)
<cell_name_1>                          // top cell in source design
...
<cell_name_n>                          // intermediate cells in source design
                                         // last cell in source design
```

Acknowledgments

OK., NOK(21), ERROR(101), ERROR(102)

Related Topics

[CELLS LAYOUT](#)

[CELL CORRESPONDING LAYOUT](#)

[CELLS CORRESPONDING](#)

[RESPONSE FILE](#)

[Cell Query Commands](#)

EXTENT

Query Server command. Corresponding Tcl shell command: [dfm::get_db_extent](#).

Returns the extent of the layout design in top-level space.

Usage

EXTENT

Arguments

None.

Response

The following response occurs:

```
Extent <precision> // "Extent" and precision
<cell_name>           // query cell
1 1 0 <date_stamp> // indicates 1 polygon returned and date stamp
p 1 4                // polygon 1 having four vertices
<X1> <Y1>          // four coordinate sets
<X2> <Y1>
<X2> <Y2>
<X1> <Y2>
```

Acknowledgments

OK., ERROR(101), ERROR(102)

Examples

```
Extent 1000
mix
1 1 0 Mar 3 09:52:31 2003
p 1 4
-134000 -102000
80000 -102000
80000 32000
-134000 32000
END OF RESPONSE
0 0 0 Mar 3 09:52:31 2003
OK.
```

Related Topics

[Cell Query Commands](#)

[RESPONSE FILE](#)

[Results Transformation Commands](#)

Browse Deviceless or Pseudo Cells Commands

This set of commands simplifies cell browsing in the Query Server by allowing you to peek through pseudocells and cells that do not contain devices, like vias.

BROWSE DEVICELESS CELLS — Controls how cells without devices are reported by BROWSE commands.

BROWSE PSEUDO CELLS — Controls how pseudocells are reported by BROWSE commands.

BROWSE DEVICELESS CELLS

Query Server command.

Controls how BROWSE commands report cells that do not contain any devices throughout their hierarchy (via cells, for example).

Usage

BROWSE DEVICELESS CELLS {NO | YES}

Arguments

- **NO**

A keyword that specifies instances of cells without devices are not reported, but their contents are. This is the default.

- **YES**

A keyword that specifies instances of cells without devices are reported, but their contents are not.

Acknowledgments

OK.

Examples

Assume VIA12 is a cell without devices. This is output of **NET BROWSE SHAPES** with **BROWSE DEVICELESS CELLS NO** specified (device-less cells are transparent):

```
NET BROWSE SHAPES gnd x649
Net_Browse_Shapes 1000
M1
1 1 0 Dec 11 10:27:34 2001
p 1 8
364900 13800
365700 13800
365700 19700
368800 19700
368800 13800
369600 13800
369600 21900
364900 21900
VIA1      <<< (shapes on layer VIA1 come from cell VIA12)
2 2 0 Dec 11 11:22:27 2001
p 1 4
364850 13750
365750 13750
365750 19750
364850 19750
p 2 4
365300 20350
369200 20350
369200 21250
365300 21250
Placements:
0 0 1 Dec 11 11:22:27 2001
X649/X4 NAND
END OF RESPONSE
0 0 0 Dec 11 11:22:27 2001
OK.
```

Output with YES specified:

```
NET BROWSE SHAPES gnd x649
Net_Browse_Shapes 1000
M1
1 1 0 Dec 11 10:27:34 2001
p 1 8
364900 13800
365700 13800
365700 19700
368800 19700
368800 13800
369600 13800
369600 21900
364900 21900
Placements:
0 0 3 Dec 11 10:27:34 2001
X649/X2 VIA12 <<< VIA12 instances are reported, but not the shapes inside
X649/X3 VIA12
X649/X4 NAND
END OF RESPONSE
0 0 0 Dec 11 10:27:34 2001
OK.
```

BROWSE PSEUDO CELLS

Query Server command.

Controls how BROWSE commands report LVS-generated pseudocells.

Usage

BROWSE PSEUDO CELLS {NO | YES}

Arguments

- **NO**

A keyword that specifies instances of pseudocells are not reported, but their contents are. This is the default.

- **YES**

A keyword that specifies instances of pseudocells are reported, but their contents are not.

Description

Causes how the BROWSE commands report pseudocells. The Calibre hierarchical database generates pseudocells with names like ICV_<n> (where n is a number).

This command also affects the [NET NAMES](#) command when the FILTER option is used.

Acknowledgments

OK.

Examples

If placement X34 contains two pseudocell placements of pseudocell ICV_22, the following default output for [NET BROWSE DEVICES](#) might be produced with BROWSE PSEUDO CELLS NO:

```
NET BROWSE DEVICES GND X34
Net_Browse_Devices 1000
Devices:
0 0 6 Dec 11 10:27:34 2001
X34/X0/M6 1 <<< devices in the pseudocell
X34/X0/M7 1
X34/X0/M8 1
X34/X1/M6 1
X34/X1/M7 1
X34/X1/M8 1
Placements:
0 0 0 Dec 11 10:27:34 2001
END OF RESPONSE
0 0 0 Dec 11 10:27:34 2001
OK.
```

Output with YES specified:

```
NET BROWSE DEVICES GND X34
Net_Browse_Devices 1000
Devices:
0 0 0 Dec 11 10:27:34 2001
Placements:
0 0 2 Dec 11 10:27:34 2001
X34/X0 ICV_22
X34/X1 ICV_22
END OF RESPONSE
0 0 0 Dec 11 10:27:34 2001
OK.
```

Device Query Commands

This set of commands queries device information.

Table 5-5. Device Query Commands

Command	Description
DEVICE BAD	Identifies bad device seed shapes.
DEVICE INFO	Returns device data for a layout device instance.
DEVICE LAYOUT	Returns pathnames of layout devices corresponding to a source device.
DEVICE LOCATION	Returns the pathname of a device at a specified location.
DEVICE NAMES	Returns a list of all devices in the current query cell.
DEVICE PINS	Returns layout device pin and net information.
DEVICE SOURCE	Returns pathnames of source devices corresponding to a layout device.
DEVICE TABLE	Returns a table of information about all devices in the design.
DEVICE TEMPLATES USED	Returns a report of the instances formed during device recognition.
DEVICE VALID	Indicates whether the specified device path is valid.

A corresponding command in the Query Server Tcl shell is [dfm::get_device_instances](#). The iterator returned by this command can be queried by the [dfm::get_device_data](#) command for information that is similar to many of the standard commands in the preceding table.

For commands that issue responses, output can be sent to a [RESPONSE FILE](#).

DEVICE BAD

Query Server command. Corresponding Tcl shell command: [dfm::get_devices -bad](#).

Identifies bad device seed shapes.

Usage

DEVICE BAD

Arguments

None.

Description

Returns seed shape layers and vertices for each bad device in the query cell.

Only bad devices in the query cell itself are reported. That is, no bad devices from lower-level cell placements are reported. To obtain all the bad devices in the design, each cell must be queried for bad devices.

Response

```
Device_Bad <precision> // "Device_Bad" and precision
<seed_layer_name_1>    // name of first seed layer containing bad
                        // devices
<n> <n> 0 <date>    // <n> seed shapes follow
p 1 n                 // first bad device seed shape
...                   // vertices of the first shape
...                   // remaining n-1 shapes on this layer
...                   // remaining seed layers on which bad devices are present
```

Acknowledgments

OK., NOK(20), ERROR(101), ERROR(102)

Related Topics

[DEVICE VALID](#)

[RESPONSE FILE](#)

[Device Query Commands](#)

[Results Transformation Commands](#)

DEVICE INFO

Query Server command. Corresponding Tcl shell command:

`dfm::get_device_data -device_info`.

Returns device data for a layout device instance.

Usage

DEVICE INFO *layout_device_path*

Arguments

- *layout_device_path*

A required path to a device instance relative to the query cell.

Description

Returns the device type number, a list of nets attached to pins, a list of property values, and the seed shape for the device specified by the *layout_device_path*.

The number of device pins and properties are the same as what appears in the [DEVICE TABLE](#) output. The layout net paths appear in the same order as the pin names in the device table. The property values listed appear in the same order as the property names in the device table. The layout net paths are reported at the same hierarchical level as the device. That is, they are not shortened to their highest point in the query cell hierarchy. Vertices are in the coordinate space of the viewing cell.

In addition to the usual output, the DEVICE INFO command provides instance-specific model information for devices that use the TEXT MODEL LAYER keyword in the rule file [Device](#) operation. When an instance-specific model exists, it appears in the DEVICE INFO response after the initial device information section and following the string “Model:”.

Response

```
Device_Info <precision> // "Device_Info" and precision
Info:                      // Information begins here
0 0 <n> <date>           // n lines of text follow
<device_type_number>      // 0-based index of this device type in
                           // device table
<layout_net_path_1>        // net connected to first pin in device table
...
<property_value_1>         // value of first device property in device
                           // table
...
[<text_model_name>]          // text model name, if applicable
<seed_layer_name>:          // name of layer containing seed shape
                           // followed by a colon
1 1 0 <date>              // one seed shape follows
p 1 n                      // seed shape with n vertices
...
                           // n vertices of the seed shape
```

Acknowledgments

OK., NOK(9), NOK(33), ERROR(101), ERROR(102)

Examples

Example 1

The following command requests information about device C4 relative to the current query cell:

```
DEVICE INFO X95/X4/X312/C4
```

Example 2

This example shows the difference between instance-specific output from a device classified using TEXT MODEL LAYER and one that does not. A capacitor recognized with this device statement:

```
DEVICE R RPPOLY IPOLY IPOLY TEXT MODEL LAYER DEVTEXT
```

has text R1 from layer DEVTEXT on the device seed shape produces this output:

```
DEVICE INFO R3
Device_Info 1000
Info:
0 0 4 May 8 09:50:45 2002
11
19
22
42
5
Model:
0 0 1 May 8 09:50:45 2002
R1      <<<<< Note this.
RPPOLY
1 1 0 May 8 09:50:45 2002
p 1 4
-15000 0
-12000 0
-12000 3000
-15000 3000
END OF RESPONSE
0 0 0 May 8 09:50:45 2002
OK.
```

The same device without the R1 text produces this output:

```
DEVICE INFO R3
Device_Info 1000
Info:
0 0 4 May 8 09:50:45 2002
11
19
22
42
5
RPPOLY
1 1 0 May 8 09:50:45 2002
p 1 4
-15000 0
-12000 0
-12000 3000
-15000 3000
END OF RESPONSE
0 0 0 May 8 09:50:45 2002
OK.
```

Related Topics

[PLACEMENT INFO](#)

[RESPONSE FILE](#)

[Device Query Commands](#)

[Results Transformation Commands](#)

DEVICE LAYOUT

Query Server command. Corresponding Tcl shell command:

`dfm::get_layout_name -device_instance`.

Returns pathnames of layout devices corresponding to a source device.

Usage

DEVICE LAYOUT *source_device_path*

Arguments

- *source_device_path*

A required pathname of a source device relative to the current query cell.

Description

Returns the pathnames of all corresponding layout devices, relative to the current query cell, to the device specified in the *source_device_path*.

Calibre nmLVS may create many-to-many correspondences between devices. The acknowledgment returns a list of all layout devices corresponding to the given source device. The first device in the list is the primary corresponding device according to the cross-reference.

Acknowledgments

OK: *layout_dev_path1...layout_dev_pathn*, NOK(2), NOK(7), NOK(21), NOK(70)

Examples

This example shows a query of which cell is the query cell, followed by a DEVICE LAYOUT query of a source device path. The corresponding layout device path is returned.

```
status query cell
  OK: Query cell: TOPCELL
device layout X0/X0/M1
  OK: X0/X1/M0
```

Related Topics

[DEVICE SOURCE](#)

[PLACEMENT LAYOUT](#)

[DEVICE INFO](#)

[Device Query Commands](#)

DEVICE LOCATION

Query Server command. Corresponding Tcl shell command: [qs::get_devices_at_location](#).

Returns the pathname of a device at a specified location.

Usage

DEVICE LOCATION *x* *y*

Arguments

- *x*

A required floating-point number representing an x-coordinate in user units in viewing cell space.

- *y*

A required floating-point number representing a y-coordinate in user units in viewing cell space.

Description

Returns the pathname of a device based upon the seed shape directly under location *x* *y* in the viewing cell.

Devices whose device type number is not in the client's [FILTER DEVICES](#) list are ignored. Devices having seed shape layers not in the client's [FILTER DEVICELAYERS](#) list are ignored. If two or more device seed shapes overlap the location, one is arbitrarily selected and the others are ignored.

Filters

These commands control the availability of devices for location reporting.

[FILTER DEVICES](#), [FILTER DEVICELAYERS](#), [FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#), [FILTER DISTANCE](#)

Acknowledgments

OK: *device.pathname*, NOK(6), ERROR(116)

Related Topics

[DEVICE INFO](#)

[Device Query Commands](#)

[Results Transformation Commands](#)

DEVICE NAMES

Query Server command. Corresponding Tcl shell command: [dfm::get_device_instances](#) with [dfm::get_device_data -device_name](#).

Returns a list of all devices in the current query cell. The devices are listed in an arbitrary order.

Usage

DEVICE NAMES [FLAT]

Arguments

- FLAT

An optional keyword that specifies to report all devices in the query cell's internal hierarchy, relative to the query cell. By default, only the devices at the primary level of the query cell are reported.

Filters

[FILTER DEVICES](#) controls the availability of name reporting.

Response

- The following response occurs:

```
Device_Names <precision>          // "Device_Names" and precision
Devices:                           // "Devices:"
0 0 <n> <date>                  // n lines of text follow
<device_name_1> <device_type_1> // first device name and type
...
<device_name_n> <device_type_n> // last device name and type
```

Acknowledgments

OK., NOK(18), NOK(19), ERROR(101), ERROR(102)

Related Topics

[PLACEMENT NAMES](#)

[RESPONSE FILE](#)

[Device Query Commands](#)

DEVICE PINS

Query Server command. Corresponding Tcl shell command: [dfm::get_device_data -pin_info](#).

Returns layout device pin and net information.

Usage

DEVICE PINS *layout_device_path*

Arguments

- *layout_device_path*

A required pathname of a layout device relative to the current query cell.

Description

Returns the device type number, a list of the nets attached to pins, and the locations of pin shapes for the device given in the *layout_device_path*. The [Mask SVDB Directory](#) statement in the rule file requires the PINLOC keyword for pin location information to be available for the Query Server.

Pin shapes are reported within an area equal to the rectangular extent of the seed shape plus the [MARKER SIZE](#). Returned vertex coordinates are in viewing cell space.

The returned layout net paths appear in the same order as the pin names in the [DEVICE TABLE](#). The property names are as in the device table. The net paths are reported at the same hierarchical level as the device. That is, they are not shortened to their highest point in the query cell hierarchy. This command is only available when pin location information is saved to the PHDB.

This command provides pin location information if [LVS Push Devices SEPARATE PROPERTIES YES](#) is specified and the query context is the top-level cell. In this case, the *layout_device_path* must be a flattened pathname for instances that are not at the top level. If used in the context of a lower-level cell in the pushdown separated properties (PDSP) flow, ERROR(189) is given.

Response

```
Device_Pins <precision> // "Device_Pins" and precision
Info: // "Info:"
0 0 <n> <date> // n lines of text follow
<device_type_number> // 0-based index of this device type in
// device table
<layout_net_path_1> // net connected to first pin in device table
... // nets connected to other pins in device table order
<pin_layer_name> // name of layer containing pin shape
np np 0 <date> // np pin shapes follow
p 1 n // seed shape with n vertices
...
...
<pin_layer_name> // name of layer containing last pin shape
... // polygon information for last pin shape
```

Acknowledgments

OK., NOK(18), NOK(19), ERROR(101), ERROR(102), ERROR(159), ERROR(160)

Related Topics

[NET PINS](#)

[LAYOUT NETLIST PIN LOCATIONS](#)

[RESPONSE FILE](#)

[Device Query Commands](#)

[Results Transformation Commands](#)

DEVICE SOURCE

Query Server command. Corresponding Tcl shell command:

`dfm::get_source_name -device_instance.`

Returns pathnames of source devices corresponding to a layout device.

Usage

DEVICE SOURCE *layout_device_path*

Arguments

- *layout_device_path*

A required pathname of a layout device relative to the current query cell.

Description

Returns the pathnames of all corresponding source devices, relative to the current query cell, to the device specified in the *layout_device_path*.

Calibre nmLVS may create many-to-many correspondences between devices. The acknowledgment returns a list of all source devices corresponding to the given layout device. The first device in the list is the primary corresponding device according to the cross-reference.

See also [DEVICE LAYOUT](#).

Acknowledgments

OK: *source_dev_path1...source_dev_pathn*, NOK(2), NOK(7), NOK(21), NOK(71)

Examples

This example shows a query of which cell is the query cell, followed by a DEVICE SOURCE query of a layout device path. The corresponding source device path is returned.

```
status query cell
    OK: Query cell: TOPCELL
device source X0/X1/M0
    OK: X0/X0/M1
```

Related Topics

[DEVICE LAYOUT](#)

[Device Query Commands](#)

DEVICE TABLE

Query Server command. Corresponding Tcl shell command: [qs::device_table](#).

Returns a table of information about all devices in the design.

Usage

DEVICE TABLE

Arguments

None.

Description

Returns a table of device information as discussed under “[Devices](#)” on page 24.

The device table references all devices recognized in the design and is independent of the current context.

The device type is one of the following constants: DIODE, RESISTOR, CAPACITOR, MOS, BIPOLAR, USER.

If any of model name, netlist element name, or netlist model name were not specified in the rule file, they are represented in the response by the string “null”.

The pin info lines contain the following entries separated by white space: pin_name, pin_layer, list_number, where list_number is the ordinal of the swap list in the [DEvice](#) statement:

```
pin_name layer_name list_number
```

If pin groups are used in the DEDevice statement instead of individual pins, then there are two numbers indicating the pin group element of the pin. The numbers can be thought of as the column and row indices of a matrix. For example, assume this is the DEDevice statement:

```
DEVICE user seed W(a1) X(a2) Y(b1) Z(b2) ([a1 b1] [a2 b2])
```

The pin groups are row vectors of a 2×2 matrix, where the rows are shown between brackets. The DEVICE TABLE command would return this:

```
a1 W 1 1
a2 X 1 2
b1 Y 2 1
b2 Z 2 2
```

Pin a1 corresponds to the first column and first row of the matrix, and so forth.

Pin and property names are lowercase. The layer names have the same case as in the rule file.

The precision value in the output response header is multiplied by the [MAGNIFY RESULTS](#) factor. Magnification factors can be useful in certain third-party timing analysis tool flows.

Response

```
Device_Table <precision> // "Device_Table" and precision
Table Count           // "Table count"
0 0 1 <date>        // 1 line of text follows
<k>                  // k device entries are defined (indexed 0
                      // through k-1)
Device Entry 0       // "Device Entry" followed by the device number
0 0 <n> <date>      // n lines of text follow
<device_layer_name> // name of seed layer for this device
<device_type>        // basic type of device (see description for
                      // values)
<element_name>       // element name of the device
<model_name>         // model name of the device; "(null)" if
                      // none
<netlist_element_name> // netlist element name of the device;
                      // "(null)" if none
<netlist_model_name> // netlist model name of the device;
                      // "(null)" if none
<pin_count>          // number of pins of device
<pin_info_0>          // information about first pin (see description)
...
<aux_layer_count>    // number of auxiliary layers associated
                      // with device
<aux_layer_0>          // name of first auxiliary layer
                      // remaining auxiliary layers
<property_count>     // number of device properties
<property_name_0>     // first property name
...
                      // remaining property names
...
                      // remaining device entries 1 through k-1
```

Acknowledgments

OK., ERROR(101), ERROR(102)

Related Topics

[DEVICE INFO](#)

[DEVICE TEMPLATES USED](#)

[ANNOTATED DEVICE LAYER MAP](#)

[RESPONSE FILE](#)

[Device Query Commands](#)

[Results Transformation Commands](#)

DEVICE TEMPLATES USED

Query Server command. Corresponding Tcl shell command: [qs::device_templates_used](#).

Returns a report of the types of instances formed during device recognition. Device types and subtypes that are not used to generate instances (as in LVS Box cells) are not output.

Usage

DEVICE TEMPLATES USED

Arguments

None.

Response

- The following response occurs:

```
Device_Templates_Used <precision> // "Device_Templates_Used"
// and precision
Templates:
0 0 <n> <date> // Templates:
// number of lines to follow
// and date stamp.
<type>[(<subtype>)] <k> <HDC> (<FDC>)
// device type, optional subtype
// in parentheses, DEVICE TABLE
// index number,
// hierarchical instance count,
// flat instance
// count in parentheses.
// remaining device entries.
...
...
...
END OF RESPONSE
0 0 0 <date>
```

Acknowledgments

OK., ERROR(101), ERROR(102)

Examples

```
Device_Templates_Used 1000
Templates:
0 0 4 Aug 15 13:26:43 2012
mn 2 21 (140)
mp 3 21 (140)
r(pl) 4 1 (3)
r(d) 5 1 (22)
END OF RESPONSE
0 0 0 Aug 15 13:26:43 2012
```

Related Topics

[DEVICE TABLE](#)

[RESPONSE FILE](#)

[Device Query Commands](#)

DEVICE VALID

Query Server command. Corresponding Tcl shell command: [qs::device_valid](#).

Indicates whether the specified device is valid.

Usage

DEVICE VALID {LAYOUT | SOURCE} *device_path*

Arguments

- **LAYOUT**

A keyword that specifies the *device_path* is for the layout.

- **SOURCE**

A keyword that specifies the *device_path* is for the source.

- ***device_path***

A required pathname to a device relative to the query cell.

Description

Indicates whether the layout or source device at the specified *device_path* is valid. This command is identical to [PLACEMENT VALID](#).

PHDBs and XDBs (when available) are searched for layout device names. XDBs (when available) are searched for source names.

Acknowledgments

OK: *device_path*, NOK(7), NOK(8), NOK(9)

Related Topics

[DEVICE BAD](#)

[NET BROWSE DEVICES](#)

[PLACEMENT BROWSE DEVICES](#)

[Device Query Commands](#)

Net Query Commands

This set of commands queries net information.

Table 5-6. Net Query Commands

Command	Description
NET BROWSE DEVICES	Returns information about devices on a layout net.
NET BROWSE NETS	Returns information about lower-level nets connected to a layout net.
NET BROWSE PORTS	Returns information about ports on a layout net.
NET BROWSE SHAPES	Returns information about shapes on a layout net.
NET DEVICENAMES	Returns all device instance paths on a layout net.
NET EXTERNAL SHAPES	Returns vertices of external net shapes connected to the current query cell.
NET LAYERS	Returns the names of layers on a specified net in the current query cell.
NET LAYOUT	Returns the pathnames of layout nets in the current query cell that correspond to a source net.
NET LOCATION	Returns the pathnames of nets at a specified location.
NET NAMES	Returns a list of net names in the current query cell.
NET NOT CONNECTED	Returns placements of a layout cell for which a specified net is not connected to a reference net.
NET PINS	Returns names and locations of intentional device pins on a layout net.
NET PORTNAMES	Returns the names of all ports on a layout net.
NET PORTS	Returns port names and locations on a layout net.
NET SHAPES	Returns shapes of a layout net.
NET SOURCE	Returns the pathnames of source nets that correspond to a layout net.
NET TEXT MAP	Returns the mapping of layout net names to system-generated node numbers.
NET TRACE	Returns the complete pathname of the highest representative of a net.
NET VALID	Indicates whether the specified net path is valid.

For commands that issue responses, output can be sent to a [RESPONSE FILE](#).

A corresponding command in the Query Server Tcl shell is [dfm::get_nets](#). The iterator returned by this command can be queried using the [dfm::get_data](#) command for information about nets that is similar to many of the standard commands in the preceding table.

NET BROWSE DEVICES

Query Server command.

Returns information about devices on a layout net.

Usage

NET BROWSE DEVICES *layout_net_path* [*placement_path*]

Arguments

- *layout_net_path*
A required pathname of a layout net relative to the query cell.
- *placement_path*
An optional pathname of a cell placement relative to the query cell.

Description

Returns information about devices on the *layout_net_path* and that optionally are in a specified *placement_path*.

The *layout_net_path* is traced upward to its highest hierarchical representative before listing of devices occurs. The response also shows placements that contain additional devices on the specified net down the hierarchy.

Filters

[FILTER DEVICES](#) controls the availability of device reporting.

Response

```
Net_Browse_Devices <precision> // "Net_Browse_Devices" and precision
Devices:                      // "Devices:"
0 0 k <date>                 // 0 0 number_of_devices datestamp
<device_name_1> <device_index_1>
// name of the device, index into the device table
... additional devices ...     // additional names and indices
Placements:                   // "Placements:"
0 0 k <date>                 // 0 0 number_of_placements datestamp
<placement_name_1><cell_name> // placements containing additional
                                // devices.
...
<additional placements>      // additional placements.
...
```

Acknowledgments

OK., NOK(5), NOK(32), NOK(33), NOK(34), NOK(35)

Related Topics

[RESPONSE FILE](#)

Net Query Commands

NET BROWSE NETS

Query Server command.

Returns information about lower-level nets connected to a layout net.

Usage

NET BROWSE NETS *layout_net_path* [*placement_path*]

Arguments

- *layout_net_path*
A required pathname of a layout net relative to the query cell.
- *placement_path*
An optional pathname of a cell placement relative to the query cell.

Description

Returns net names and placements connected to the *layout_net_path* at lower levels of the hierarchy and that optionally are connected to a specified *placement_path*.

The *layout_net_path* is traced upward to its highest hierarchical representative before listing of nets occurs. The response also shows placements that contain additional nets on the specified net down the hierarchy.

Filters

[FILTER LAYERS](#) and [FILTER {IN | OUT} LAYERS](#) control the availability of net reporting.

Response

```
Net_Browse_Nets <precision>          // "Net_Browse_Nets" and precision
Nets:                                // "Nets:"
 0 0 k <date>                      // 0 0 number_of_nets datestamp
 <net_name_1>                        // name of the net
 ... additional nets ...             // additional names
 Placements:                          // "Placements:"
 0 0 k <date>                      // 0 0 number_of_placements datestamp
 <placement_name_1> <cell_name>    // placements containing additional
                                     // nets.
 ...
 <additional placements>           // additional placements.
 ...
```

Acknowledgments

OK., NOK(5), NOK(32), NOK(33), NOK(35)

Related Topics

[NET BROWSE SHAPES](#)

[NET NAMES](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

[Results Transformation Commands](#)

NET BROWSE PORTS

Query Server command.

Returns information about ports on a layout net.

Usage

NET BROWSE PORTS *layout_net_path* [*placement_path*]

Arguments

- *layout_net_path*
A required pathname of a layout net relative to the query cell.
- *placement_path*
An optional pathname of a cell placement relative to the query cell.

Description

Return information about ports (pins) on the *layout_net_path* and that optionally are in a specified *placement_path*.

The *layout_net_path* is traced upward to its highest hierarchical representative before listing of ports occurs. The response also shows placements that contain additional ports on the specified net down the hierarchy.

If a port is not named by a port text object, this is shown with the string “(unnamed port)”.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#)

Response

```
Net_Browse_Ports <precision>      // "Net_Browse_Ports" and precision
Ports:                           // "Ports:"
0 0 k <date>                   // 0 0 number_of_ports datestamp
<port_name_1> <device_index_1> // name of the port
... additional ports ...          // additional names
Placements:                      // "Placements:"
0 0 k <date>                   // 0 0 number_of_placements datestamp
<placement_name_1> <cell_name> // placements containing additional
                                // ports.
...
<additional placements>        // additional placements.
...
```

Acknowledgments

OK., NOK(5), NOK(12), NOK(32), NOK(33), NOK(35)

Related Topics

[NET PORTNAMES](#)
[NET PORTS](#)
[RESPONSE FILE](#)
[Net Query Commands](#)

NET BROWSE SHAPES

Query Server command.

Returns information about shapes on a layout net.

Usage

NET BROWSE SHAPES *layout_net_path* [*placement_path*]

Arguments

- *layout_net_path*
A required pathname of a layout net relative to the query cell.
- *placement_path*
An optional pathname of a cell placement relative to the query cell.

Description

Returns information about shapes making up the net corresponding to the *layout_net_path* and that optionally are in a specified *placement_path*.

The *layout_net_path* is traced upward to its highest hierarchical representative before listing of shapes occurs. The response also shows placements that contain additional shapes on the specified net down the hierarchy.

Vertex coordinates are returned in the space of the viewing cell.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#)

Response

```
Net_Browse_Shapes <precision> // "Net_Browse_Shapes" and precision
Layer_name_1 // first layer on which net appears
<k> <k> 0 <date> // k polygons on this layer
p 1 4 // first polygon on this layer
... // vertices of the first polygon
... // remaining k-1 polygons on this layer
...
Placements: // "Placements:"
0 0 k <date> // 0 0 number_of_placements datestamp
<placement_name_1><cell_name> // placements containing additional
// devices.
...
<additional placements> // additional placements.
...
```

Acknowledgments

OK., NOK(5), NOK(29), NOK(32), NOK(33), NOK(35)

Related Topics

[NET BROWSE NETS](#)
[NET EXTERNAL SHAPES](#)
[NET SHAPES](#)
[RESPONSE FILE](#)
[Net Query Commands](#)

NET DEVICENAMES

Query Server command. Corresponding Tcl shell command: [qs::net_devicenames](#).

Returns all device instance paths on a specified layout net.

Usage

NET DEVICENAMES *layout_net_name*

Arguments

- *layout_net_name*

A required name of a layout net in the current query cell.

Filters

[FILTER DEVICES](#)

Response

- The following response occurs:

```
Net_Devicenames <precision>      // "Net_Devicenames" and precision
Devices:                         // "Devices:"
0 0 k <date>                   // 0 0 number_of_devices datestamp
<device_path_1> <device_index_1> // path of the device, index in the
                                    // device table
...
<additional_paths>              // additional paths and indices
...
```

Acknowledgments

OK., NOK(5), NOK(33), NOK(34)

Related Topics

[NET BROWSE DEVICES](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

[DEVICE TABLE](#)

NET EXTERNAL SHAPES

Query Server command.

Returns vertices of external net shapes connected to the current query cell.

Usage

NET EXTERNAL SHAPES

Arguments

None.

Description

Determines which nets in the current query cell connect upward in the design hierarchy. The command then flattens all such nets into the coordinate system of the viewing cell and returns the vertices of the corresponding shapes within the query cell as a response.

Only shapes on the layers allowed by the current setting of the [FILTER LAYERS](#) command are returned. Only pins allowed by the current setting of the [FILTER WINDOW](#) command are returned.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#), [FILTER WINDOW](#), [FILTER CULL](#)

Response

```
Net_Shapes <precision> // "Net_Shapes" and precision
Layer_name_1           // first layer on which shapes were found
<k> <k> 0 <date>      // k polygons on this layer
p 1 4                  // first polygon on this layer
...
...                   // vertices of the first polygon
...
...                   // remaining k-1 polygons on this layer
...
...                   // remaining layers on which shapes were found.
```

Acknowledgments

OK., NOK(29), NOK(33), ERROR(101), ERROR(102)

Related Topics

[NET BROWSE SHAPES](#)

[NET SHAPES](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

[Results Transformation Commands](#)

NET LAYERS

Query Server command. Corresponding Tcl shell command: [qs::net_layers](#).

Returns the names of layers on a specified net in the current query cell.

Usage

NET LAYERS *layout_net_path*

Arguments

- *layout_net_path*

A required pathname of a layout net relative to the query cell.

Description

Returns the names of layers with shapes on the net corresponding to the *layout_net_path*.

The net pathname specified need not be the highest representative of the net in query cell. It is traced through out the query cell, not just downward from the given path.

Filters

None.

Response

```
Net_Layers <precision> // "Net_Layers" and precision
Layers:           // first layer on which shapes were found
 0 0 n <date>   // n unique layers on this net
 Layer_name_1     // first layer on this net
 ...             // remaining n-1 layers on this net
```

Acknowledgments

OK., NOK(5)

Related Topics

[NET BROWSE SHAPES](#)

[NET SHAPES](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

NET LAYOUT

Query Server command. Corresponding Tcl shell command: [dfm::get_layout_name -net](#).

Returns the pathnames of layout nets in the current query cell that correspond to a source net.

Usage

NET LAYOUT *source_net_path*

Arguments

- *source_net_path*

A required pathname of a source net relative to the query cell.

Description

Returns the pathname of all corresponding layout nets in the query cell that correspond to the source net specified in the *source_net_path*.

Calibre nmLVS may create one-to-many or many-to-many correspondences between nets. The acknowledgment returns a list of all layout nets corresponding to the given source net. The first net in the list is the primary corresponding net according to the cross-reference. The remaining nets are listed in an arbitrary order.

See the [NET SOURCE](#) command for a description of circumstances under which Calibre nmLVS removes nets for comparison purposes. The Query Server does not find incorrect nets or nets that have been removed due to device reduction, but it does find unmatched nets. Incorrect and unmatched nets are shown in the LVS Report.

Acknowledgments

OK: *layout_net_path1...layout_net_path_N*, NOK(2), NOK(4), NOK(21), NOK(44), NOK(62), NOK(64), NOK(66)

Related Topics

[NET BROWSE NETS](#)

[Net Query Commands](#)

NET LOCATION

Query Server command. Corresponding Tcl shell command: [qs::get_nets_at_location](#).

Returns the pathnames of nets at a specified location.

Usage

NET LOCATION *x y*

Arguments

- *x y*

A required pair of floating-point numbers in user units representing coordinates in viewing cell space.

Description

Returns the pathnames of nets in the current query context that intersect the *x y* coordinates.

The search for nets considers all shapes obtained by flattening the hierarchy directly below the query point. Shapes that are not on a layer in the current client's [FILTER LAYERS](#) list are ignored. Layers in the filter layer list that are not connectivity layers are ignored. If two or more net shapes overlap the specified location, all nets are selected and returned.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#), [FILTER DISTANCE](#), [FILTER WINDOW](#)

Acknowledgments

OK: *net_path_name*, NOK(14), NOK(116)

Related Topics

[Net Query Commands](#)

[Results Transformation Commands](#)

NET NAMES

Query Server command. Corresponding Tcl shell command: [dfm::get_nets](#) with [dfm::get_data -net_name](#).

Returns a list of net names in the current query cell.

Usage

NET NAMES {{**FLAT** | **UNIQUE** [**FILTER**]}} | **FILTER**

Arguments

- **FLAT**

A keyword that specifies to flatten the internal hierarchy of the query cell before returning the net names.

- **UNIQUE**

A keyword that specifies to return net names from lower-level cells in the current query cell that do not connect to higher-level cells.

- **FILTER**

A keyword that specifies to return only net names that have geometry that satisfies the filters provided by the [FILTER LAYERS](#) and [FILTER WINDOW](#) commands. This keyword can be specified by itself or optionally with **UNIQUE**. If specified with **UNIQUE**, then the behaviors of both keywords apply.

Description

Returns a list of all net names in the current (optionally flattened) query cell. The order in which the nets are listed is arbitrary.

If **FLAT** is used, nets of cells placed in the query cell at any level are reported with a query-cell-based net name leading to the cell instance containing the net. The resulting list contains the same design-wide net under several names. For example, if the net CLK in the top cell connects to the net CLK1 in cell instance X23, the resulting list contains both CLK and X23/CLK.

The presence of pseudocells and device-less cells may affect the visibility of nets in your design. If the **FILTER** option is set, and the **UNIQUE** option is not, you can set the [BROWSE PSEUDO CELLS](#) and [BROWSE DEVICELESS CELLS](#) options to report nets from lower level cells.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#), [FILTER WINDOW](#)

Response

```
Net_Names <precision> // "Net_Names" and precision
Nets: // "Nets:"
 0 0 <n> <date> // n lines of text follow
  <net_name_1> // first net name in current context
  ...
  <net_name_n> // intermediate net names in current context
                // last net name in current context
```

Acknowledgments

OK., NOK(15), NOK(16), NOK(17), NOK(33), ERROR(101), ERROR(102), ERROR(130)

Examples

If a cell has 4 nets VSS, VDD, 3, and 4, and nets VSS and VDD appear on the layers MET1, but layers 3 and 4 do not, then the following filtering occurs:

```
NET NAMES
Net_Names 1000
Nets:
 0 0 4 Feb 28 16:54:12 2005
VSS
VDD
3
4
END OF RESPONSE
0 0 0 Feb 28 16:54:12 2005
OK.
```

```
FILTER LAYERS MET1
OK.
```

```
NET NAMES FILTER
Net_Names 1000
Nets:
 0 0 2 Feb 28 16:55:00 2005
VSS
VDD

END OF RESPONSE
0 0 0 Feb 28 16:55:00 2005
OK.
```

FILTER may be used in conjunction with the **UNIQUE** option to produce filtered net names from lower levels of hierarchy in the query cell. Additionally, if only net VSS appears inside the coordinate window (0,0) (10,10), then the following occurs:

```
FILTER WINDOW INCLUDE 0 0 10 10
OK.
```

```
NET NAMES FILTER
Net_Names 1000
Nets:
0 0 1 Feb 28 16:55:14 2005
VSS
END OF RESPONSE
0 0 0 Feb 28 16:55:14 2005
OK.
```

Related Topics

[NET BROWSE NETS](#)

[NET TEXT MAP](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

NET NOT CONNECTED

Query Server command. Corresponding Tcl shell command: [qs::net_not_connected](#).

Returns placements of a layout cell for which a specified net is not connected to a reference net.

Usage

NET NOT CONNECTED *reference_net* [*reference_net* ...] *layout_cell target_net*

Arguments

- ***reference_net***

A required net name to which connections are checked. More than one *reference_net* may be specified. A hierarchical net path is accepted.

- ***layout_cell***

A required name of a layout cell.

- ***target_net***

A required net name in the current query cell to check for connections to any *reference_net*.

Description

Returns a list of all placements of the *layout_cell* for which the *target_net* is not connected to any *reference_net*. If *target_net* is connected in the hierarchy to *reference_net*, an empty list is returned.

Cell names can be original cell names or FAUX BIN cells as identified in the LVS run transcript.

The order in which the returned placements are listed is arbitrary.

A path that extends upward in the hierarchy is searched from its highest point in the hierarchy. For example, if net X1/X3/2 is used as the *reference_net*, and this net connects to the net GND in the current query cell, the commands NET NOT CONNECTED X1/X3/2 NANDCELL GND and NET NOT CONNECTED GND NANDCELL GND are equivalent.

Response

```
Net_Not_Connected <precision> // "Net_Not_Connected" and precision
Placements:           // "Placements:"
 0 0 <n> <date>      // n lines of text follow
 <placement_name_1>    // first placement name rooted in current
                       // context
 ...
 <placement_name_n>    // intermediate names rooted in current
                       // context
                           // last placement name rooted in current
                           // context
```

Acknowledgments

OK., NOK(5), NOK(33), NOK(36), NOK(37), NOK(38), NOK(39)

Examples

NET NOT CONNECTED VCC VCCA NANDCELL VCC

When executed from the top-level cell, this command finds placements of NANDCELL where the VCC net in NANDCELL is not connected to the top-level VCC or VCCA nets.

Related Topics

[RESPONSE FILE](#)

[Net Query Commands](#)

NET PINS

Query Server command. Corresponding Tcl shell command: [qs::net_pins](#).

Returns names and locations of intentional device pins on a layout net.

Usage

NET PINS *layout_net_path*

Arguments

- *layout_net_path*

A required path to a layout net referenced from the current query cell.

Description

Returns the intentional device pins attached to the net associated with the *layout_net_path*.

Each pin is represented as a device pathname, a device type number (that is, the 0-based index of the device in the [DEVICE TABLE](#)), and a marker square centered at the pin location on the pin layer. The location on which the square is centered is the lowest leftmost point where the pin shape touches or overlaps the device seed shape. The actual shape of the pin is not given. Vertex coordinates are given in viewing cell space.

Under certain circumstances, the device recognizer assigns more than one logical device pin to the same physical pin shape. In this case, a marker is present for each of the logical device pins, but they all center on the same location. The size of the marker squares is determined by the current value of the [MARKER SIZE](#) argument.

The pin location information is only available if the [Mask SVDB Directory](#) PINLOC or CCI options are specified in the rule file that generated the database.

Only pin shapes on the layers allowed by the current [FILTER LAYERS](#) setting are returned. Only pins allowed by the current [FILTER WINDOW](#) setting are returned. The pins are listed in the “Pins:” check section in an arbitrary order. Each line is a white-space-separated list of the following items:

```
pin_number // number of pin in the list of pins
device_path // instance pathname ending with device instance name
device_type // index of the device in the device table
pin_index   // index of the pin in the ordered list of device pins
```

See “[DEVICE TABLE](#)” on page 326 for the meaning of device_type and pin_index.

The pin_number is not strictly necessary since it simply increments by one for each line, but is included for convenience.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#), [FILTER WINDOW](#)

Response

```
Net_Pins <precision> // "Net_Pins" and precision
Pins: // "Pins:"
0 0 <n> <date> // n lines of text follow (n pins were found)
<pin_info_1> // pin number 1 (see description for definition of
                // pin_info)
...
<pin_info_n> // pin number n
Layer_name_1 // first layer on which pins are present
<k> <k> 0 <date> // k pins on this layer
p <i> 4 // i is the number of this pin in the "Pins:" check
        // section
...
... // four vertices of the first square
... // remaining m-1 pins on this layer
... // remaining layers on which pins are present
```

Acknowledgments

OK., NOK(1), NOK(5), NOK(33), ERROR(101), ERROR(102)

Related Topics

NET PORTS

NET PORTNAMES

RESPONSE FILE

Net Query Commands

NET PORTNAMES

Query Server command. Corresponding Tcl shell command: [dfm::get_ports -flat -net -list](#).

Returns the names of all ports on a layout net.

Usage

NET PORTNAMES *layout_net_path*

Arguments

- *layout_net_path*

A required path to a layout net referenced from the current query cell.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#)

Response

- The following response occurs:

```
Net_Portnames <precision> // "Net_Portnames" and precision
Ports:
// "Ports:"
0 0 k <date>           // 0 0 number_of_ports datestamp
<port_name_1>           // name of the port
...
<additional_ports>       // additional ports.
...
```

Acknowledgments

OK., NOK(5), NOK(12), NOK(33)

Related Topics

[NET PORTS](#)

[NET PINS](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

NET PORTS

Query Server command. Corresponding Tcl shell command: [dfm::get_ports -flat -net -list](#).

Returns port names and locations on a layout net.

Usage

NET PORTS *layout_net_path*

Arguments

- *layout_net_path*

A required path to a layout net referenced from the current query cell.

Description

Returns the names of the ports attached to the net associated with the *layout_net_path* and a marker square for each port. The ports are listed in an arbitrary order. The ports returned are not only those in the query cell, but those in any cell placement into which the net extends.

A port may have more than one location. Each location is listed separately in the Ports: section of the response, and a separate marker is given on the appropriate layer for each location. The size of the marker is specified with the [MARKER SIZE](#) command.

Only ports on the layers allowed by the current [FILTER LAYERS](#) setting are returned. Only ports allowed by the current [FILTER WINDOW](#) setting are returned.

The number <i> which appears in the polygon entry for each marker square is the index number of the corresponding port in the Ports: section of the response. Information on any given port may be obtained by the [PORT INFO](#) command.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#), [FILTER WINDOW](#)

Response

```
Net_Ports <precision> // "Net_Ports" and precision
Ports:           // "Ports:"
  0 0 <n> <date>      // n lines of text follow (n ports were found)
  1 <port_name_1>       // index of first port and first port name
  ...
  <n> <port_name_n>     // more indices and ports
  Layer_name_1          // index of last port last port name
                        // first unfiltered layer on which ports are
                        // present
  <k> <k> 0 <date>      // k ports on this layer
  p <i> 4                // i is the number of this port in the "Ports:"
                        // check section
  ...
  ...                  // four vertices of the first square
  ...                  // remaining m-1 ports on this layer
  ...                  // remaining unfiltered layers on which ports are
                        // present
```

Acknowledgments

OK., NOK(5), NOK(12), NOK(33), ERROR(101), ERROR(102)

Related Topics

[NET PORTNAMES](#)

[NET PINS](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

[Results Transformation Commands](#)

NET SHAPES

Query Server command.

Returns shapes of a layout net.

Usage

NET SHAPES *layout_net_path*

Arguments

- *layout_net_path*

A required path to a layout net referenced from the current query cell.

Description

Returns the shapes of the net associated with the *layout_net_path*. The shapes are flattened into the coordinate system of the viewing cell. The *layout_net_path* is traced throughout the query cell, not just downward from the given path.

Only shapes on the layers allowed by the current FILTER LAYERS and FILTER WINDOW settings are returned. Polygons in the response are limited to 4096 vertices by default. Polygons that have more than this number of vertices are segmented into polygons that obey the MAXIMUM VERTEX COUNT setting.

Any layer that contains node numbers can have a polygon returned by this command, including Connect layers and derived layers that get connectivity from Stamp operations or from node-preserving layer constructors (for example NOT and AND).

Filters

FILTER LAYERS, FILTER {IN | OUT} LAYERS, FILTER WINDOW, FILTER CULL

Response

```
Net_Shapes <precision> // "Net_Shapes" and precision
Layer_name_1           // first layer on which the net has a presence
<k> <k> 0 <date>      // k polygons on this layer
p 1 4                  // first polygon on this layer
...
...                   // vertices of the first polygon
...
...                   // remaining k-1 polygons on this layer
...
...                   // remaining layers on which net has a presence.
```

Acknowledgments

OK., NOK(5), NOK(29), NOK(33), ERROR(101), ERROR(102)

Examples

The following command requests shapes of net CLOCK1 relative to the current query cell:

```
NET SHAPES X12/X45/CLOCK1
```

Related Topics

- [NET BROWSE SHAPES](#)
- [NET EXTERNAL SHAPES](#)
- [RESPONSE FILE](#)
- [Net Query Commands](#)
- [Results Transformation Commands](#)

NET SOURCE

Query Server command. Corresponding Tcl shell command: [dfm::get_source_name -net](#).

Returns the pathnames of source nets that correspond to a layout net.

Usage

NET SOURCE *layout_net_path*

Arguments

- *layout_net_path*

A required pathname of a layout net relative to the query cell.

Description

Returns the pathnames of all corresponding source nets to the *layout_net_path* in the corresponding source cell.

Calibre nmLVS may create one-to-many or many-to-many correspondences between nets. The acknowledgment returns a list of all source nets corresponding to the given layout net. The first net in the list is the primary corresponding net according to the cross-reference.

Calibre nmLVS may discard certain nets during comparison. The Query Server has limited ability to cross-reference nets that have been discarded during comparison. Nets that can be traced in the layout database are matched to a lower level net if an unambiguous correspondence point can be found (PHDB database must be present).

Net names may also be lost during comparison for the following reasons:

- A lower level net is flattened to a higher level in the hierarchy where it connects to a higher level net. If so, Calibre nmLVS-H uses the name of the higher-level net.
- A net is filtered out for comparison purposes if it only connects devices that are combined into a structure used for comparison purposes (that is, device reduction and logic gate recognition).
- A net is filtered out for comparison purposes if it does not connect to more than one device. This includes nets that are left floating after device filtering.

Query Server does not find incorrect nets or nets that have been removed due to device reduction, but it does find unmatched nets. Incorrect and unmatched nets are shown in the LVS report.

Acknowledgments

OK: *source_net_path_1...source_net_path_N*, NOK(2), NOK(3), NOK(21), NOK(45), NOK(63), NOK(65), NOK(67)

Related Topics

[NET LAYOUT](#)

[LAYOUT NETLIST NAMES](#)

[Net Query Commands](#)

NET TEXT MAP

Query Server command. Corresponding Tcl shell command: [qs::net_text_map](#).

Returns the mapping of layout net names to system-generated node numbers.

Usage

NET TEXT MAP [INVALID]

Arguments

- INVALID

An optional keyword that specifies only the correspondence for invalid nets is returned.

Description

Generates a list of layout net names mapped to Calibre nmLVS generated net numbers.

Certain texts are not used for SPICE netlist net names by Calibre. These text names are not included in the map by default. The INVALID keyword causes the command to return only those names.

Response

```
Net_Text_Map <precision> // "Net_Text_Map" and precision
Net Texts:
0 0 <n> <date>           // n lines of text follow
<net_text> <net_number> // net_text is the name of net net_number
```

Acknowledgments

OK., NOK(46)

Related Topics

[NET NAMES](#)

[RESPONSE FILE](#)

[Net Query Commands](#)

NET TRACE

Query Server command. Corresponding Tcl shell command: [qs::net_trace](#).

Returns the complete pathname of the highest representative of a net.

Usage

NET TRACE *layout_net_path*

Arguments

- *layout_net_path*

A required pathname of a layout net relative to the query cell.

Description

Returns the pathname of the highest representative of the net associated with the *layout_net_path* in the query cell, as well as all intermediate net pathnames between the highest net and the given net.

The trace proceeds upward from the given *layout_net_path* until it reaches the highest hierarchical representative within the current query cell. This does not imply that the net does not also descend into other instances below the query cell instances, or that the net does not descend into the instances on the given path by more than one pin, or that the net does not connect through an external pin of the query cell into higher levels of the design.

Acknowledgments

OK: *net_path net_name_1...net_name_N*, NOK(5)

Examples

Consider the following command and acknowledgment:

```
NET TRACE X1/X2/X3/X4/X5/clk5
OK: X1/X2/clk2 X1/X2/X3/clk3 X1/X2/X3/X4/clk4 X1/X2/X3/X4/X5/clk5
```

This response means the following:

Net X1/X2/clk2 is the highest representative of X1/X2/X3/X4/X5/clk5 relative to the current query cell. That is, net X1/X2/clk2 is not connected to an external pin of instance X1/X2.

Net X1/X2/clk2 connects to net X1/X2/X3/clk3.

Net X1/X2/X3/clk3 connects to net X1/X2/X3/X4/clk4.

Net X1/X2/X3/X4/clk4 connects to net X1/X2/X3/X4/X5/clk5.

Net X1/X2/clk2 may be connected not only to net X1/X2/X3/clk3, but also to some net like X1/X2/X3/sync by a second pin in instance X1/X2/X3.

Related Topics

[NET BROWSE NETS](#)

[Net Query Commands](#)

NET VALID

Query Server command. Corresponding Tcl shell command: [qs::net_valid](#).

Indicates whether the specified net path is valid.

Usage

NET VALID {LAYOUT | SOURCE} *net_path*

Arguments

- **SOURCE**

A keyword that specifies the *net_path* is a source net path.

- **LAYOUT**

A keyword that specifies the *net_path* is a layout net path.

- ***net_path***

A required net path of the form $X_m/X_n/\dots X_o/net_ID$ relative to the current query cell.

Description

Checks whether a layout or source net associated with the *net_path* is valid. A PHDB and XDB are required for **LAYOUT** nets. An XDB is required for **SOURCE** nets.

Acknowledgments

OK: *net_path*, NOK(3), NOK(4), NOK(5)

Related Topics

[NET NAMES](#)

[PLACEMENT BROWSE NETS](#)

[Net Query Commands](#)

Placement Query Commands

This set of commands queries placement information.

Table 5-7. Cell Placement Query Commands

Command	Description
PARSE PATH	Returns cell names corresponding to an instance pathname. For primitive devices, returns their Device type and subtype (if any). Net names that terminate a path are ignored.
PLACEMENT BROWSE DEVICES	Returns layout device names in a cell placement.
PLACEMENT BROWSE NETS	Returns layout net names in a cell placement.
PLACEMENT BROWSE PLACEMENTS	Returns layout placement names in a cell placement.
PLACEMENT BROWSE PORTS	Returns layout port names in a cell placement.
PLACEMENT INFO	Returns the name and extent of a cell placement.
PLACEMENT LAYOUT	Returns the name of a layout placement that corresponds to a source placement.
PLACEMENT LOCATION	Returns the pathnames of placements at a specified location.
PLACEMENT NAMES	Returns layout placement names in a query cell.
PLACEMENT SOURCE	Returns the name of a source placement that corresponds to a layout placement.
PLACEMENT TRANSFORM	Returns transform matrix data for a given placement.
PLACEMENT VALID	Indicates whether a layout or source placement path is valid.
PLACEMENTS OF	Returns layout placements of a cell.

For commands that issue responses, output can be sent to a [RESPONSE FILE](#).

PARSE PATH

Query Server command. Corresponding Tcl shell command: [qs::parse_path](#).

Returns cell names corresponding to instances in an instance pathname. For primitive devices, returns their Device type and subtype (if any). Net names that terminate a path are returned in a separate part of the response.

Usage

PARSE PATH {LAYOUT | SOURCE} *placement_path*

Arguments

- **LAYOUT**
A keyword that specifies the *placement_path* is from the Layout Path design.
- **SOURCE**
A keyword that specifies the *placement_path* is from the Source Path design.
- ***placement_path***
A required pathname of a cell, device, or net placement relative to the query context cell.

Description

Decomposes the *placement_path* into its constituent parts, verifies instances against the cross-reference database (XDB), and returns cell names and port counts for cells having instance references in the path. An XDB is required as part of the SVDB in order for this command to function.

For primitive device instance paths, the Device type and subtype (if any) are returned for the device instance.

Net names at the tail of a pathname or unresolved parts of a path are returned in an Unparsed Path section of the response.

Recall that hierarchical LVS flattens cells that are used in parameterized subcircuit calls. These cells are also flattened in the cross-reference system and in the PARSE PATH command. For this reason, instance pathnames can contain flattened instance paths to parameterized devices rather than a hierarchical structure. For example, this parameterized instance call:

```
X0 1 2 3 ARB xyz=0.3
```

will flatten the cell ARB in the cross-reference database in order to evaluate the parameter xyz=0.3 within the cell. As a result, the cell ARB is not presented in the results from the PARSE PATH command. Rather, its devices are returned, including their flattened names. So a path query for X99/X0/M0 that goes through the instance X0 in the example returns X0/M0 as the instance name for the transistor directly in the context of the cell called at the higher level by X99.

Response

The following response occurs:

```
Parse_Path <precision>
Path:
0 0 <instance_count> <date_stamp>
<instance_0> <cell_name> <port_count>
<instance_1> <cell_name> <port_count>
...
<instance_n> <cell_name> <port_count>
[<device_instance_name> <device_type>[(<subtype>)] <pin_count>]
[Unparsed Path:]
[0 0 1 <date_stamp>]
[<net_name> | <unresolved_path>]
END OF RESPONSE
0 0 0 <date_stamp>
OK.
```

Acknowledgments

OK., NOK(2), NOK(21), ERROR(111), ERROR(166), ERROR(167).

Examples

This shows the output of cell placement path:

```
PARSE PATH SOURCE X0/X0/GND
Parse_Path 1000
Path:
0 0 2 May 27 20:51:09 2020
X0 CellA 3
X0 nand 5
END OF RESPONSE
0 0 0 May 11 16:33:59 2020
OK.
```

An NMOS device instance path response would contain something like this:

```
...
M0 MN(nfet) 4
END OF RESPONSE
...
```

A net path response would look like this:

```
PARSE PATH SOURCE X0/X0/GND
Parse_Path 1000
Path:
0 0 2 May 27 20:51:09 2020
X0 CellA 3
X0 nand 5
Unparsed Path:
0 0 1 May 27 20:51:09 2020
GND
END OF RESPONSE
0 0 0 May 27 20:51:09 2020
OK.
```

Related Topics

[Placement Query Commands](#)

PLACEMENT BROWSE DEVICES

Query Server command. Output of this command is affected by the [BROWSE PSEUDO CELLS](#) and [BROWSE DEVICELESS CELLS](#) commands.

Returns layout device names in a cell placement.

Usage

PLACEMENT BROWSE DEVICES *placement_path*

Arguments

- *placement_path*

A required path of a layout placement relative to the query cell.

Response

- The following response occurs:

```
Placement_Browse_Devices <precision> // "Placement_Browse_Devices"
                                // and precision
    0 0 n <date_stamp>      // 0 0 <number_of_devices> date/time stamp
    <device_name> ...        // <device_name> of <placement_cell>
    ...                      // additional devices
```

Acknowledgments

OK., NOK(18), ERROR(32), ERROR(33)

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

PLACEMENT BROWSE NETS

Query Server command. Output of this command is affected by the [BROWSE PSEUDO CELLS](#) and [BROWSE DEVICELESS CELLS](#) commands.

Returns layout net names in a cell placement.

Usage

PLACEMENT BROWSE NETS *placement_path*

Arguments

- *placement_path*

A required path of a layout placement relative to the query cell.

Acknowledgments

OK., NOK(16), ERROR(32), ERROR(33)

Response

- The following response occurs:

```
Placement_Browse_Nets <precision> // "Placement_Browse_Nets"
                        // and precision
 0 0 n <date_stamp>           // 0 0 <number_of_nets> date/time stamp
 <net_name> ...              // <net_name> of <placement_cell>
   ...                         // additional nets
```

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

PLACEMENT BROWSE PLACEMENTS

Query Server command. Output of this command is affected by the [BROWSE PSEUDO CELLS](#) and [BROWSE DEVICELESS CELLS](#) commands.

Returns layout placement names in a cell placement.

Usage

PLACEMENT BROWSE PLACEMENTS *placement_path*

Arguments

- *placement_path*

A required path of a layout placement relative to the query cell.

Response

- The following response occurs:

```
Placement_Browse_Placements <precision>
// "Placement_Browse_Placements" and precision
<placement_cell>      // following placements are of cell
                        // <placement_cell>
0 0 n <date_stamp>    // 0 0 <number_of_placements> date/time stamp
<placement_name> ... // <placement_name> of <placement_cell>
...                   // additional cells
...                   // additional placements
```

Acknowledgments

OK., NOK(22), ERROR(32), ERROR(33)

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

PLACEMENT BROWSE PORTS

Query Server command. Output of this command is affected by the [BROWSE PSEUDO CELLS](#) and [BROWSE DEVICELESS CELLS](#) commands.

Returns layout port names in a cell placement.

Usage

PLACEMENT BROWSE PORTS *placement_path*

Arguments

- *placement_path*

A required path of a layout placement relative to the query cell.

Response

- The following response occurs:

```
Placement_Browse_Ports <precision>
// "Placement_Browse_Ports" and precision
0 0 n <date_stamp> // 0 0 <number_of_ports> date/time stamp
<port_name> ... // <port_name> of <placement_cell>
... // additional ports
```

Acknowledgments

OK., NOK(10), ERROR(32), ERROR(33)

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

PLACEMENT INFO

Query Server command. Corresponding Tcl shell command: [dfm::get_data -cell_name](#) and [-extent](#) in conjunction with [dfm::get_placements](#).

Returns the name and extent of a cell placement. Coordinates are returned in viewing cell space.

Usage

PLACEMENT INFO *placement_path*

Arguments

- *placement_path*

A required path of a layout placement relative to the query cell. If a device placement name is provided, this has the same effect as using the [DEVICE INFO](#) command.

Response

- The following response occurs:

```
Placement_Info 1000 // "Placement_Info" and precision
cell_name          // placement_path is a placement of cell_name
1 1 0 <date>       // 1 polygon in response
p 1 4              // four sets of vertices marking the extent of
...                // a placement
```

Acknowledgments

OK, NOK(32), ERROR(102), ERROR(104)

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

[Results Transformation Commands](#)

PLACEMENT LAYOUT

Query Server command. Corresponding Tcl shell command: [dfm::get_layout_name -instance](#).

Returns the name of a layout placement that corresponds to a source placement.

Usage

PLACEMENT LAYOUT *source_placement_path*

Arguments

- ***source_placement_path***

A required path of a source placement relative to the query cell.

Description

Returns the name of the corresponding layout placement for the specified ***source_placement_path***. Because device and placement correspondence are maintained in the same data structures, this command is an alias for the [DEVICE LAYOUT](#) command when the ***source_placement_path*** is a device.

Acknowledgments

OK: *layout_placement*, NOK(2), NOK(8), NOK(21), NOK(70)

Related Topics

[PLACEMENT SOURCE](#)

[Placement Query Commands](#)

PLACEMENT LOCATION

Query Server command. Corresponding Tcl shell command: [qs::get_placements_at_location](#).

Returns pathnames of placements at a specified location.

Usage

PLACEMENT LOCATION `{ {cell_name x y ll_x ll_y ur_x ur_y} | {x y [FLAT]} }`

Arguments

- ***cell_name***

A name of a layout cell.

- ***ll_x ll_y***

The coordinates of the lower-left corner of a placement's extent in viewing cell space. The arguments are floating-point numbers in user units.

- ***ur_x ur_y***

The coordinates of the upper-right corner of a placement's extent in viewing cell space. The arguments are floating-point numbers in user units.

- ***x y***

The required coordinates of a placement origin in viewing cell space. The arguments are floating-point numbers in user units.

- **FLAT**

An optional keyword that specifies all placements at any level of the hierarchy below the ***x y*** coordinates are reported. This keyword may only be specified with ***x y***.

Description

Returns the pathname of each matching placement in the layout that corresponds to the given parameters.

A placement is selected by the first form of this command if the placement meets the following criteria:

- It is a placement of cell ***cell_name*** in the layout hierarchy in or below the current query cell.
- The rectangular extent of the placement in the layout database overlaps the rectangular extent given in the command.
- The origin of the placement in viewing cell coordinates is within the current [FILTER DISTANCE](#) from the point specified by ***x y***.

The first form (***cell_name* ...**) of this command is primarily for use with layout editors. It is necessary because, given a placement of a cell in a database, an editor cannot determine the layout path by which that placement is known to the Query Server. This is because the editor

cannot determine the numbering system used in the layout from which the Query Server operates, and the layout may have had additional cells and placements added to the design by hierarchical injection.

The second form (no *cell_name*) of the command is for interactive use or use in a script. It returns all placements of any cell placed under a specified point with coordinates *x y* at the hierarchical level of the current context. If FLAT is used, all placements at any level of the hierarchy below the coordinates are reported.

Cell names can be original cell names or FAUX BIN cells as identified in the LVS run transcript.

The layout pathnames of each placement meeting the previous criteria are returned in the acknowledgment. The placement extents known to the Query Server are based on the shapes used for the placement when the hierarchical database was created from the original layout file. For this reason, they may differ from the extent known to a layout editor. The matching placements are returned in order of decreasing area overlap between the hierarchical database extent and the editor extent. That is, the first placement path returned is the most likely to be the desired placement because it has the greatest overlap with the editor's placement extent specified as *ll_x ll_y ur_x ur_y* in the command.

Response

```
Placement_Location 1000 // "Placement_Location" and precision
placement_name cell_name // placement_name of cell cell_name
...
```

Acknowledgments

OK: *placement_path_1...placement_path_N*, NOK(24)

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

[Results Transformation Commands](#)

PLACEMENT NAMES

Query Server command. Corresponding Tcl shell command: [dfm::get_placements](#) together with [dfm::get_data -placement_name](#).

Returns layout placement names in a query cell.

Usage

PLACEMENT NAMES [FLAT]

Arguments

- FLAT

An optional keyword that specifies all cell placements in the flattened query cell are reported.

Description

Returns a list of all layout placement names that appear in the current query cell. By default, only cell placements appearing immediately in the query cell are reported. The FLAT keyword reports all placements in the internal hierarchy of the query cell. The order in which the placements are listed is arbitrary.

Response

```
Placement_Names <precision> // "Placement_Names" and design precision
Placements:                      // "Placements:"
0 0 <n> <date>                  // n lines of text follow
<placement_name_1>                // first placement name in current context
...
<placement_name_n>                // intermediate placement names in
                                  // current context
                                  // last placement name in current context
```

Acknowledgments

OK., NOK(22), NOK(23), NOK(33), ERROR(101), ERROR(102)

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

PLACEMENT SOURCE

Query Server command. Corresponding Tcl shell command: [dfm::get_source_name -instance](#).

Returns the name of a source placement that corresponds to a layout placement.

Usage

PLACEMENT SOURCE *layout_placement_path*

Arguments

- *layout_placement_path*

A required path of a layout placement relative to the query cell.

Description

Returns the name of the corresponding source placement for the specified *layout_placement_path*. Because device and placement correspondence are maintained in the same data structures, this command is an alias for the [DEVICE SOURCE](#) command when the placement is a device.

Acknowledgments

OK: *source_name*, NOK(2), NOK(7), NOK(21), NOK(71)

Related Topics

[PLACEMENT LAYOUT](#)

[Placement Query Commands](#)

PLACEMENT TRANSFORM

Query Server command.

Returns transform matrix data for a given placement.

Usage

PLACEMENT TRANSFORM *placement_path*

Arguments

- *placement_path*

A required path of a layout placement relative to the query cell.

Description

Returns the transform information (x-offset, y-offset, reflection, rotation) about the placement associated with the *placement_path*. Offsets are in database units. Reflection is 0 for no reflection and 1 for reflection. Rotation is in degrees.

In the Query Server Tcl shell, the equivalent to this command is the following:

- Use `dfm::get_path_context` or `dfm::get_data -path_context` to create a path context navigation object.
- Use `dfm::get_data -extent` to get the bounding box coordinates of the placement from the path context navigation object. The first two elements of the returned list are the X and Y translation offsets of the placement.
- Use `dfm::get_data -xform` to get a transformation object from the path navigation object.
- Use `dfm::get_xform_data` to get reflection and rotation from the transformation object.

Acknowledgments

OK: *x-offset y-offset reflection rotation*, NOK(32)

Related Topics

[Placement Query Commands](#)

[Results Transformation Commands](#)

PLACEMENT VALID

Query Server command. This is the same as the [DEVICE VALID](#) command when the placement is a device. Corresponding Tcl shell command: `qs::placement_valid`.

Indicates whether a layout or source placement path is valid.

Usage

PLACEMENT VALID {LAYOUT | SOURCE} *placement_path*

Arguments

- **LAYOUT**
A keyword that specifies the *placement_path* is for the layout.
- **SOURCE**
A keyword that specifies the *placement_path* is for the source.
- ***placement_path***
A required path of a layout placement relative to the query cell.

Acknowledgments

OK: *placement_path*, NOK(7), NOK(8), NOK(9)

Related Topics

[Placement Query Commands](#)

PLACEMENTS OF

Query Server command. Corresponding Tcl shell command:

`dfm::get_placements -of_cell [-flat]` with `dfm::get_data -path_name`.

Returns layout placements of a cell.

Usage

PLACEMENTS OF *cell_name* [FLAT]

Arguments

- ***cell_name***
A required name of a layout cell.
- FLAT
An optional keyword that specifies the query cell is flattened before reporting placements.

Description

Returns the layout placements of the specified ***cell_name*** in the query cell. By default, only placements that appear at the level of hierarchy of the query cell are reported. The order in which the placements are listed is arbitrary.

Cell names can be original cell names or FAUX BIN cells as identified in the LVS run transcript.

Response

```
Placements_Of <precision> // "Placements_Of" and precision
Placements:           // "Placements:"
 0 0 <n> <date>      // n lines of text follow
 <placement_name_1>   // first placement name rooted in the
                      // current context
 ...
 <placement_name_n>   // intermediate names rooted in the current
                      // context
                      // last placement name rooted in the
                      // current context
```

Acknowledgments

OK., NOK(22), NOK(23), NOK(33), ERROR(101), ERROR(102)

Related Topics

[RESPONSE FILE](#)

[Placement Query Commands](#)

Port Query Commands

This set of commands queries port information.

Table 5-8. Port Query Commands

Command	Description
PORT INFO	Returns information about a port object.
PORT LOCATION	Returns the pathname of a port object nearest the given coordinates.
PORT NAMES	Returns the name of ports in the current query cell.
PORT TEXT MAP	Returns the names of ports and their numeric node numbers.

A corresponding command in the Query Server Tcl shell is [dfm::get_ports](#). The iterator returned by this command can be queried by the [dfm::get_port_data](#) command for information about nets that is similar to many of the standard commands in the preceding table.

For commands that issue responses, output can be sent to a [RESPONSE FILE](#).

POR T INFO

Query Server command. Corresponding Tcl shell command: [dfm::get_port_data](#).

Returns information about a port object.

Usage

POR T INFO *port_name*

Arguments

- *port_name*

A required name of a port in the current query cell.

Description

Returns the name of the net to which the port associated with the *port_name* is attached in the current query cell and the signal direction of the port. For each location associated with the port, a marker square is returned that is centered at that location on the appropriate layer. The size of the marker squares is determined by the current value of the [MARKER SIZE](#) parameter.

This command does not return port information for [Port Layer Polygon](#) ports. However, [POR T TABLE WRITE](#) with [POR T TABLE NAME POLYGON PORTS YES](#) can be used for this purpose.

Response

```
Port_Info <precision> // "Port_Info" and precision
Port: <port_name> // constant "Port:" followed by port name
0 0 2 <date> // two lines of text follow
<net_name> // net attached to port
<signal_direction> // signal direction number: 0 = in, 1 = out,
// 2 = in/out
Layer_name_1 // first layer on which port appears
k k 0 <date> // k shapes on this layer
p 1 4 // first shape on this layer
...
... // four vertices of the first shape
... // remaining k-1 shapes on this layer
... // remaining layers on which port appears.
```

Acknowledgments

OK., NOK(11), ERROR(101), ERROR(102)

Related Topics

[NET BROWSE PORTS](#)

[NET PORTNAMES](#)

[PLACEMENT BROWSE PORTS](#)

[PORT NAMES](#)

[RESPONSE FILE](#)

[Port Query Commands](#)

[Results Transformation Commands](#)

PORT LOCATION

Query Server command. Corresponding Tcl shell command: [dfm::get_port_data -location](#).

Returns the pathname of a port object nearest the given coordinates.

Usage

PORT LOCATION *x* *y*

Arguments

- *x*

A required floating-point number representing an x-coordinate in user units in viewing cell space.

- *y*

A required floating-point number representing a y-coordinate in user units in viewing cell space.

Description

Returns the pathname of the closest port in the current query instance to the *x* *y* coordinates in the viewing cell.

Ports of cells with placements in the query cell are ignored. That is, only ports of the query cell itself are examined. Ports that are not on a layer in the current client's [FILTER LAYERS](#) list are ignored. If two or more ports are equidistant from the location, one is arbitrarily selected and the others are ignored.

Filters

[FILTER LAYERS](#), [FILTER {IN | OUT} LAYERS](#), [FILTER DISTANCE](#)

Acknowledgments

OK: *port_name*, NOK(13), ERROR(116)

Related Topics

[Port Query Commands](#)

PORT NAMES

Query Server command. Corresponding Tcl shell command: [dfm::get_ports -list](#).

Returns the name of ports in the current query cell.

Usage

PORT NAMES [FLAT]

Arguments

- FLAT

An optional keyword that causes all ports in the flattened query cell to be reported.

Description

Returns a list of all port names in the current query cell.

If the FLAT keyword is omitted, only ports at the primary level of the query cell are reported. If the FLAT keyword is used, all ports in the flattened query cell are reported. That is, ports of cells placed in the query cell at any hierarchical level are reported with a query-cell-based pathname leading to the cell instance containing the port. The ports are listed in alphabetical order. Ports with more than one location are listed only once.

Response

```
Port_Names <precision>      // "Port_Names" and precision
Ports:                      // "Ports:"
0 0 <n> <date>            // n lines of text follow
<port_name_1>              // first port name in current context
...
<port_name_n>              // intermediate port names in current context
                           // last port name in current context
```

Acknowledgments

OK., NOK(10), NOK(15), NOK(33), ERROR(101), ERROR(102)

Related Topics

[PORT INFO](#)

[NET PORTNAMES](#)

[NET BROWSE PORTS](#)

[PLACEMENT BROWSE PORTS](#)

[RESPONSE FILE](#)

[Port Query Commands](#)

PORT TEXT MAP

Query Server command. Corresponding Tcl shell command: [dfm::get_ports -spice_name](#).

Returns the names of ports and their numeric node numbers.

Usage

PORT TEXT MAP [INVALID]

Arguments

- INVALID

An optional keyword that causes only those net names that are invalid for SPICE netlisting to be returned.

Description

Returns a list of port names and corresponding LVS-generated net numbers.

Certain text objects are not used for SPICE netlist net names. These text object values are not included in the map.

Response

```
Port_Text_Map <precision> // "Port_Text_Map" and precision
Port Texts:
0 0 <n> <date>           // n lines of text follow
<port_text> <net_number> // port_text is the name of a port connected
                           // to net net_number
```

Acknowledgments

OK., NOK(47)

Related Topics

[NET BROWSE PORTS](#)

[NET PORTNAMES](#)

[PLACEMENT BROWSE PORTS](#)

[RESPONSE FILE](#)

[Port Query Commands](#)

Rule File Query Commands

Rule file query commands return rule file settings that were used for the LVS run.

Third-party tools may require information from the Calibre SVRF rules, such as connectivity between layers, global power and ground net definitions, case sensitivity for layout and source input databases, and so forth. Such commands are documented in this section.

Table 5-9. Rule File Query Commands

Command	Description
LVS SETTINGS REPORT WRITE	Reports LVS rule file settings that are of interest to third-party tools.
RULES CONNECTIVITY	Returns the Connect and Sconnect settings, as well as all connections made by node-preserving operations.
RULES FILE NAME	Returns the name of the rule file used to create the SVDB database.
RULES LAYER TYPES	Returns a listing of rule file layers and their types.
RULES {LAYOUT SOURCE} CASE	Returns the Layout Case or Source Case setting.
RULES LAYOUT NETLIST	Returns the layout netlist name.
RULES LVS COMPARE CASE	Returns the LVS Compare Case setting.
RULES LVS DB LAYER	Returns the LVS DB Layer setting.
RULES LVS DOWNCASE DEVICE	Returns the LVS Downcase Device setting.
RULES LVS {GROUND POWER} NAME	Returns the LVS Ground Name or LVS Power Name setting.
RULES LVS PDSP	Returns the state of LVS Push Devices SEPARATE PROPERTIES YES output.
RULES LVS REPORT	Returns the name of the LVS Report.
RULES LVS SPICE REPLICATE DEVICES	Returns the LVS SPICE Replicate Devices setting.
RULES {LAYOUT SOURCE} PATH	Returns the layout or source pathname.
RULES PRECISION	Returns the Precision statement setting.
RULES {LAYOUT SOURCE} SYSTEM	Returns the Layout System or Source System setting.
RULES SVDB DIRECTORY	Returns the name of the Mask SVDB Directory.
RULES UNIT LENGTH	Returns the Unit Length statement value.

LVS SETTINGS REPORT WRITE

Query Server command. Corresponding Tcl shell command: [qs::write_lvs_settings_report](#).

Reports LVS rule file settings that are of interest to third-party tools.

Usage

LVS SETTINGS REPORT WRITE *filename*

Arguments

- *filename*

A required pathname to an output file.

Description

Returns a report file with a summary of LVS rule file settings to the specified *filename*. The contents are similar to settings that can be obtained from separate Query Server commands in the RULES family.

Most of the report entries are self-explanatory, reflecting either the names of rule file statements or the names of Query Server command or settings.

The LAYER TYPES section of the report contains a list showing the names of layers and their types. Layers can have more than one type. These are the type definitions:

Type	Definition
CONNECT	Connect statement layer.
DEV_AUX	Device operation auxiliary layer.
DEV_PIN	Device operation pin layer.
DEV_SEED	Device operation seed layer.
LVS_DB_LAYER	LVS DB Layer statement argument.
LVS_DB_CONNECTIVITY_LAYER	LVS DB Connectivity Layer statement argument.
PEX	PEX statement layer.
SCONNECT	Sconnect statement layer.
STAMP_BY	Stamp operation BY keyword layer.

The CONNECTIVITY section of the report uses the word CONNECT when the connection is due to a [Connect](#) or [Sconnect](#) statement. If connectivity is established through node-preserving layer derivation, this appears in the IMPLICIT CONNECTIVITY section.

Layer aliases are different names for the same layer. For example, a rule file could have the following code, where alias1 and alias2 are logically equivalent:

```
alias1 = diff AND poly
alias2 = diff AND poly
```

These appear in the LAYER ALIASES section.

If the **MAGNIFY RESULTS** factor is other than 1.0, that factor is multiplied by the rule file Precision, and the result appears in the PRECISION report item.

Response

```
# SVDB: <software version> <time stamp>
# SVDB: LVS Settings Report (lsrf) (File format 1)
# SVDB: Layout Primary <cell>
# SVDB: Rules -0 <filename> <time stamp>
# SVDB: <format> -0 <filename> <time stamp>
# SVDB: SNL -0 <filename>
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
HIERARCHY_SEPARATOR "<character>"
LVS_POWER_NAME <list of power nets>
LVS_GROUND_NAME <list of ground nets>
SOURCE_PATH <filename>
SOURCE_SYSTEM <format>
UNIT_LENGTH <number>
PRECISION <number>
LVS_COMPARE_CASE <NO | YES>
LAYOUT_CASE <NO | YES>
SOURCE_CASE <NO | YES>
LVS_SPICE_REPLICATE_DEVICES <NO | YES>
LVS_DB_LAYER <layer list>
LVS_DB_CONNECTIVITY_LAYER <layer list>
LVS_DOWNCASE_DEVICE <NO | YES>
# LAYER TYPES:
<layer_name> <type>
# CONNECTIVITY:
CONNECT ...
# IMPLICIT CONNECTIVITY:
IMPLICIT_CONNECT <derived_layer> <input_layer>
# LAYER ALIASES
ALIAS <layer1> <layer2> ...
```

Acknowledgments

OK., ERROR(101)

Examples

```
# SVDB: Calibre Version v2017.2_3 Thu Mar 30 12:43:51 PDT 2017
# SVDB: LVS Settings Report (lsrf) (File format 1)
# SVDB: Layout Primary TOPCELL
# SVDB: Rules -0 rules Fri Dec 21 14:11:29 2012
# SVDB: GDSII -0 /sandbox/QUERY_SERVER./inv_nand.oas Fri Dec 21 14:11:29
2012
# SVDB: SNL -0 ./src.spi
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
HIERARCHY_SEPARATOR "/"
LVS_POWER_NAME PWR VDD
LVS_GROUND_NAME GND VSS
SOURCE_PATH ./src.spi
SOURCE_SYSTEM SPICE
UNIT_LENGTH 1e-06
PRECISION 1000
LVS_COMPARE_CASE NO
LAYOUT_CASE NO
SOURCE_CASE NO
LVS_SPICE_REPLICATE_DEVICES NO
LVS_DB_LAYER
LVS_DOWNCASE_DEVICE NO
# LAYER TYPES:
pwell DEV_PIN
pwell SCONNECT
nwell DEV_PIN
nwell SCONNECT
poly CONNECT
psd DEV_PIN
psd CONNECT
pgate DEV_SEED
pgate DEV_PIN
ptap CONNECT
ptap SCONNECT
nsd DEV_PIN
nsd CONNECT
ngate DEV_SEED
ngate DEV_PIN
ntap CONNECT
ntap SCONNECT
metal2 CONNECT
metal1 CONNECT
contact CONNECT
contact SCONNECT
via CONNECT
```

```
# CONNECTIVITY:  
CONNECT metal1 nsd BY contact  
CONNECT metal1 ntap BY contact  
CONNECT metal1 psd BY contact  
CONNECT metal1 ptap BY contact  
CONNECT metal1 poly BY contact  
CONNECT metal1 metal2 BY via  
CONNECT ptap pwell BY contact  
CONNECT ntap nwell BY contact  
# IMPLICIT CONNECTIVITY:  
IMPLICIT_CONNECT pgate poly  
IMPLICIT_CONNECT ngate poly  
# LAYER ALIASES:
```

Related Topics

[RESPONSE FILE](#)

[Rule File Query Commands](#)

RULES CONNECTIVITY

Query Server command. Corresponding Tcl shell command: [qs::rules_connectivity](#).

Returns the Connect and Sconnect settings, as well as all connections made by node-preserving operations.

Usage

RULES CONNECTIVITY

Arguments

None.

Description

Returns the [Connect](#) and [Sconnect](#) statements in the rule file used to create the SVDB database, as well as layer connections made by node-preserving layer operations. Connectivity derived from [Stamp](#) operations is also returned.

Encrypted layers and device layers with no connectivity are not returned.

Connections established through Connect or Sconnect are reported using the word CONNECT. Connections established through layer derivation are reported using the string IMPLICIT_CONNECT.

The response can be written to a [RESPONSE FILE](#).

Response

```
Rules_Connectivity <precision>
Connects:
0 0 <number of connect lines to follow> <time stamp>
CONNECT ...
IMPLICIT_CONNECT ...
...
END OF RESPONSE
0 0 0 <time stamp>
OK.
```

Acknowledgments

OK., ERROR(101)

Examples

Assume the following in the rule file:

```
CONNECT A B BY C
D = B NOT E
F = D AND G
H = I STAMP BY F
```

This is the output:

```
Rules_Connectivity 1000
Connects:
0 0 4 May 3 10:41:46 2012
CONNECT A B BY C
IMPLICIT_CONNECT D B
IMPLICIT_CONNECT F B
IMPLICIT_CONNECT H B
END OF RESPONSE
0 0 0 May 3 10:41:46 2012
OK.
```

Related Topics

[Rule File Query Commands](#)

RULES FILE NAME

Query Server command. Corresponding Tcl shell command: [qs::rules_file_name](#).

Returns the name of the rule file used to create the SVDB database.

Usage

RULES FILE NAME

Arguments

None.

Acknowledgments

OK: *rule_file*, NOK(43)

Related Topics

[Rule File Query Commands](#)

RULES LAYER TYPES

Query Server command. Corresponding Tcl shell command: [qs::rules_layer_types](#).

Returns a listing of rule file layers and their types.

Usage

RULES LAYER TYPES

Arguments

None.

Description

Returns a response containing a list of rule file layers and corresponding types. The type descriptions are given under [LVS SETTINGS REPORT WRITE](#).

Response

```
Rules_Layer_Types <precision>
Layer_Types:
0 0 <count> <time_stamp>
<layer> <type>
...
END OF RESPONSE
0 0 0 <time_stamp>
OK.
```

Acknowledgments

OK., ERROR(101)

Examples

```
RULES LAYER TYPES
Rules_Layer_Types 1000
Layer_Types:
0 0 22 Sep 18 10:02:14 2018
Nwell DEV_PIN
Nwell SCONNECT
Nwell PEX
nmos DEV_SEED
nmos DEV_PIN
nmos CONNECT
...
END OF RESPONSE
0 0 0 Sep 18 10:02:14 2018
OK.
```

Related Topics

[RESPONSE FILE](#)

[Rule File Query Commands](#)

RULES {LAYOUT | SOURCE} CASE

Query Server command. Corresponding Tcl shell commands: [qs::rules_layout_case](#) and [qs::rules_source_case](#).

Returns the Layout Case or Source Case setting in the rule file used to create the SVDB database.

Usage

RULES LAYOUT CASE

RULES SOURCE CASE

Arguments

- **LAYOUT**

A keyword that specifies the [Layout Case](#) setting is returned.

- **SOURCE**

A keyword that specifies the [Source Case](#) setting is returned.

Acknowledgments

OK: NO or YES

Related Topics

[RESPONSE FILE](#)

[Rule File Query Commands](#)

RULES LAYOUT NETLIST

Query Server command. Corresponding Tcl shell command:

`qs::get_run_summary_data -layout_netlist`.

Returns the layout netlist name used to create the SVDB database.

Usage

RULES LAYOUT NETLIST

Arguments

None.

Description

Returns the pathname of the layout netlist used for LVS comparison. The returned pathname may be either the path given in the [Layout Path](#) statement in the rule file (for SPICE-to-SPICE comparison) or the argument to the calibre -spice command line option (for geometric-to-SPICE comparison).

Acknowledgments

OK: *netlist_path_name*, NOK(43)

Related Topics

[Rule File Query Commands](#)

RULES LVS COMPARE CASE

Query Server command. Corresponding Tcl shell command: [qs::rules_lvs_compare_case](#).

Returns the LVS Compare Case keyword setting. The acknowledgment contains space-delimited list of keywords.

Usage

RULES LVS COMPARE CASE

Arguments

None.

Acknowledgments

OK: *keyword_list*

Related Topics

[RULES {LAYOUT | SOURCE} CASE](#)

[Rule File Query Commands](#)

[LVS Compare Case \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES LVS DB LAYER

Query Server command. Corresponding Tcl shell command: [qs::rules_lvs_db_layer](#).

Returns the LVS DB Layer parameters. The acknowledgment contains a space-delimited list of layer names.

Usage

RULES LVS DB LAYER

Arguments

None.

Acknowledgments

OK: *layer_list*

Related Topics

[Rule File Query Commands](#)

[LVS DB Layer \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES LVS DB CONNECTIVITY LAYER

Query Server command. Corresponding Tcl shell command:

[qs::rules_lvs_db_connectivity_layer](#).

Returns the LVS DB Connectivity Layer parameters. The acknowledgment contains a space-delimited list of layer names.

Usage

RULES LVS DB CONNECTIVITY LAYER

Arguments

None.

Acknowledgments

OK: *layer_list*

Related Topics

[Rule File Query Commands](#)

[LVS DB Connectivity Layer \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES LVS DOWNCASE DEVICE

Query Server command. Corresponding Tcl shell command: [qs::rules_lvs_downcase_device](#).

Returns the LVS Downcase Device setting in the rule file used to create the SVDB database.

Usage

RULES LVS DOWNCASE DEVICE

Arguments

None.

Acknowledgments

OK; NO or YES

Related Topics

[Rule File Query Commands](#)

[LVS Downcase Device \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES LVS {GROUND | POWER} NAME

Query Server command. Corresponding Tcl shell commands: [qs::rules_lvs_ground_name](#) and [qs::rules_lvs_power_name](#).

Returns the LVS Ground Name or LVS Power Name setting.

Usage

RULES LVS GROUND NAME

RULES LVS POWER NAME

Arguments

- **GROUND**

A keyword that specifies the LVS Ground Name setting is returned.

- **POWER**

A keyword that specifies the LVS Power Name setting is returned.

Description

Returns the [LVS Ground Name](#) or [LVS Power Name](#) nets in the rule file. If nets are declared as variables, the values of the variables are returned. If the names are declared with wildcards, the wildcards are returned, not the matching names. The acknowledgment contains a space-delimited list of net names.

Acknowledgments

OK: *net_name_list*

Related Topics

[Rule File Query Commands](#)

RULES LVS PDSP

Query Server command.

Returns the state of the LVS Push Devices SEPARATE PROPERTIES YES command output, if specified.

Usage

RULES LVS PDSP

RULES LVS PUSH DEVICES SEPARATE PROPERTIES

Arguments

None.

Acknowledgments

OK: svdb contains PDSP layers, NOK(58)

Related Topics

[Rule File Query Commands](#)

[LVS Push Devices \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES LVS REPORT

Query Server command. Corresponding Tcl shell command:

[`qs::get_run_summary_data -lvs_report`](#).

Returns the filename from the LVS Report specification statement in the rule file used to create the SVDB database.

Usage

RULES LVS REPORT

Arguments

None.

Acknowledgments

OK: *lvs_report_filename*, NOK(27), NOK(43)

Related Topics

[Rule File Query Commands](#)

[LVS Report \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES LVS SPICE REPLICATE DEVICES

Query Server command. Corresponding Tcl shell command:

`qs::rules_lvs_spice_replicate_devices`.

Returns the LVS Spice Replicate Devices setting in the rule file used to create the SVDB database.

Usage

RULES LVS SPICE REPLICATE DEVICES

Arguments

None.

Acknowledgments

OK: NO or YES

Related Topics

[Rule File Query Commands](#)

[LVS Spice Replicate Devices \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES {LAYOUT | SOURCE} PATH

Query Server command. Corresponding Tcl shell commands: [dfm::get_layout_path](#), [dfm::get_source_path](#).

Returns the layout or source design pathname.

Usage

RULES LAYOUT PATH

RULES SOURCE PATH

Arguments

- **LAYOUT**

A keyword that specifies the [Layout Path](#) setting is returned.

- **SOURCE**

A keyword that specifies the [Source Path](#) setting is returned.

Acknowledgments

OK: *pathname*, NOK(30), NOK(31)

Related Topics

[Rule File Query Commands](#)

RULES PRECISION

Query Server command. Corresponding Tcl shell command: [qs::rules_precision](#).

Returns the Precision statement setting in the rule file used to create the SVDB database. If the MAGNIFY RESULTS setting is other than 1.0, that factor is multiplied by the rule file Precision value and returned by RULES PRECISION.

Usage

RULES PRECISION

Arguments

None.

Acknowledgments

OK: *precision*

Related Topics

[Rule File Query Commands](#)

[Precision \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES {LAYOUT | SOURCE} SYSTEM

Query Server command. Corresponding Tcl shell commands: [dfm::get_layout_system](#), [dfm::get_source_system](#).

Returns the Layout System or Source System format.

Usage

RULES LAYOUT SYSTEM

RULES SOURCE SYSTEM

Arguments

- **LAYOUT**

A keyword that specifies the [Layout System](#) setting is returned.

- **SOURCE**

A keyword that specifies the [Source System](#) setting is returned.

Acknowledgments

OK: *system_format*

Related Topics

[Rule File Query Commands](#)

[Layout System \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Source System \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES SVDB DIRECTORY

Query Server command.

Returns the pathname from the Mask SVDB Directory specification statement in the rule file used to create the SVDB database.

Usage

RULES [MASK] SVDB DIRECTORY

Arguments

- **MASK**

An optional keyword used for consistency with the rule file statement name.

Acknowledgments

OK: *svdb_directory_path*, NOK(28)

Related Topics

[Rule File Query Commands](#)

[Mask SVDB Directory \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

RULES UNIT LENGTH

Query Server command. Corresponding Tcl shell command: [qs::rules_unit_length](#).

Returns the Unit Length setting in the rule file used to create the SVDB database. The acknowledgment contains a string representing the unit length in meters in scientific notation.

Usage

RULES UNIT LENGTH

Arguments

None.

Acknowledgments

OK: unit_length

Related Topics

[Rule File Query Commands](#)

[Unit Length \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Hcell Analysis Commands

This set of commands reads netlists, identifies hcells through various manual or automatic means, and produces reports detailing effectiveness of hcells, instantiation statistics, and so forth. Hcell lists can also be output.

Note

 The commands in this section are superseded by the corresponding commands in the Query Server Tcl shell, which are documented in the chapter “[Hcell and Hierarchical Analysis](#)” on page 211.

The query netlist commands may be used in any order. For example, a new hcell file can be read in without re-reading the netlists to see the effects of adding a particular hcell to the hcell file. However, the contents of the output reports are dependent on the prevailing conditions at the time the reports are generated.

See “[Hcell List Management Using Standard Commands](#)” for detailed usage information regarding the commands in this section.

Table 5-10. Netlist and Hierarchy Analysis Commands

Command	Description
NETLIST ADD MATCHING HCELLS	Adds cells matched by NETLIST AUTOMATCH or NETLIST PLACEMENTMATCH to the system hcell list.
NETLIST AUTOMATCH	Specifies that candidate hcells are matched by name.
NETLIST CLEAR HCELL	Removes a cell pair from the current hcell list maintained by the system.
NETLIST CLEAR HCELLS	Removes all hcell pairs from the current hcell list.
NETLIST EVALUATE CURRENT HCELLS	Specifies whether hcells in the current hcell list undergo evaluation.
NETLIST EVALUATION THRESHOLD	Specifies threshold values for evaluating hcell effectiveness.
NETLIST EXPAND UNBALANCED	Specifies how to handle unbalanced hcell expansion.
NETLIST HCELL	Adds an hcell pair to the system hcell list.
NETLIST HCELLS	Reads in an existing hcell list.
NETLIST PLACEMENTMATCH	Specifies that candidate hcells are matched by placement count.

Table 5-10. Netlist and Hierarchy Analysis Commands (cont.)

Command	Description
NETLIST READ	Reads the rule file for netlist information.
NETLIST REPORT HCELLS	Writes the current hcell list.
NETLIST REPORT HIERARCHY	Writes a design hierarchy report.
NETLIST SELECT HCELLS	Sets the current hcell list using hcell effectiveness evaluation thresholds and generates an evaluation report.
NETLIST STATUS	Returns the current command settings for hcell analysis.

Hcell List Management Using Standard Commands

Hcell lists specify cells that correspond between layout and source netlists. Judicious selection of hcells can make Calibre nmLVS-H comparison run much faster than otherwise is possible. The Query Server can assist in constructing hcell lists.

Note

 This section discusses using the hcell analysis and reporting features that are a part of the standard Query Server. These procedures are superseded by what appears in the section “[Hcell and Hierarchical Analysis](#)” on page 211.

Initially, the rule file is typically read using the [NETLIST READ](#) command. You can then specify hcells explicitly or automatically.

You can see a hierarchy report by using the [NETLIST REPORT HIERARCHY](#) command. This report can assist in choosing hcell candidates manually.

You can add cells to the current hcell list maintained by the system in any of the following ways:

- The [Hcell](#) rule file statement adds hcells through the rule file. These hcells become part of the current hcell list maintained by the system when NETLIST READ is executed.
- [NETLIST HCELLS](#) adds hcells from an hcell file.
- [NETLIST HCELL](#) adds hcells manually.
- [NETLIST AUTOMATCH](#) or [NETLIST PLACEMENTMATCH](#) identify matching hcells automatically. These cells are not added to the current hcell list until either of the next two commands is used.
- [NETLIST ADD MATCHING HCELLS](#) or [NETLIST SELECT HCELLS](#) adds cells identified with the automatch or placement matching heuristics.

Hcells that have been explicitly identified are part of the current hcell list maintained by the system.

You can remove cells from the current hcell list with the following commands:

- [NETLIST CLEAR HCELL](#) removes an individual hcell pair.
- [NETLIST CLEAR HCELLS](#) removes all hcells.
- The [LVS Exclude Hcell](#) specification statement in the rule file prevents cells from serving as hcells.

You can examine the current hcell list and status with the following commands:

- [NETLIST REPORT HCELLS](#) reports the current hcells.
- [NETLIST STATUS](#) reports current settings of various NETLIST commands.

You can control how the Query Server handles unbalanced hcell using the [NETLIST EXPAND UNBALANCED](#) command.

The [NETLIST SELECT HCELLS](#) selects hcells for the designs that have been read in. This command iteratively identifies hcells based upon their potential to reduce the THIC, given the existing set of hcells (including those chosen on previous iterations). By default, the savings threshold is set so that when the potential remaining THIC savings falls below 30 percent, further hcells will not be selected unless they meet the instance count criterion.

In addition, hcells whose instance count (at the top level of the cell) represents at least one percent of the total flat instance count of the design are selected by default, regardless of their contribution to THIC savings. This is a selectable threshold.

You can configure execution of the NETLIST SELECT HCELLS command through the following commands:

- The [NETLIST EVALUATE CURRENT HCELLS](#) command includes or excludes the current hcell list in the evaluation. By default, current hcells are evaluated and are removed from the current hcell list if they do not meet the evaluation thresholds. When NETLIST EVALUATE CURRENT HCELLS NO is set, current hcells are always kept on the current hcell list, regardless of whether they actually meet the evaluation thresholds.
- The [NETLIST EVALUATION THRESHOLD](#) command configures the cutoff thresholds for additional hcells.

The NETLIST SELECT HCELLS command evaluates hcells and creates a new current hcell list. The thresholds ensure that effective hcells are selected.

See “[Hcell Cycles and Conflicts](#)” on page 215 for related information.

NETLIST ADD MATCHING HCELLS

Query Server command. Corresponding Tcl shell command: [hcells::add_matching_hcells](#).

Adds cells identified with the NETLIST AUTOMATCH or NETLIST PLACEMENTMATCH command heuristics to the system hcell list.

Usage

NETLIST ADD MATCHING HCELLS

Arguments

None.

Acknowledgments

OK., ERROR(134)

Description

Adds matched hcells to the current hcell list from either of the automatic matching commands.

If [NETLIST AUTOMATCH STRICT](#) has been specified and only one netlist has been read (a single design), then ERROR(207) is given because two input designs are required for strict automatching.

Examples

This sequence of commands shows how to add and verify which cells are matched using the NETLIST AUTOMATCH and [NETLIST PLACEMENTMATCH](#) commands. This assumes a geometric layout and a SPICE source.

```
netlist automatch strict
  OK.
netlist placementmatch on
  OK.
netlist read rules
  <netlists read>
  OK.
netlist add matching hcells
  OK.
netlist report hcells
<layout> <source>
...
  OK.
```

Related Topics

[NETLIST READ](#)

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST AUTOMATCH

Query Server command. Corresponding Tcl shell command: [hcells::automatch](#).

Specifies that candidate hcells are matched by name.

Usage

NETLIST AUTOMATCH {OFF | ON | STRICT}

Arguments

- **OFF**

A keyword that specifies automatic matching by cell name is disabled. This is the default.

- **ON**

A keyword that specifies automatic matching by cell name is enabled.

- **STRICT**

A keyword that specifies automatic matching by cell name is enabled, but only of the matched cells have the same number of ports in layout and source.

Description

Controls whether cells with the same name in layout and source should be considered hcells (similar to the LVS -automatch command line option) when the [NETLIST ADD MATCHING HCELLS](#) or [NETLIST SELECT HCELLS](#) command is issued. When [NETLIST READ](#) is used, both input designs must be SPICE for NETLIST SELECT HCELLS to evaluate hcell efficiency.

The **ON** setting should only be used when it is certain that source subcircuits match corresponding layout cells of the same name.

The **STRICT** keyword may not be used with NETLIST SELECT HCELLS when the layout is geometric (two netlists are required as inputs), and it may not be used with NETLIST ADD MATCHING HCELLS when only one netlist is read (a single design).

When the **STRICT** keyword is used prior to reading the layout and source designs, the Layout System is not SPICE, and NETLIST ADD MATCHING HCELLS has added any matched cells to the current hcell list, then the hcell pairs written by [NETLIST REPORT HCELLS](#) receive a **STRICT** keyword appended to their line. These cells are treated as hcells during hierarchical LVS comparison if their port counts match.

When automatic matching is enabled, cell names that are the same are paired as hcells. Injected cell names having the prefix **ICV_** are reserved for Calibre internal use and are not used as hcells.

Acknowledgments

OK., ERROR(134)

Related Topics

[NETLIST PLACEMENTMATCH](#)

[Hcell Analysis Commands](#)

[Historical Hcell Reporting and Selection Scripts](#)

NETLIST CLEAR HCELL

Query Server command. Corresponding Tcl shell command: [hcells::clear_hcell](#).

Removes a cell pair from the current hcell list maintained by the system.

Usage

NETLIST CLEAR HCELL *layout_cell source_cell*

Arguments

- *layout_cell*
A required layout cell name corresponding to the *source_cell*.
- *source_cell*
A required source cell name.

Acknowledgments

OK., NOK(49)

Examples

This example shows the clearing of CellA from the current hcell list.

```
netlist read rules
<rules read and cells evaluated>
    OK.
netlist report hcells
nand    nand
inv     inv
CellA   CellA
CellB   CellB
    OK.
netlist clear hcell CellA CellA
    OK.
netlist report hcells
nand    nand
inv     inv
CellB   CellB
    OK.
```

Related Topics

[NETLIST CLEAR HCELLS](#)

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST CLEAR HCELLS

Query Server command. Corresponding Tcl shell command: [hcells::clear_hcells](#).

Removes all hcell pairs from the current hcell list.

Usage

NETLIST CLEAR HCELLS

Arguments

None.

Description

Removes all matched cell pairs from the hcell list maintained by the system. This command is typically followed by a [NETLIST READ](#), [NETLIST HCELLS](#), or [NETLIST HCELL](#) command to add hcells to the current hcell list maintained by the system.

Acknowledgments

OK., ERROR(134)

Related Topics

[NETLIST CLEAR HCELL](#)

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST EVALUATE CURRENT HCELLS

Query Server command. Corresponding Tcl shell command: [hcells::evaluate_current_hcells](#).

Specifies whether hcells in the current hcell list undergo evaluation.

Usage

NETLIST EVALUATE CURRENT HCELLS {YES | NO}

Arguments

- **YES**

A keyword that specifies hcells are evaluated. This is the default.

- **NO**

A keyword that specifies hcells are not evaluated.

Description

Controls whether cells in the current hcell list maintained by the system undergo evaluation by [NETLIST SELECT HCELLS](#).

When the setting is **NO**, all cells in the current list remain in the list regardless of the THIC (total hierarchical instance count) benefit they provide.

When the setting is **YES** (the default), all cells in the current list are evaluated along with cells selected by automatic means ([NETLIST AUTOMATCH](#) and [NETLIST PLACEMENTMATCH](#)). Cells in the current hcell list are removed from the list if they do not meet the evaluation thresholds configured through the [NETLIST EVALUATION THRESHOLD](#) command.

Acknowledgments

OK., NOTE: No explicit hcells are currently available for evaluation.

Related Topics

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST EVALUATION THRESHOLD

Query Server command. Corresponding Tcl shell command: [hcells::select -threshold](#).

Specifies threshold values for evaluating hcell effectiveness.

Usage

NETLIST EVALUATION THRESHOLD *THIC_threshold* [*flat_threshold*]

Arguments

- ***THIC_threshold***

A required positive integer that represents the cutoff point for total hierarchical instance count (THIC) as a percent. The default ***THIC_threshold*** is 30.

- ***flat_threshold***

An optional positive integer that represents the proportion of a cell's internal instance count (top level of the cell) to the entire design's flat instance count as a percent. The default is 1.

Description

Configures the cutoff thresholds for hcell effectiveness evaluation. Evaluation only occurs when [NETLIST EVALUATE CURRENT HCELLS YES](#) is specified, which is the default, and thresholds are applied as selection criteria only when [NETLIST SELECT HCELLS](#) is used.

By default, cells that represent less than a 30 percent reduction in total hierarchical instance count (THIC) and that contain less than 1 percent of the total flat instance count of the design are not selected. If a cell satisfies the *flat_threshold*, it is selected as an hcell regardless of THIC savings. The *flat_threshold* assists in permitting large hcells to be selected even if they do not contribute toward meeting the ***THIC_threshold***.

Increasing either *THIC_threshold* or *flat_threshold* tends to reduce the number of selected hcells. See the description of Potential Remaining Savings under “[Hcell Evaluation Report Format](#)” on page 240.

Cells that meet the conditions of the command are added to the current hcell list by NETLIST SELECT HCELLS.

See [Hcell List Management Using Standard Commands](#) for a complete discussion of the threshold values.

Acknowledgments

OK., ERROR(149)

Related Topics

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST EXPAND UNBALANCED

Query Server command. Corresponding Tcl shell command: [hcells::expand_unbalanced](#).

Specifies how to handle unbalanced hcell expansion.

Usage

NETLIST EXPAND UNBALANCED {RULES | YES | NO}

Arguments

- **RULES**

A keyword that specifies the [LVS Expand Unbalanced Cells](#) specification statement in the rule file controls unbalanced hcell expansion. The default of that statement is YES. This is the default.

- **YES**

A keyword that specifies unbalanced hcells are expanded, regardless of the LVS Expand Unbalanced Cells setting.

- **NO**

A keyword that specifies unbalanced hcells are not expanded, regardless of the LVS Expand Unbalanced Cells setting.

Description

Controls unbalanced cell expansion during analysis of a netlist using the [NETLIST READ](#) command.

See “[Unbalanced Hcell Reporting](#)” on page 246 for details of using this command.

Acknowledgments

OK., ERROR(134)

Related Topics

[Hcell Analysis Commands](#)

NETLIST HCELL

Query Server command. Corresponding Tcl shell command: [hcells::hcell](#).

Adds an hcell pair to the current hcell list maintained by the system.

Usage

NETLIST HCELL *layout_cell source_cell*

Arguments

- *layout_cell*
A required layout cell name corresponding to the *source_cell*.
- *source_cell*
A required source cell name.

Acknowledgments

OK., ERROR(134)

Related Topics

[NETLIST HCELLS](#)

[NETLIST READ](#)

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST HCELLS

Query Server command. Corresponding Tcl shell command: [hcells::hcells](#).

Reads in an existing hcell list.

Usage

NETLIST HCELLS *filename*

Arguments

- *filename*

A required pathname to a file containing an hcell list.

Description

Causes the *filename* to be used as an hcell file (similar to the LVS -hcell command line option).
Removes any existing hcells from the hcells list maintained by the system.

Acknowledgments

OK., ERROR(134), ERROR(146)

Related Topics

[NETLIST HCELL](#)

[NETLIST READ](#)

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST PLACEMENTMATCH

Query Server command. Corresponding Tcl shell command: [hcells::placementmatch](#).

Specifies that candidate hcells are matched by placement count.

Usage

NETLIST PLACEMENTMATCH {OFF | ON | LOOSE}

Arguments

- **OFF**

A keyword that specifies automatic matching by placement count and pin count is disabled.
This is the default.

- **ON**

A keyword that specifies automatic matching by placement count and pin count is enabled.

- **LOOSE**

A keyword that specifies automatic matching by placement count is enabled, but pin count is ignored.

Description

Controls whether the [NETLIST ADD MATCHING HCELLS](#) or [NETLIST SELECT HCELLS](#) command adds hcells that are matched using the placement matching heuristic to the current hcell list maintained by the system. Cells can be matched as hcells based on whether the number of times the cells appear in the layout and source is identical and the number is unique.

If **ON** is specified, cells are matched as hcells based upon placement counts, and pin counts must also match in the layout and source.

Cells that match (or could match) using placement matching have their names preceded by a pound (#) character in the netlist evaluation report.

See “[Cell Matches Using Placement Matching](#)” on page 246 for additional details.

Acknowledgments

OK., ERROR(134)

Related Topics

[Hcell Analysis Commands](#)

[Historical Hcell Reporting and Selection Scripts](#)

NETLIST READ

Query Server command. Corresponding Tcl shell command: [hcells::read](#).

Reads the rule file for netlist information.

Usage

NETLIST READ *rule_file* [LAYOUT | SOURCE]

Arguments

- ***rule_file***
A required pathname to the LVS rule file.
- **LAYOUT**
An optional keyword that causes only the layout netlist to be analyzed. If the [Layout System](#) in the rule file is not SPICE, then the LAYOUT option is not useful.
- **SOURCE**
An optional keyword that causes only the source netlist to be analyzed.

Description

Causes the source and layout netlists specified in the *rule_file* to be read into memory for analysis.

The SOURCE and LAYOUT keywords cause the command to read only that netlist and to use it as both source and layout. Using these options allows you to analyze the effectiveness of one set of hcells from either the layout or source set.

When neither SOURCE nor LAYOUT is specified, this command reads the source netlist and the layout netlist if Layout System SPICE is used (if not, layout cells are not evaluated for reporting purposes). Hcell analysis finds corresponding cells based upon the matching conditions specified: automatch, placementmatch, or explicit cell matching list.

Acknowledgments

OK., ERROR(134), ERROR(135), ERROR(145)

Examples

This sequence of commands is useful when the layout and source are both SPICE.

NETLIST AUTOMATCH ON	(match hcells by names)
NETLIST PLACEMENTMATCH ON	(match hcells by placement count)
NETLIST READ rules	(read the rule file)
NETLIST ADD MATCHING HCELLS	(add matching cells)
RESPONSE FILE cells.report	(send the report to this file)
NETLIST SELECT HCELLS	(write the effective hcell report)
RESPONSE FILE hcells	(send an initial hcell list to this file)
NETLIST REPORT HCELLS	(initial hcell list)

The *hcells* file contains the hcells the Query Server chooses based upon default selection criteria. This file can be tested in an LVS run for performance and further modification. The *cells.report* follows the “[Hcell Evaluation Report Format](#)” on page 240.

Related Topics

[NETLIST ADD MATCHING HCELLS](#)

[NETLIST HCELL](#)

[NETLIST HCELLS](#)

[Hcell Analysis Commands](#)

[Historical Hcell Reporting and Selection Scripts](#)

NETLIST REPORT HCELLS

Query Server command. Corresponding Tcl shell command: [hcells::print_hcells](#).

Writes the current hcell list maintained by the system.

Usage

NETLIST REPORT HCELLS

Arguments

None.

Response

- The following response is given for each hcell pair:

```
<layout_cell> <source_cell> // matched layout and source hcells
```

The [RESPONSE FILE](#) command can be used to print the hcell list to a file.

Acknowledgments

OK.

Examples

This sequence of commands is used to generate an hcell list that has been built up from previous commands:

NETLIST SELECT HCELLS	(set the current hcell list)
RESPONSE FILE hcells	(send an hcell list to this file)
NETLIST REPORT HCELLS	(write hcell list)

Related Topics

[NETLIST REPORT HIERARCHY](#)

[NETLIST SELECT HCELLS](#)

[Hcell Analysis Commands](#)

[Historical Hcell Reporting and Selection Scripts](#)

NETLIST REPORT HIERARCHY

Query Server command. Corresponding Tcl shell command: [hcells::print_hierarchy](#).

Writes a design hierarchy report.

Usage

NETLIST REPORT HIERARCHY {LAYOUT | SOURCE}

Arguments

- **LAYOUT**

A keyword that specifies a layout netlist hierarchy report is written. Using **LAYOUT** assumes [Layout System SPICE](#) in the rules.

- **SOURCE**

A keyword that specifies a source netlist hierarchy report is written.

Description

Creates an hcell analysis and hierarchy tree report for the source or layout netlist. This report is useful for analyzing the cells that are hcell candidates. See “[Hcell Analysis and Hierarchy Tree Report Format](#)” on page 242 for details about the report.

For any hcell candidates identified by [NETLIST AUTOMATCH](#) or [NETLIST PLACEMENTMATCH](#), these cells must first be added to the current hcell list before they are considered by NETLIST REPORT HIERARCHY. This is done by using [NETLIST ADD MATCHING HCELLS](#) or [NETLIST SELECT HCELLS](#).

The [RESPONSE FILE](#) command can be used to print the report to a file. Since the report can be long for large designs, using a response file is recommended.

Acknowledgments

OK., ERROR(136), ERROR(143), ERROR(144)

Related Topics

[NETLIST REPORT HCELLS](#)

[NETLIST SELECT HCELLS](#)

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST SELECT HCELLS

Query Server command. Corresponding Tcl shell command: [hcells::select](#).

Sets the current hcell list and generates an evaluation report.

Usage

NETLIST SELECT HCELLS

Arguments

None.

Description

Generates an Hcell Evaluation Report based on layout and source contributions, and sets the current hcell list maintained by the system.

Selection threshold criteria can be configured through the [NETLIST EVALUATION THRESHOLD](#) command. The [RESPONSE FILE](#) can be used to print the report to a file.

Candidate cell pairs identified by [NETLIST AUTOMATCH](#) and [NETLIST PLACEMENTMATCH](#) are added by NETLIST SELECT HCELLS to the current hcell list.

If NETLIST AUTOMATCH STRICT is active, [NETLIST READ LAYOUT](#) or SOURCE is used or the Layout System is not SPICE, then ERROR(206) is given because hcell selection depends upon strict automatching occurring first. This can only be performed with two input netlists.

See “[Historical Hcell Reporting and Selection Scripts](#)” on page 429 for a selection of scripts that show how commands are used together.

Response

See “[Hcell Evaluation Report Format](#)” on page 240.

Acknowledgments

OK., ERROR(134), ERROR(136), ERROR(148), ERROR(206)

Related Topics

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

NETLIST STATUS

Query Server command. Corresponding Tcl shell command: [hcells::status](#).

Lists the current status of netlist analysis commands and whether the netlists and rule file have been read.

Usage

NETLIST STATUS

Arguments

None.

Response

- The following response occurs:

netlist automatch:	<on off>
netlist placementmatch:	<on off>
netlist THIC threshold:	<percent>
netlist Flat instance count threshold:	<percent>
netlist evaluate current hcells:	<yes no>
netlist hcell file:	<file null>
netlist expand unbalanced	<from rules no yes>
[netlist read from rule file:	<rule_file_name>
netlist Netlists not read.]	

Acknowledgments

OK.

Related Topics

[RESPONSE FILE](#)

[Hcell Analysis Commands](#)

[Hcell List Management Using Standard Commands](#)

Historical Hcell Reporting and Selection Scripts

Following are examples using standard Query Server commands for hcell list generation and reporting. The scripts can either be used interactively in the Query Server shell or used in a runtime script.

The rule file is assumed to have [Layout System](#) SPICE. (If this is not the case, then the NETLIST HCELLS command must be used to specify hcell names and evaluation occurs for the source netlist only.) Here is a sequence of commands for hcell evaluation:

```
% calibre -query
...
NETLIST AUTOMATCH STRICT          (match hcells by name and pins)
NETLIST READ rules                 (read the rule file)
NETLIST ADD MATCHING HCELLS       (add matching cells to current list)
RESPONSE FILE cells.eval          (send the report to this file)
NETLIST SELECT HCELLS             (create hcell report and add
                                  new cells to hcell list)
RESPONSE FILE cells               (send cell names to this file)
NETLIST REPORT HCELLS             (write the hcell list)
```

This writes a *cells.eval* report with cell statistics. The report is sorted by cell count and shows memory savings percentages for hcell pairs in layout and source. See “[Hcell Evaluation Report Format](#)” on page 240 for an example report. Using the report, you can choose to add ([NETLIST HCELL](#)) or remove hcells ([NETLIST CLEAR HCELL](#)) from the current list.

The file *cells* contains a list of hcells that meet the default evaluation criteria. This can be used as an LVS -hcell option list. This list may not be optimal, so you should test it thoroughly, checking memory use and LVS results for validity.

This shows an interactive sequence of commands for checking the hcell candidates from the automatic matching algorithms:

```
% calibre -query
...
NETLIST READ rules                 (read the rule file netlists)
NETLIST CLEAR HCELLS              (clear the current hcell list)
NETLIST AUTOMATCH ON              (match hcells by names)
NETLIST PLACEMENTMATCH ON         (match hcells by placement count)
NETLIST SELECT HCELLS             (select hcells in the current list
                                  and return a report to the shell)
NETLIST EVALUATE CURRENT HCELLS NO (now discount hcell efficiency)
NETLIST SELECT HCELLS             (update the current hcell list)
NETLIST REPORT HCELLS             (show the hcell list again)
```

In this case, the reports and cell lists are returned to the shell. The final hcell list shows all cells that were matched automatically, not taking into account their efficiency as hcells. The RESPONSE FILE command can be used to save the outputs to separate files for comparison.

Hierarchy Reporting

The NETLIST REPORT HIERARCHY command produces a report of cells found in the source or layout hierarchy. Cells that would make good hcells appear in order of priority, from top to bottom, in the table found in the report.

When trying to identify potential hcells, generate reports for both the source and the layout, if possible. (If both designs are not available, create a rule file that compares the available design with itself.) For example, given a rule file named *rules* with the following lines:

```
LAYOUT PATH "lay.net"
LAYOUT PRIMARY "chip"
LAYOUT SYSTEM SPICE

SOURCE PATH "src.net"
SOURCE PRIMARY "chip"
SOURCE SYSTEM SPICE
```

You can start the Query Server:

```
% calibre -query
```

and execute the following commands interactively:

```
NETLIST READ rules
NETLIST CLEAR HCELLS
RESPONSE FILE lay.rep
NETLIST REPORT HIERARCHY layout
RESPONSE FILE src.rep
NETLIST REPORT HIERARCHY source
```

Now select an hcell pair from the reports, add it using the NETLIST HCELL command, and regenerate the reports:

```
NETLIST HCELL layout_cell source_cell
NETLIST REPORT HIERARCHY layout
RESPONSE FILE lay.new.rep
NETLIST REPORT HIERARCHY layout
RESPONSE FILE src.new.rep
NETLIST REPORT HIERARCHY source
```

This sequence identifies the next highest potential hcell candidates taking into account the new hcell. (Specifying additional hcells lessens the potential importance of other cells that share hierarchy above and below that hcell.) This process can be followed iteratively to build an hcell list. The overall effectiveness of hcells can be determined using HCELLS SELECT when the hcell list is added to the current hcell list.

See “[Hcell Analysis and Hierarchy Tree Report Format](#)” on page 242 for details.

Generating Hcell and Xcell Lists

This procedure is for parasitic extraction flows that use xcells. This shows the historical hcell list generation commands.

Prerequisites

- A Calibre Query Server license.

- A Mask SVDB Directory generated from a connectivity extraction run (calibre -spice), where the QUERY keyword has been used.
- An LVS connectivity extraction rule file.
- Extracted layout SPICE netlist.

Procedure

1. Copy the connectivity extraction rule file to a new name, as follows:

```
cp extract.rules hcell_gen.rules
```

2. Open the *hcell_gen.rules* file and ensure these settings are active:

```
LAYOUT SYSTEM SPICE
LAYOUT PATH layout.sp // extracted SPICE netlist
```

3. Enter the following commands in a text file called *query_script*:

```
NETLIST AUTOMATCH STRICT
NETLIST READ hcell_gen.rules
RESPONSE FILE cells.eval
NETLIST SELECT HCELLS
RESPONSE FILE new_hcells
NETLIST REPORT HCELLS
QUIT
```

4. Execute this command in a C shell:

```
% calibre -query < query_script >&! extract_hcells.log
```

The file *new_hcells* contains the hcell file.

Results

You can review the *extract_hcells.log* file to view the HCELL EVALUATION REPORT section.

The *new_hcells* file identifies xcells for Calibre parasitic extraction flows. A subset of the list can be used for the PDB and FMT stages, if desired.

Related Topics

[Hcell Evaluation Report Format](#)

Chapter 6

Calibre Connectivity Interface

The Calibre Connectivity Interface (CCI) provides downstream silicon modeling tools access to extracted layout connectivity information from a Calibre LVS run. This information includes layout shapes, nets, devices, rule file configuration details, and corresponding schematic or source names. Third-party parasitic extractors are an example of potential clients for this interface.

All data required for the CCI interface is enabled with the [Mask SVDB Directory ...](#) CCI rule file statement. The file formats produced are standard Query Server formats that do not conform to the usual DRC results format.

There are two CCI interface modes—Tcl shell and standard. The former is covered first in this chapter and is preferred.

A basic script for generating all CCI files needed for downstream flows is discussed under “[Generating CCI Output Files](#)” on page 480.

The CCI commands require a Calibre Connectivity Interface license. Refer to the [Calibre Administrator’s Guide](#) for licensing information.

Tcl Shell CCI Command Reference	434
Rule File Query Commands	462
ERC TVF Commands	464
Property Definitions and Netlist Behavior for CCI Flows	477
Generating CCI Output Files	480
CCI Generated Files Reference	485
Customized Files for Pushdown Separated Properties (PDSP) Flow	514
Generating a Flat Netlist for a Specific Device	518
Viewing Connect and Device Layers	519
Standard Interface CCI Command Reference	520

Tcl Shell CCI Command Reference

Calibre Connectivity Interface commands are implemented in the Query Server Tcl shell and in Calibre YieldServer.

The following commands are of interest in a CCI context. Some require a CCI license, which is mentioned on a commands reference page when applicable.

Commands in the dfm:: scope are documented in the *Calibre YieldServer Reference Manual*.

Table 6-1. Tcl Shell CCI Commands

Command	Description
dfm::create_filter	Creates a filter object used by other CCI commands.
dfm::list_layout_netlist_options	Returns a Tcl list of dfm::set_layout_netlist option settings.
dfm::set_layout_netlist_options	Configures the dfm::write_spice_netlist command for writing a layout netlist from the PHDB.
dfm::set_netlist_options	Configures the output of dfm::write_spice_netlist from a netlist read by dfm::read_netlist. Controls how comment-coded properties are written.
dfm::write_ixf	Writes an instance cross-reference (IXF) file.
dfm::write_nxf	Writes a net cross-reference (NXF) file.
dfm::write_lph	Writes a layout placement hierarchy (LPH) file.
dfm::write_sph	Writes a source placement hierarchy (SPH) file.
dfm::write_reduction_data	Writes a file containing information about transformation reductions occurring in LVS comparison.
dfm::write_spice_netlist	Writes a SPICE netlist. If used without a dfm::read_netlist handle, writes a SPICE netlist similar to calibre -spice based upon the PHDB.
dfm::xref_xname	Configures the subcircuit call format of cross-reference file commands.
qs::agf_map	Returns the layer map scheme of annotated geometry format output.
qs::annotated_device_layer_map	Returns a Tcl list of layers that are used for annotations on particular devices.
qs::device_table	Returns a table of information about all devices in the layout design.
qs::device_templates_used	Returns lists of the instances formed during device recognition, along with associated statistics.
qs::log_cci_commands	Provides a trace log for certain commands of interest in CCI runs.

Table 6-1. Tcl Shell CCI Commands (cont.)

Command	Description
<code>qs::port_table</code>	Returns a listing of information about cell ports in the layout design.
<code>qs::set_agf_options</code>	Configures annotated geometry format output settings for the <code>qs::write_agf</code> command.
<code>qs::write_agf</code>	Writes an annotated geometry format file.
<code>qs::write_cell_extents</code>	Writes a cell extents file.
<code>qs::write_layout_netlist_names</code>	Writes a Layout Netlist Names correspondence file.
<code>qs::write_lvs_settings_report</code>	Returns information from the LVS rules used to generate the SVDB database.
<code>qs::write_separated_properties</code>	Writes push-down separated properties to a file.

Conflicting Annotated Device Layers

Conflicting annotated device layers exist when there is more than one annotated layer used for a single device. This can be an important consideration when using CCI device annotation commands.

Here is a simple example of conflicting layers:

```
L1 = DEVICE LAYER MP(P1) ANNOTATE dfm_1
L2 = DEVICE LAYER MP(P1) ANNOTATE dfm_2

LVS ANNOTATE DEVICES L1
LVS ANNOTATE DEVICES L2
```

L1 and L2 are conflicting layers because they both represent the same device. Mixing annotated device layers with devices of the same type also causes layer conflicts. Here is an example:

```
L1 = DEVICE LAYER MP(P1) ANNOTATE dfm_1
L2 = DEVICE LAYER MP ANNOTATE dfm_2

LVS ANNOTATE DEVICES L1
LVS ANNOTATE DEVICES L2
```

L1 and L2 are conflicting layers because they both represent the same device type.

Only one layer from a set of conflicting layers can be a part of the current set of annotated layers at any given time. When the CCI client starts, if there is a set of conflicting layers, the first layer of the conflicting set that appears in the [LVS Annotate Devices](#) rule file statements is used, and the remainder of the conflicting layers are ignored. You can select which conflicting layer to filter by using the `dfm::create_filter` command in the Tcl shell and the FILTER IN ANNOTATED LAYERS command in the standard interface.

qs::annotated_device_layer_map

CCI command. Corresponding standard command: [ANNOTATED DEVICE LAYER MAP](#). A Calibre Connectivity Interface license is required for this command. Mask SVDB Directory CCI must be specified in the rules used to generate the loaded database.

Returns a Tcl list of layers that are used for annotations on particular devices.

Usage

qs::annotated_device_layer_map [-write *filename*] [-help]

Arguments

- **-write *filename***

An optional argument set that specifies to direct output to the specified filename. When this option is used, the output format is similar to the response generated by ANNOTATED DEVICE LAYER MAP.

- **-help**

An optional argument that shows the command usage.

Return Values

When -write is not used, a Tcl list of lists is returned, where each sub-list has this form:

```
{<device_name>[(<model>)] <$D_index> <layer_name>}
```

The device_name is the device element name, possibly followed by a model name in parentheses. The \$D_index corresponds to the \$D property in the extracted netlist. This identifies which Device statement in the rules classified the device. The layer_name is the name of the annotation layer.

Then -write is used, the output format is like this:

```
Annotated_Device_Layer_map <precision>
Device_Layer_Mappings:
 0 0 <k> <date>                                // k layer entries
 <dev> <dev_idx> <layer_name>
  ...
  ...
```

[Tcl Shell Error Messages](#).

Description

Returns a list of lists of any layers that are used for annotations on particular devices. Each sub-list contains the device type and subtype (if a subtype is present), the device table index, and the DFM property annotation layer name for any layer that is currently mapped. LVS Annotate Devices must be specified in the rules for this command to be useful.

If conflicting layers are present as discussed under “[Conflicting Annotated Device Layers](#)” on page 435, only one layer of the set of conflicting layers is reported by this command.

Examples

This shows the command output from an interactive shell session:

```
> qs::annotated_device_layer_map
Loading layer DEV_LAYER_MP into the hierarchical database
Loading layer DEV_A_LAYER_MN into the hierarchical database
Loading layer DEV_P_LAYER_MN into the hierarchical database
Loading layer DLAYER_MN_PROP into the hierarchical database
{mn 0 DEV_A_LAYER_MN} {mp 1 DEV_LAYER_MP}
```

Related Topics

[Tcl Shell CCI Command Reference](#)

qs::agf_map

Annotated Geometry Format CCI command. Corresponding standard commands: [AGF MAP](#) and [AGF SVRF MAP](#). A Calibre Connectivity Interface license is required for this command. Mask SVDB Directory CCI must be specified in the rules used to create the loaded database. Also used in Calibre YieldServer.

Returns the layer map scheme of annotated layout output.

Usage

```
qs::agf_map [{-remap calibre_layer design_layer design_datatype [-device | -original]} |  
 {{-svrf filename} | {-write filename} } [-filter filter_object] } ] [-help]
```

Alternate Command Name

[**qs::gds_map**](#)

Arguments

- **-remap *calibre_layer design_layer design_datatype***

Optional argument set that specifies to map a Calibre layer to an AGF output layer number and datatype. The *calibre_layer* is a layer name. The *design_layer* and *design_datatype* arguments are an output design layer number and datatype, respectively. Care should be taken that downstream tools accept the mapped layer and datatype values. This option is mutually exclusive with -svrf and -write.

- **-device**

An optional argument used with -remap that specifies recognized device layers are mapped. This is the default setting, and is similar in purpose to the standard command AGF MAP ... DEVICE output. May not be specified with -original.

- **-original**

An optional argument used with -remap that specifies original seed layers are mapped. This is similar in purpose to the standard command AGF MAP ... ORIGINAL output. May not be specified with -device.

- **-svrf *filename***

Optional argument set that specifies SVRF compliant output is sent to the *filename*. This is analogous to AGF SVRF MAP YES output. This option is mutually exclusive with -remap and -write.

- **-write *filename***

Optional argument set that specifies to output the default map to the specified *filename*. This option is mutually exclusive with -remap and -svrf.

- **-filter *filter_object***

Optional argument set that specifies a [dfm::create_filter](#) object. When this option is used, the output includes just the layers referenced in the *filter_object*. This option may only be

used with -svrf or -write. This argument set is analogous to [FILTER LAYERS](#) in the standard interface.

- -help

An optional argument that shows the command usage.

Return Values

If qs::agf_map is used without any options, a Tcl list of lists is returned. Each sublist in the output has this form. Quoted braces are literal; otherwise braces are for syntactical grouping.

```
'{ '<layer_name> <number> <datatype> { '{}' | DEVICE | ORIGINAL |  
TENTATIVE_PROMOTION}' }'
```

If the -svrf option is used, each line in the output file has this form:

```
LAYER { "<name>" | "<name>_ORIGINAL" | "<name>_TENTATIVE_PROMOTION" }  
<number> [// DEVICE]
```

If the -write option is used, then each line representing a layer in the output file has this form:

```
<name> <number> <datatype> [DEVICE | ORIGINAL | TENTATIVE_PROMOTION]
```

Tcl Shell Error Messages.

Description

Without any optional arguments, this command returns the current Calibre layer name to layer number and datatype mapping for all [Device](#) layers output by a [qs::write_agf](#) command.

The name qs::gds_map is an alternate and historical form for the command.

With no options specified, qs::agf_map reports mappings only for layers containing recognized device seed shapes. In a corresponding output AGF file, each shape is annotated with the device name as a string property (like M0).

The default output is a Tcl list of lists. Each sublist represents a layer mapping and contains four fields: Calibre layer name, AGF layer number, AGF datatype (0 by default), and a null list {}. If [qs::set_agf_options](#) -seed_property is used, then the null list is replaced by a keyword, depending on which -seed_property options are used. The -seed_property configuration also affects the outputs of the -svrf and -write options.

The mapping of Calibre layer numbers to AGF layer and datatype numbers is assigned by the system unless the -remap option is used beforehand to define the mapping. Note that qs::agf_map does not configure the AGF output layers themselves. Use the [qs::set_agf_options](#) command for this purpose.

The -original option configures the output layer as an original layer. By default, or when -device is specified, the output is configured as a device seed layer. To configure both -device and -original, two qs::agf_map commands are required.

The difference between -device and -original shapes is seed shapes can be promoted to other cells by interaction between device layers. Also, original layer shapes do not have the recognized device name annotation in the AGF output. (They can have a node number if the seed shape is a connectivity layer.)

When the -write option is used, the output map is written to the specified *filename* using a format like the standard Query Server response format.

When the -svrf argument set is used, layer mappings are written to the given *filename* in SVRF format suitable for inclusion in a rule file or for use in Calibre DESIGNrev for naming layers in the layer palette.

Examples

This is an example of the default output:

```
> qs::agf_map

{pwell 1 0 {}} {nwell 2 0 {}} {poly 3 0 {}} {psd 4 0 {}} {pgate 6 0 {}}
{ptap 7 0 {}} {nsd 8 0 {}} {ngate 10 0 {}} {ntap 11 0 {}} {metal1 12 0 {}}
{contact 13 0 {}} {via 14 0 {}} {metal2 15 0 {}}
```

This shows the output when the -svrf option is used:

```
LAYER "pwell" 10
LAYER "nwell" 2
LAYER "poly" 3
LAYER "psd" 4
LAYER "pgate" 6 // DEVICE
LAYER "ptap" 7
LAYER "nsd" 8
LAYER "ngate" 10 // DEVICE
LAYER "ntap" 11
LAYER "metal1" 12
LAYER "contact" 13
LAYER "via" 14
LAYER "metal2" 15
```

This shows the output when the -write option is used:

```
Gds_Map 1000
Layers:
0 0 13 May 28 12:26:49 2020
pwell 10 0
nwell 2 0
poly 3 0
psd 4 0
pgate 6 0
ptap 7 0
nsd 8 0
ngate 10 0
ntap 11 0
metall 12 0
contact 13 0
via 14 0
metal2 15 0
```

For information about tentative seed promotion analysis, see the Examples section under [qs::set_agf_options](#).

Related Topics

[Tcl Shell CCI Command Reference](#)

qs::log_cci_commands

CCI administrative command.

Provides a trace log for certain commands of interest in CCI runs. The log provides a record of command calls in chronological order.

Usage

```
qs::log_cci_commands {-start filename | -stop | -list_commands | -help}
```

Arguments

- **-start *filename***

An argument set that specifies to begin command tracing and to write a log to the specified *filename*. Specifying an existing *filename* overwrites the file.

- **-stop**

An option that specifies stop tracing and to close the current log file.

- **-list_commands**

An option that specifies to return a list of traceable commands.

- **-help**

An option that shows the command usage.

Return Values

-list_commands returns a list. **-help** returns a help message. Other options return nothing.

[Tcl Shell Error Messages](#).

Description

Provides a trace log that lists commands that were called during a run in chronological order. The **-start** option starts the trace, which continues until a command with the **-stop** option is executed, or the end the run, whichever comes first.

The **-list_commands** option returns a list of the traceable commands, which include the following:

- “[Tcl Shell CCI Commands](#)” on page 434.
- Commands in the “qs::rules_” family. See “[LVS Rule File Query Commands](#)” on page 38.

This command is useful for human readability of **-list_commands** output:

```
join [qs::log_cci_commands -list_commands] "\n"
```

Examples

Example 1

This is an example trace log:

```
-----  
# CALIBRE::QS SERVER CCI Command Trace File - Wed Jul 12:48:33 2022  
#  
#     Calibre version:v2022.4_0.56 - Wed Jul 13 02:51:57 PDT 2022  
#  
#     /home/project/trees/latest/calibre -qs -svdb svdb -64  
#-----  
qs::log_cci_commands -start cci_trace.log  
qs::port_table  
...  
qs::log_cci_commands -stop
```

qs::port_table

CCI command. Corresponding standard command: [PORT TABLE WRITE](#). A Calibre Connectivity Interface license is required for this command. Also used in Calibre YieldServer.

Returns a listing of information about cell ports in the layout design.

Usage

```
qs::port_table [-cells] [-db_connectivity] [-filter filter_object]  
[-name_polygon_ports {NO | YES}] [-write filename] [-help]
```

Arguments

- **-cells**
An optional argument that returns port information for all cells. Cell names are added to the output, which by default, they are not. By default, only top-level ports are output. This is equivalent to the PORT TABLE CELLS WRITE command.
- **-db_connectivity**
An optional argument that specifies to substitute layers from [LVS DB Connectivity Layer](#) rule file statements for Connect or Sconnect layers (which are the default). Port objects that intersect layers from LVS DB Connectivity Layer statements are then listed as if the ports are attached to those layers.
- **-filter *filter_object***
An optional argument set that specifies a [dfm::create_filter](#) transformation object. If this argument set is specified, then the geometric transformations stored in the filter are applied to the output.
- **-name_polygon_ports {NO | YES}**
An optional argument set that controls whether <UNNAMED> ports get names in the output of the command. The default is NO. This option is equivalent in behavior to the [PORT TABLE NAME POLYGON PORTS](#) command.
- **-write *filename***
An optional argument that specifies to write a port table to the specified file. If the *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.
- **-help**
An optional argument that shows the command usage.

Return Values

When -write is not used, a Tcl list of lists is returned. Each sub-list is of this form:

```
{<port_name> <node_ID> <net_name> <x> <y> <attachment_layer_name>
{ [<cell_name>] }}
```

The element definitions are these:

port_name — Name of the port. If the port has no name (as for [Port Layer Polygon](#)), then the name is <UNNAMED> by default. The -name_polygon_ports YES option causes unnamed ports to have names.

node_ID — An integer corresponding to the node ID assigned by the circuit extractor.

net_name — Name of the net to which the port is attached. If the net is unnamed, then this argument is identical to the node_ID.

x y — Coordinates of the port in cell space.

attachment_layer_name — Name of the layer to which the port is attached.

cell_name — Name of the cell in which the port appears when the -cell option is used. By default, this field is empty.

With the -write option, a report is output as discussed under “[Port Table File Format](#)” on page 504.

[Tcl Shell Runtime Messages](#).

Description

This command returns a listing of all ports in the layout. If the -write option is not used, the output format is a Tcl list of lists; otherwise, the output is in tabular form written to a file.

By default, the layer a port is attached to is a Connect or Sconnect layer (hereafter referred to as a connect set layer). If the -db_connectivity option is used, then qualifying layers from LVS DB Connectivity Layer statements are listed instead as the layers that ports are attached to.

For brevity, the term “LDC” is used to stand for “LVS DB Connectivity Layer” in this discussion.

LDC layers are specified in the rule file in order to pass connectivity layers along with their nodal information to the Mask SVDB Directory for use in CCI flows. Assume the following rule file excerpt:

```
CONNECT M1
M1_A = M1 AND A
M1_B = M1 AND B
PORT LAYER TEXT M1
LVS DB CONNECTIVITY LAYER M1_A M1_B
MASK SVDB DIRECTORY svdb CCI
```

Assume that the port objects intersect layers M1 and possibly M1_A or M1_B, or all three. Given those things, the following could be the default output:

```
P1 1 1 1040 2000 M1
P2 1 1 1110 2000 M1
P3 1 1 1210 2000 M1
```

Notice that M1 is the port attachment layer.

When -db_connectivity is specified, the output shows any LDC layers a port intersects instead of connect set layers. So, the previous table could appear as follows:

```
P1 1 1 1040 2000 M1
P2 1 1 1110 2000 M1_B
P3 1 1 1210 2000 M1_A
```

Here, port P1 is attached to the Connect layer, port P2 intersects LDC layer M1_B, and port P3 intersects LDC layer M1_A.

If a port intersects more than one LDC layer, an entry appears in the table for each LDC layer to which a port can be attached. For example:

```
P1 1 1 1040 2000 M1
P2 1 1 1110 2000 M1_B
P3 1 1 1210 2000 M1_A
P3 1 1 1210 2000 M1_B
```

Here, port P3 intersects two LDC layers (which may not be your intent).

When LDC layers are specified in the rule file but -db_connectivity is not specified qs::port_table, then these messages appear in the CCI transcript:

NOTE: LVS DB CONNECTIVITY layers are present, but the DB_CONNECTIVITY option was not used.

NOTE: Ports are attached to the original CONNECT/SCONNECT layers.

A flow that uses LDC layers generally ensures these conditions are met:

- LDC layers completely cover associated connect set layer polygons. This is not a Calibre circuit extraction requirement, but may be required by downstream parasitic extraction tools.
- LDC layers reside in the same cells as their associated connect set layers. This requires care in the derivation of LDC layers so that interaction with shapes higher up in the hierarchy does not cause promotion out of cells in which port objects reside.
- LDC layer shapes from only one layer intersect port objects. That is, shapes from more than one LDC layer do not overlay a given port object attached to a connect set layer.

In cases where LDC layers do not intersect a port object, either due to layer promotion or lack of intersection with a port object, then the port object retains its original attachment to the connect set layer, and a warning like this is issued:

```
WARNING: Port <name>(<x>,<y>) in cell <name> was not reported in
DB_CONNECTIVITY mode.
```

Failure to enforce the third condition by having more than one LDC layer intersect a port object results in the port object being listed twice in a port table, which is shown previously.

Examples

This shows the default output:

```
qs::port_table
{GND 1 GND 1300 3600 metal2 {}} {<UNNAMED> 1 GND 20625 6250 metal2 {}}
{PWR 2 PWR 1300 82400 metal2 {}} {<UNNAMED> 2 PWR 20625 78875 metal2 {}}
{<UNNAMED> 3 3 20000 45100 metal2 {}}
```

The ports are all from the top level. The coordinates are in top-level space. Cell names are not reported in this case. Ports generated from Port Layer Polygon are shown as <UNNAMED> by default.

Related Topics

[Tcl Shell CCI Command Reference](#)

[Port Commands](#)

qs::set_agf_options

Annotated Geometry Format CCI command.

Configures AGF output settings for the [qs::write_agf](#) command.

Usage

qs::set_agf_options *option* [*option* ...]

Alternate Command Name

qs::set_gds_options

Arguments

One or more of the following must be specified. The options that are similar to standard CCI commands include descriptions with links to the related commands. The behavioral details are available on the linked pages. The main difference in behavior is these options configure [qs::write_agf](#) rather than AGF WRITE.

- **-annotated_devices {NO | YES}**

An optional argument set that controls whether annotated device seed layers are output instead of original seed layers. Default is NO. This argument set is analogous to [AGF ANNOTATED DEVICES](#).

- **-clear_merge_layers**

An optional argument set that clears the list of layers selected to be merged in the AGF output. This argument is analogous to [AGF CLEAR MERGE LAYERS](#).

- **-clear_pcell_promote_layers**

An optional argument set that clears the list of layers specified by a previous use of the -pcell_promote_layers option. This argument is analogous to [AGF CLEAR PCELL PROMOTE LAYERS](#).

- **-devprop_name *name***

An optional argument set that configures the name of the device instance name property for OASIS output. The default *name* is Device_Property. The property's value is a device instance name. The property is assigned to device seed shapes. This argument set is analogous to the DEVPROP option of [AGF {DEVPROP | NETPROP | PLACEPROP} NAME](#).

- **-devprop_number *number***

An optional argument set that configures the device instance name PROPATTR record number for PROPVALUE records in GDS output. The default is 1. The property is assigned to device seed shapes. This argument set is analogous to the DEVPROP option of [AGF {DEVPROP | NETPROP | PLACEPROP} NUMBER](#).

- -format {GDSII | {OASIS [NOSTRICT | STRICT]}}}

An optional argument set that specifies the layout format of the output. GDSII is the default. This argument set is analogous to [AGF FORMAT](#).

OASIS output uses CBLOCK compression. All OASIS records to which strict-mode formatting applies are output using that mode except for PROPSTRING records, to which these keywords apply:

NOSTRICT — An optional keyword that specifies PROPSTRING records are output in non-strict mode. This is the default when OASIS is specified. This keyword can reduce OASIS file output memory consumption by up to 3×. It may also substantially reduce memory consumption of downstream tools that use the output OASIS file as an input.

STRICT — An optional keyword that specifies PROPSTRING records are output in strict mode.

- -hierarchy {AGF | HCELL | FLAT}

An optional argument set that configures the hierarchy of the output design. The default is AGF. This argument set is analogous to [AGF HIERARCHY](#).

- -hierarchy_clear_expand_cells

An option that clears the current list of cells specified with the -hierarchy_expand_cell option. This argument is analogous to [AGF HIERARCHY CLEAR EXPAND CELLS](#).

- -hierarchy_expand_cell *cell_name*

An optional argument set that specifies the name of a cell to expand one level. This argument set is analogous to [AGF HIERARCHY EXPAND CELL](#).

- -hierarchy_expand_deviceless_cells {NO | YES}

An optional argument set that controls automatic expansion of cells that contain no devices in them or in the lower-level placements within them. The default is NO. This argument set is analogous to [AGF HIERARCHY EXPAND DEVICELESS CELLS](#).

- -magnify_user_units {YES | NO}

An optional argument set that specifies whether to apply [dfm::create_filter](#) -magnify objects, if any. The default is YES. This argument set is analogous to [AGF MAGNIFY USER UNITS](#).

- -merge_layer *layer*

An optional argument set that specifies the *layer* is to be merged in the AGF output. This argument set is analogous to [AGF MERGE LAYER](#).

- -merge_pin_layers *element_name*[(*model_name*)] *pin_name*

An optional argument set that specifies the pin layers corresponding to the *pin_name* of a device have the SPICE *element_name* and optional *model_name* are merged in the AGF output. This argument set is analogous to [AGF MERGE PIN LAYERS](#).

- **-netprop_name *name***

An optional argument set that configures the name of the net ID property for OASIS output. The default *name* is Net_Property. The property's value is a node number. The property is assigned to interconnect shapes. This argument set is analogous to AGF NETPROP NAME.

- **-netprop_number *number***

An optional argument set that configures the net number PROPATTR record number for PROPVALUE records in GDS output. The default is 1. The property is assigned to interconnect shapes. This argument set is analogous to AGF NETPROP NUMBER.

- **-pcell_promote_layers “*layer_list*”**

An optional argument set that specifies layers in the PHDB from PEX xcells and cells specified in a [PEX Preserve Cell List](#) statement that are available for promotion. At least one layer name must be specified in the *layer_list*.

If a layer is a Device seed layer, then the -seed_property ORIGINAL keyword must also be specified, otherwise the seed shapes are not promoted. “TENTATIVE_PROMOTION” information produced by the -seed_property TENTATIVE keyword is unaffected by pcell layer promotion.

If a layer is also specified by [dfm::create_filter](#) -box_layers, then the promotion occurs first, followed by the box layer filtering.

To clear the current set of promotion layers, use the -clear_pcell_promote_layers option.

This argument set is analogous to [AGF PCELL PROMOTE LAYERS](#).

- **-placeprop_name *name***

An optional argument set that configures the name of the placement name property for OASIS output. The default *name* is Placement_Property. The property's value is a placement path. The property is assigned to cell placements. This argument set is analogous to AGF PLACEPROP NAME.

- **-placeprop_number *number***

An optional argument set that configures the placement name PROPATTR record number for PROPVALUE records in GDS output. The default is 1. The property is assigned to cell placements. This argument set is analogous to AGF PLACEPROP NUMBER.

- **-reset**

An optional argument set that sets the AGF configuration options to their default values. This argument is analogous to [AGF RESET](#).

- **-seed_property {DEVICE | ORIGINAL | DEVICE ORIGINAL} [TENTATIVE]**

An optional argument set that configures the type of device seed properties that can appear in AGF output. The default is DEVICE, which annotates device instance names. The ORIGINAL keyword annotates net names. If the TENTATIVE keyword is specified, then the AGF output from [qs::write_agf](#) contains tentatively promoted device seed shapes, and the output from [qs::agf_map](#) contains "TENTATIVE_PROMOTION" entries for layers

with tentatively promoted shapes. This argument set is analogous to [AGF SEED PROPERTY](#).

- `-units precision dbu_size`

An optional argument set that configures the UNITS record of the AGF output. The *precision* specifies the physical precision of the output database. The value is typically on the interval (0, 0.001]. The reciprocal of the rule file [Precision](#) is the default. The *dbu_size* specifies the database unit in meters. The number may be floating-point or in scientific notation. The default is the ratio of the [Unit Length](#) rule file parameter to the rule file Precision. This argument set is analogous to [AGF UNITS](#).

- `-use_names {LAYOUT | NETLIST}`

An optional argument set that specifies which cell names to use in the output. The default is LAYOUT, which means to use the layout cell names. The NETLIST keyword specifies to use cell names from the extracted SPICE netlist.

The NETLIST keyword is useful when the layout contains cell names that begin with dollar signs (\$), which are disallowed in SPICE. Dollar signs are replaced with double underscore character (_) prefixes in extracted netlists, so the NETLIST keyword causes the output cell names to match this prefix mapping.

- `-help`

Shows the command usage.

Return Values

[Tcl Shell Error Messages](#).

Examples

This example shows a method for analyzing tentative device seed promotion. In an annotated layout flow, tentative seed promotion information appears in the circuit extraction transcript when the [Mask SVDB Directory](#) CCI NOPINLOC options are used or when [LVS Push Devices SEPARATE PROPERTIES YES](#) is used.

When the `qs::set_agf_options -seed_property device original tentative` keyword is used, the AGF writer can output geometries at the locations of tentative seed promotions. These geometries appear on a dedicated layer in the AGF file. In the `qs::agf_map` output, these layers are labeled with "TENTATIVE_PROMOTION" where <layer> is the name of the original seed layer of the devices. This output is helpful in determining the cause of seed promotion during device extraction. Here is a typical command sequence:

```
qs::set_agf_options -seed_property device original tentative
qs::agf_map -svrf qs_agf_map.rules
qs::write_agf qs_agf.gds
```

The SVRF rule file *qs_agf_map.rules* contains entries similar to this:

```
LAYER "psd" 4
LAYER "pgate_ORIGINAL" 5
LAYER "pgate" 6 // DEVICE
LAYER "pgate_TENTATIVE_PROMOTION" 16
LAYER "ptap" 7
LAYER "nsd" 8
LAYER "ngate_ORIGINAL" 9
LAYER "ngate" 10 // DEVICE
LAYER "ngate_TENTATIVE_PROMOTION" 17
LAYER "ntap" 11
```

In this example, the tentative seed promotion marker layers are 16 and 17 in the *qs_agf.gds* file.

The output AGF file can then be loaded into Calibre DESIGNrev. The output SVRF rule file can be loaded using the **Layer > Load Input SVRF Names** menu item. The layer palette will then contain the annotated layer names, along with the *_TENTATIVE_PROMOTION layers. You can toggle the visibility of layers as desired to debug seed promotion issues.

Related Topics

[Tcl Shell CCI Command Reference](#)

qs::write_agf

Annotated Geometry Format CCI command. Corresponding CCI command: [AGF WRITE](#). A Calibre Connectivity Interface license is required for this command. Also used in Calibre YieldServer.

Writes an annotated AGF file.

Usage

```
qs::write_agf filename [-filter filter_object] [-help]
```

Arguments

- *filename*

A required filename of the output file. If the *filename* ends in the .Z or the .gz suffix, the output file is compressed using the compress or gzip utility, respectively. Using compressed output suffixes assumes you have the appropriate utilities in your environment. Directories in a pathname are created if they do not exist.

- -filter *filter_object*

Optional argument set that specifies a [dfm::create_filter](#) object. When this option is used, the output includes just the layers referenced in the *filter_object*.

- -help

An optional argument that shows the command usage.

Return Values

None. Writes AGF file.

[Tcl Shell Error Messages](#).

Description

Writes layout geometry in AGF format to the specified *filename*. The [qs::set_agf_options](#) command configures the output.

By default, device seed shapes are fully merged on output by qs::write_agf. However, device seed layers might not be merged if the ORIGINAL option to [qs::set_agf_options -seed_property](#) is used in the flow. Also, the Calibre device extraction module does not require pin layers to be merged in order to extract a device successfully. However, certain parasitic extraction tools do require merged pin layers in order to generate accurate results. The [qs::set_agf_options -merge_layer](#) and [qs::set_agf_options -merge_pin_layers](#) options can be used to manage layer merging in the AGF output.

For GDS output, device instance names, net IDs, and placement names are written to the property records PROPATTR and PROPVVALUE. By default, the PROPATTR record value is 1 for all properties. The value for PROPATTR records may be changed using the [qs::set_agf_options -devprop_number](#), [qs::set_agf_options -netprop_number](#), and [qs::set_agf_options -placeprop_number](#) options.

For OASIS output, device instance names, net IDs, and placement names are written to the property records PROPNAME and PROPSTRING. The values for the PROPNAME records can be changed using the qs::set_agf_options -devprop_name, -netprop_name, and -placeprop_name.

The maximum vertex count is 200 (same as the GDSII standard).

By default, the database precision is the reciprocal of the rule file [Precision](#) setting. The database precision can be changed with the qs::set_agf_options -units option.

The -filter option constrains the output to just the layers in the filter object. If the filter object is created with the -box_layers option, then any [LVS Box](#) cells that are written contain only the layers specified with that option. Otherwise, box cells are written with all layers that CCI outputs for other cells.

This command requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license. Refer to the [Calibre Administrator's Guide](#) for license information.

Examples

Example 1

```
# output layer map and devices for downstream tools
qs::agf_map -device -original -svrf gds_map.svrf
qs::write_agf devices.gds.gz
```

Example 2

```
# create filter object with three layers
set filter [ dfm::create_filter -filter_layers "met4 met5 cap" ]
# AGF output consists of only those three layers
qs::write_agf svdb.agds -filter $filter
```

Related Topics

[Tcl Shell CCI Command Reference](#)

qs::write_cell_extents

CCI command. Corresponding standard command: [CELL EXTENTS WRITE](#). A Calibre Connectivity Interface license is required for this command. Also used in Calibre YieldServer.

Writes a cell extents file.

Usage

```
qs::write_cell_extents filename [-filter filter_object] [-help]
```

Arguments

- *filename*

A required filename of the output file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- -filter *filter_object*

An optional argument set that specifies a [dfm::create_filter](#) transformation object. If this argument set is specified, then the geometric transformations stored in the filter are applied to the output.

- -help

An optional argument that shows the command usage.

Return Values

[Tcl Shell Error Messages](#).

Examples

The output file contains information in this format:

```
# SVDB: Cell Extents (File format 1)
# SVDB: Layout Primary <cell>
# SVDB: Rules -0 <filename> <time stamp>
# SVDB: GDSII -0 <filename> <time stamp>
# SVDB: SNL -0 <filename>
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
<cell> x1 y1 x2 y2
...
```

The SVDB header is followed by a list of cell names and coordinate pairs of the lower-left and upper-right vertices of the cell extents.

Related Topics

[Tcl Shell CCI Command Reference](#)

[qs::write_layout_netlist_names](#)

CCI command. Corresponding standard command: [LAYOUT NAMETABLE WRITE](#). A Calibre Connectivity Interface license is required for this command.

Writes a Layout Netlist Names correspondence file.

Usage

`qs::write_layout_netlist_names filename [-expand_cells] [-invalid] [-help]`

Arguments

- *filename*

A required filename of the output file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- -expand_cells

An optional argument that specifies the output hierarchy adheres to a non-default [dfm::set_layout_netlist_options -hierarchy](#) setting.

- -invalid

An optional argument that specifies only invalid net names are returned. By default, only valid net names are output.

- -help

An optional argument that shows the command usage.

Return Values

LNN file as discussed under [Layout Netlist Names File Format](#).

[Tcl Shell Error Messages](#).

Description

Writes a Layout Netlist Names (LNN) file describing correspondence of generated net numbers to original layout names. It is generally recommended to produce this file in CCI flows. The LNN file is used in conjunction with the AGF file produced by [qs::write_agf](#) and a customized netlist produced by [dfm::write_spice_netlist](#).

The [dfm::set_layout_netlist_options -names NONE](#) option is used with `qs::write_layout_netlist_names`.

The -expand_cells option overcomes hierarchical inconsistency between the flattened output of the CCI netlist and the Layout Netlist Names (LNN) file. Such inconsistencies can occur when the [dfm::set_layout_netlist_options -hierarchy](#) setting is other than ALL. Note that when lower-level net names are expanded into higher-level cells, only top-level net names are printed at the

lower levels. Net names that connect upward to other higher-level nets in the containing cell are not printed since they are represented by the higher-level net name in the CCI netlist and in the AGF file. For example, given a top-level cell contains a cell ICV_1 that is expanded in AGF output and has the following net structure:

```
.SUBCKT TOP
X0 net_A ICV_1
.ENDS

.SUBCKT ICV_1 AAA
M0 AAA BBB CCC
.ENDS
```

the net AAA in ICV_1 connects upward to net_A in cell TOP while nets BBB and CCC do not extend upward. In the LNN file, the nets BBB and CCC are represented while AAA is not:

```
% TOP
net_A 1
X0/BBB X0/1
X0/CCC X0/2
```

Certain text names are not used for SPICE netlist net names by Calibre. These net names are not included in the output by default. The -invalid option causes the command to return only those names.

Requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license. Refer to the [Calibre Administrator's Guide](#) for license information.

Examples

Sample output file:

```
# SVDB: Layout Netlist Names (lnn) (File format 1)
# SVDB: Layout Primary TOPCELL
# SVDB: Rules -0 rules Thu Oct 13 10:32:25 2016
# SVDB: GDSII -0 ./inv_nand.oas Tue Dec 28 18:34:47 2010
# SVDB: SNL -0
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
% nand 6
% CellA 4
% inv 5
% CellB 4
% TOPCELL 0
2 GND
3 PWR
```

Related Topics

[Tcl Shell CCI Command Reference](#)

qs::write_lvs_settings_report

Rule file query command. Corresponding standard command: [LVS SETTINGS REPORT WRITE](#).

Returns a summary report of LVS rule settings.

Usage

`qs::write_lvs_settings_report filename [-help]`

Arguments

- *filename*
A required pathname of the report output file.
- -help
An optional argument that shows the command usage.

Return Values

Report.

[Tcl Shell Error Messages](#).

Description

Returns a report file with a summary of LVS rule file settings to the specified *filename*. The report format is as follows:

```
# SVDB: <software version> <time stamp>
# SVDB: LVS Settings Report (lsrf) (File format 1)
# SVDB: Layout Primary <cell>
# SVDB: Rules -0 <filename> <time stamp>
# SVDB: <format> -0 <filename> <time stamp>
# SVDB: SNL -0 <filename>
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
HIERARCHY_SEPARATOR "<character>"
LVS_POWER_NAME <list of power nets>
LVS_GROUND_NAME <list of ground nets>
SOURCE_PATH <filename>
SOURCE_SYSTEM <format>
UNIT_LENGTH <number>
PRECISION <number>
LVS_COMPARE_CASE <NO | YES>
LAYOUT_CASE <NO | YES>
SOURCE_CASE <NO | YES>
LVS_SPICE_REPLICATE_DEVICES <NO | YES>
LVS_DB_LAYER <layer list>
LVS_DB_CONNECTIVITY_LAYER <layer list>
LVS_DOWNCASE_DEVICE <NO | YES>
# LAYER TYPES:
<layer_name> <type>
# CONNECTIVITY:
CONNECT ...
# IMPLICIT CONNECTIVITY:
IMPLICIT_CONNECT <derived_layer> <input_layer>
# LAYER ALIASES
ALIAS <layer1> <layer2> ...
```

The connectivity response uses CONNECT when the connection is due to a [Connect](#) or [Sconnect](#) statement. If the connectivity is established through node-preserving layer derivation, then an IMPLICIT CONNECTIVITY section appears.

Layer aliases are different names for the same layer. For example, a rule file could have the following code, where alias1 and alias2 are logically equivalent:

```
alias1 = diff AND poly
alias2 = diff AND poly
```

These appear as ALIAS layers at the end of the report.

Definitions of layer type categories are given under [LVS SETTINGS REPORT WRITE](#).

Related Topics

- [LVS Rule File Query Commands](#)
- [Tcl Shell CCI Command Reference](#)

qs::write_separated_properties

CCI command. Corresponding standard command: [LAYOUT SEPARATED PROPERTIES WRITE](#). A Calibre Connectivity Interface license is required for this command.

Writes pushdown separated properties (PDSP) to a file.

Usage

qs::write_separated_properties *filename* [-filter *filter_object*] [-help]

Arguments

- ***filename***
A required filename of the output file. Directories in a pathname are created if they do not exist.
- **-filter *filter_object***
An optional argument set that specifies a [dfm::create_filter](#) transformation object. If this argument set is specified, then the geometric transformations stored in the filter are applied to the output.
- **-help**
An optional argument that shows the command usage.

Return Values

Output file format is shown under “[Separated Properties File Format](#)” on page 509.

[Tcl Shell Error Messages](#).

Description

Specifies a file to which the properties written by [LVS Push Devices](#) SEPARATE PROPERTIES YES are saved. The LVS Push Devices SEPARATE PROPERTIES YES statement must be in the rule file for this command to be used.

Related Topics

[Tcl Shell CCI Command Reference](#)

Rule File Query Commands

Rule file query commands provide information about LVS settings in the rule file used to generate an SVDB. These are useful in a CCI flow. As with most Query Server commands, there are Tcl shell and standard shell variants.

[qs::write_lvs_settings_report](#)

[Tcl Shell Rule File Query Commands](#)

[LVS SETTINGS REPORT WRITE](#)

[Standard Rule File Query Commands](#)

ERC TVF Commands

ERC TVF commands support device parameter calculations for the pushdown separate properties (PDSP) flow. These device parameters are only available from the PDSP files generated by CCI. The commands are Tcl-based and appear in TVF Function procedures referenced by the ERC TVF specification statement in the rule file.

ERC TVF commands are not executed in the Query Server Tcl shell. However, they are only used in a PDSP flow, which relies on CCI. So, they are documented together with the Query Server APIs that support the PDSP flow.

[Property Definitions and Netlist Behavior for CCI Flows](#)

erc::is_preflight_run	465
erc::setup_device_parameters	467
Stacked Transistor Handling in ERC TVF	473

erc::is_preflight_run

ERC TVF command.

Returns TRUE (1) if the current run is in “Preflight” testing mode and FALSE (0) during Calibre executive modules. This command is useful to prevent Tcl code from being executed when the Tcl interpreter validates code in an ERC TVF procedure.

Syntax

erc::is_preflight_run

Arguments

None.

Return Values

Integer.

Description

Note

 It is not required to use this command to prevent `erc::setup_device_parameters` from running during Preflight validation. Use it only when needed.

When **ERC TVF** is specified in the rule file, procedures that are called from that statement are validated in Preflight mode immediately after rule file compilation, as shown here in the transcript:

```
--- STANDARD VERIFICATION RULE FILE COMPILATION MODULE COMPLETED.  CPU
TIME = 0  REAL TIME = 0  LVHEAP = 1/3/3
ERC TVF Preflight...
```

The `erc::is_preflight_run` command can prevent Tcl code from being executed in Preflight mode by inserting a conditional block like this in a proc:

```
if { [ ::erc::is_preflight_run ] } { return 0 }
```

Prefixing the command with the `::` operator causes the command to be executed in the global scope.

Examples

```
ERC TVF erc1 erc_lib test_proc

TVF FUNCTION erc_lib /* 
    package require CalibreLVS_ERC_TVF_utilities

    proc test_proc {} {
# do not run this during preflight validation
        if { [ ::erc::is_preflight_run ] } { return 0 }

# other code executed outside preflight mode
        ...
    }
*/]
```

Related Topics

[ERC TVF Commands](#)

erc::setup_device_parameters

ERC TVF Command.

Calculates parallel device count and global coordinates properties for the CCI pushdown separated properties (PDSP) flow. These properties are not computed during LVS circuit extraction (calibre -spice), so they do not appear in the extracted layout netlist, nor are they available during LVS comparison.

Syntax

```
erc::setup_device_parameters -devices device_list [-globalx prop_x -globaly prop_y]
  [-add {'input_prop output_prop ['{'grouping_prop
    {'grouping_prop tolerance '}'} ... '}']}
  [-total {'count_prop ['{'grouping_prop | {'grouping_prop tolerance '}'} ... '}']}
  [-nets {'netID_prop [default_total] '}']
  [-no_prop_error] [-senpr] [-series]
  [-help]
```

Arguments

- **-devices *device_list***

A required argument set that specifies a Tcl list of device types and optional subtypes. Device types and subtypes are specified as in a [Device](#) statement: type(subtype). At least one device type must be in the *device_list*.

- **-globalx *prop_x* -globaly *prop_y***

An optional argument set that specifies to output location properties of the names *prop_x* and *prop_y* to the CCI netlist and separated properties file. The -globalx argument must be specified with -globaly. The *prop_x* and *prop_y* arguments must satisfy SPICE syntax criteria for property names. The output values associated with these property names are the top-level x and y coordinates in user units of the device seed shapes in the *device_list*.

This argument set causes flattening of all specified devices written to the output.

- **-add {'*input_prop output_prop* ['{'*grouping_prop*
 {'*grouping_prop tolerance* '}'} ... '}']}**

An optional argument set that calculates the sum of values of a given property for device instances of the same type and optional subtype in the *device_list* that are connected in parallel to other instances of the same classification. For instances that are not in parallel, *output_prop* is equal to *input_prop*. Parallel connections are determined across the hierarchy.

The arguments after the -add option are specified in a Tcl list of lists. These are the argument definitions:

input_prop — The name of the property whose values are summed. This parameter must be specified. Instances must have the specified property; otherwise, the instance is skipped for property summation, an *output_prop* is not written, and an error message is given by default.

output_prop — The name of the property that contains the sum of the *input_prop* values. This parameter must be specified.

grouping_prop — An optional name of layout netlist property (typically L or W) by which to group parallel instances. One or more *grouping_prop* arguments may be specified in a list as a part of the argument set. Instances must have the specified property in order to be grouped together; otherwise, the instance is skipped for property calculation, the output property is not written, and an error message is given by default. If *grouping_prop* is omitted, then device properties are not considered when grouping instances. If *tolerance* is specified with a *grouping_prop*, then each pair must be specified in its own list. Properties specified by this argument must be LVS-actionable (as defined under [LVS Netlist Property](#)).

tolerance — An optional non-negative floating-point number interpreted as a percent. If specified, it must appear in the same sub-list as a *grouping_prop*. By default, the *tolerance* is 0. Instances must be within a *tolerance* percentage difference of the *grouping_prop*'s associated numeric value in order to be considered part of the same parallel group.

- -total '{' *count_prop* ['{' {*grouping_prop* | '{' *grouping_prop tolerance* '}'} ... '}']
An optional argument set that calculates the number of instances of a given type and optional subtype in the *device_list* that are connected in parallel to other instances of the same classification. For instances that are not in parallel, the count is 1. Parallel connections are determined across the hierarchy. Parallel instances that are counted can optionally be grouped by a property name alone or by a property name with a tolerance for the property's value.

The arguments after the -total option are specified in a Tcl list of lists. The *count_prop* is the name of the output device property associated with the parallel count value. This parameter must be specified when -total is used. The *grouping_prop* and *tolerance* parameters are as described under -add. If the *grouping_prop* is missing for an instance, then the *count_prop* is not written, and an error is given by default.

- -nets '{' *netID_prop* [*default*] '}'
An optional argument set that groups parallel instances based upon the DFM property *netID_prop*.
Everything after -nets is a Tcl list. The *netID_prop* corresponds either to a numeric node ID or a -1 value. The *default* is an optional numeric value specifying a default parameter value for instances that do not have a node ID (that is, *netID_prop* has a value of -1).

- -no_prop_error
An optional argument that suppresses runtime error messages that are given when a device instance lacks a *grouping_prop*, or an *input_prop* when -add is used. When -no_prop_error is specified, such instances are quietly skipped for property summation calculations.
- -senpr
An optional argument that transforms the netlist graph by performing the same processing as the [LVS Short Equivalent Nodes PARALLEL statement](#). See “[Example 4](#)” on page 473.

- -series
An optional argument that causes series stacked transistors in parallel branches to be treated as parallel devices for post-layout simulation. See “[Stacked Transistor Handling in ERC TVF](#)” on page 473 for detailed information on the use of this option.
- -help
An optional argument that shows the command usage.

Return Values

None.

Description

This command is used in the CCI PDSP flow to calculate properties available to downstream simulation tools. This command is part of the CalibreLVS_ERC_TVF_device package and appears in a Tcl procedure called by the [ERC TVF](#) specification statement in the rule file. A Calibre ADP license is required.

The **-devices** argument defines the devices for which properties are calculated. Only instances with d, g, and s pins are processed. The devices are specified as they appear in Device statements in the rules, for example:

```
-device {MP(phv) MN(nhv)}
```

If a type is specified without a subtype, then that type covers all subtypes.

When used, the -globalx and -globaly options must be specified together. These options cause flattening of the design to get the top-level coordinates of specified devices; therefore, use of these options should be restricted to small blocks or devices that appear infrequently in the design. The reported coordinates are based upon the [LVS Center Device Location](#) rule file setting. YES is recommended when using -globalx and -globaly.

The -total option causes a property to be inserted in the output indicating the number of instances of the same type and subtype in the **device_list** that are connected in parallel. The minimum count of parallel instances is 1, which means the instance has no parallel counterpart. The option’s arguments can be specified as a list of lists, such as here:

```
-total { mos_in_parallel { L W } }
```

In this case, the values of the layout properties L and W are used to group the instances in parallel. The rule file must be configured so that the specified grouping properties are calculated and are LVS actionable. Appearance in [Trace Property](#) statements is sufficient to meet this condition.

Instances not having the specified grouping properties are skipped for property summation, the *count_prop* or *output_prop* is not written, and this error is given by default:

Error: Property "grouping_prop" not found on device <instance> of <type> in cell <cell> when grouping <output> property.

This error can be suppressed by using the -no_prop_error option.

By default, the *tolerance* for matching netlist property values is 0 percent. That is, the property values must match exactly in order for the instances to be grouped together as parallel. If you want a tolerance of 1 percent difference for both L and W, then the previous example could be modified as follows:

```
-total { mos_in_parallel { { L 1 } { W 1 } } }
```

The -nets option facilitates grouping parallel instances based upon a DFM property having a node ID value. The property is typically assigned by a marker layer. See [Example 2](#).

The -add option behaves similarly to the -total option. The main difference is -add sums the values of an input property rather than the number of instances. Non-parallel instances are processed with the sum effectively being *input_prop* + 0. As with -total, the sum is written to an output property. The -add behavior for grouping properties is the same as for -total.

Instances not having a specified *input_prop* are skipped for property summation, the *output_prop* is not written, and this error is given by default:

Error: Property "input_prop" not found on device <instance> of <type> in cell <cell> when processing *output_prop* property.

This error can be suppressed by using the -no_prop_error option.

The -senpr option alters the netlist graph for parallel devices in split-gate structures by shorting together equivalent nodes of parallel branches.

The calculated properties are available in the outputs of the [LAYOUT NETLIST WRITE](#) or [dfm::write_spice_netlist](#) commands when [LAYOUT NETLIST SEPARATED PROPERTIES YES](#) or [dfm::set_layout_netlist_options](#) -separated_properties is used. The properties are also available in the file generated by [LAYOUT SEPARATED PROPERTIES WRITE](#) or [qs::write_separated_properties](#).

See “[Customized Files for Pushdown Separated Properties \(PDSP\) Flow](#)” on page 514 for more details about the commands used with the PDSP flow.

Prefixing the command with the :: operator causes the command to be executed in the global scope.

Examples

Example 1

This shows a basic rule file configuration. A mos_par_total property is calculated for MN and MP devices. The property corresponds to the number of instances connected in parallel.

```
ERC TVF erc1 erc_lib mos_params

TVF FUNCTION erc_lib /* 
    package require CalibreLVS_ERC_TVF_device

    proc mos_params {} {
        puts "      EXECUTING mos_params"
        ::erc::setup_device_parameters -devices {MN MP} \
            -total { mos_par_total { L W } } \
            -globalx "xloc" -globaly "yloc"; # only use these for small blocks
    }
*/]

LVS PUSH DEVICES SEPARATE PROPERTIES YES
MASK SVDB DIRECTORY svdb QUERY CCI

TRACE PROPERTY MN L L
TRACE PROPERTY MN W W
TRACE PROPERTY MP L L
TRACE PROPERTY MP W W
```

For this CCI command run subsequently to LVS:

```
qs::write_separated_properties pdsp.data
```

notice the properties in the output file:

```
X0/X0/M0 xloc=5.125e-06 yloc=5.1e-05 as=1.075e-10 ad=1.25e-10 ps=5.075e-05
pd=5.25e-05 mos_par_total=1
$PIN_XY=6875,61000,5125,61000,5125,61000,5125,61000 $X=5125 $Y=51000 $D=1
```

Example 2

This shows the use of the -nets option.

```
// establish connectivity on marker layer to indicate devices
// sharing the same net ID
CONNECT marker

// annotate MP devices using mp_prop_layer
LVS ANNOTATE DEVICES mp_prop_layer

// if the marker layer intersects MP seed shapes, then the node ID of
// marker is assigned to the MKNET property; otherwise, the "NONETID"
// value is assigned.
mp_marker = DFM PROPERTY (DEVICE LAYER MP) marker OVERLAP MULTI
    [ MKNET = (COUNT(marker) > 0) ? NETID(marker, 1) : NONETID() ]
```

```
// annotate MP seed shapes with the numeric mkn property
mp_prop_layer = DEVICE LAYER MP ANNOTATE mp_marker [
    PROPERTY mkn
    mkn = DFM_NUM_VAL(mp_marker, MKNET)
]

// calculate parallel parameters for devices
ERC TVF parallel_check ERC_parallel_props annotate_parallel_properties

TVF FUNCTION ERC_parallel_props /*/
package require CalibreLVS_ERC_TVF

proc annotate_parallel_properties {} {
    # for MP devices not interacting with marker shapes, assign para=0
    ::erc::setup_device_parameters -devices {MP} -total {para} \
        -nets {mkn 0}

    # this causes an error because ARB devices have no mkn property
    ::erc::setup_device_parameters -devices {ARB} -total {para} \
        -nets {mkn}
}
*/]
```

MP instances receive an “mkn” property with either a node ID value from a marker shape or a -1 value. MP instances with the same property value are placed into distinct groups. The “para” property is assigned to each group based upon parallel MP instances in that group. Since a *default_value* is specified, instances with the -1 “mkn” value receive a “para” value of 0. The second command in the `annotate_parallel_devices` proc causes an error because ARB devices have no mkn property.

Example 3

Assume the following command:

```
::erc::setup_device_parameters -devices {MP MN} \
    -add {nf sum_nf {L { W 1 }}}}
```

This command sums the nf property values of MP and MN instances connected in parallel having matching L property values and W properties within a 1% tolerance of each other. The total is the value of the sum_nf property. These properties are stored in the PHDB in a PDSP flow. In a subsequent Query Server run, `qs::write_separated_properties` command could output this:

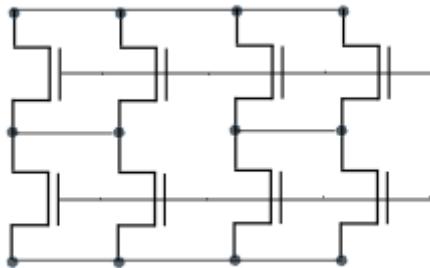
```
X7/X3/M4 sum_nf=2 $PIN_XY=443207,147150,436463,147150,436463,147150
$X=436463 $Y=117105 $D=1
X7/X3/M5 sum_nf=4 $PIN_XY=492873,139077,486129,139077,486129,139077
$X=486129 $Y=117105 $D=1
X7/X3/M6 sum_nf=4 $PIN_XY=505546,139077,498802,139077,498802,139077
$X=498802 $Y=117105 $D=1
```

Notice the sum_nf properties. Devices M5 and M6 are in parallel with each other, so their nf values are both 2 with sum_nf being 4. M4 is not in parallel with any instance, so its nf and sum_nf values are 2.

If the nf, L, or W properties are missing on any instance, then an error is given by default, and the instance is written without the sum_nf property.

Example 4

For this circuit:



without the -senpr option, the -total option would compute a property value of 2 for each device (four sets of pairs of parallel transistors). With the -senpr option, the “total” property value would be 4 (two sets of four parallel transistors).

Related Topics

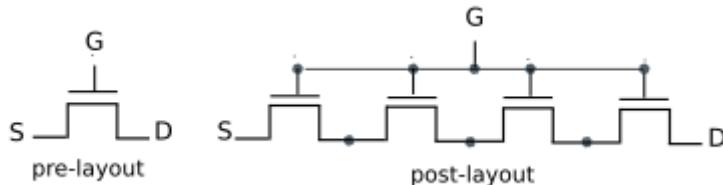
[ERC TVF Commands](#)

Stacked Transistor Handling in ERC TVF

ERC TVF supports special handling of stacked transistors through the `erc::setup_device_parameters` -series option.

One challenge in some FinFET processes is limited gate length. A possible solution is the stacked transistor.

Figure 6-1. Series Stacked Transistors

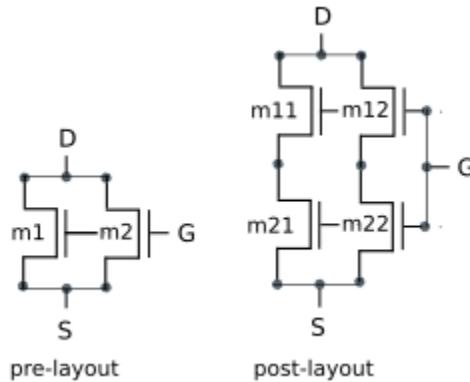


In [Figure 6-1](#), four transistors are connected in series to increase the effective gate length. In pre-layout simulation, there is a single transistor, whereas in post-layout simulation, there are four transistors. This construct is present in parallel branches of stacked transistors, which requires special handling in ERC TVF.

Parallel Stacked Transistors

Parallel device calculations for the stacked transistor could be different between pre-layout simulation and post-layout simulation. Consider the following figure:

Figure 6-2. Parallel Stacked Transistors



The pre-layout connection configuration has transistors parallel, while the post-layout configuration does not (it has stacks in parallel). Notice the S pins of M11 and M12 are not connected together in the post-layout case.

To address this issue, `erc::setup_device_parameters` has a `-series` option. In [Figure 6-2](#), when `-series` is specified, M11 and M12 are considered to be in parallel, as are M21 and M22. When `-series` is used with `-total` to derive the number of total parallel connected devices, the output property value is 2 for each of the four devices (2 per row).

Rule File Setup

When `-series` is specified, the algorithm to derive the parallel device property is altered, which affects both stacked and non-stacked devices. One way to address this issue is to provide a marker layer to annotate the stacked devices and apply the stacked treatment only to them. Here is an example:

```
CONNECT MARKER
MARKER_p = DFM PROPERTY gate MARKER
    [ MKNET = ( COUNT(MARKER) > 0 ) ? NETID(MARKER, 1) :
        NONETID() ]

MN_MKN_layer = DEVICE LAYER MN(n) ANNOTATE MARKER_p
    [ PROPERTY mkn
        mkn = DFM_NUM_VAL(MARKER_p, MKNET)
    ]

LVS ANNOTATE DEVICES MN_MKN_layer
```

For devices interacting with the MARKER layer, the property `mkn` provides the NETID of the MARKER layer polygon where the devices reside. When devices do not interact with the

MARKER layer, the value of property mkn is -1. The parallel device properties can then be derived by two ERC TVF checks:

```

ERC TVF check1 Parallel_properties annotate_adp_properties
ERC TVF check2 Parallel_properties annotate_adp_properties_stack

TVF FUNCTION Parallel_properties /* 
    package require CalibreLVS_ERC_TVF
    proc annotate_adp_properties {} {
        ::erc:::setup_device_parameters -devices { MN(n) } \
        -total { total }
    }
    proc annotate_adp_properties_stack {} {
        ::erc:::setup_device_parameters -devices { MN(n) } \
        -total { total } -nets { mkn 0 } -series
    }
*/]
```

“check1” derives the total number of parallel connected devices for each MN(n) device. “check2” performs a similar calculation except that only MN(n) devices interacting with the MARKER layer are taken into account. Only devices interacting with the same MARKER layer polygon area are considered as being in parallel with each other. The total parameter derived in “check2” overwrites the one derived in “check1” for those devices.

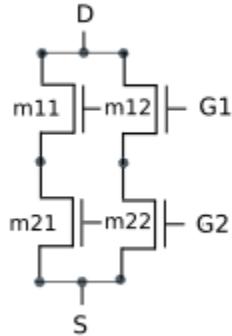
Devices are grouped according to parallel connectivity and can be grouped by property values. In the post-layout schematic of [Figure 6-2](#), assume all four devices have the same width, while the top row devices have the same length, and the bottom row devices have the same length that differs from the top row. Further assume this command:

```
::erc:::setup_device_parameters -devices { MN(n) } \
    -total { total { W L } } -series
```

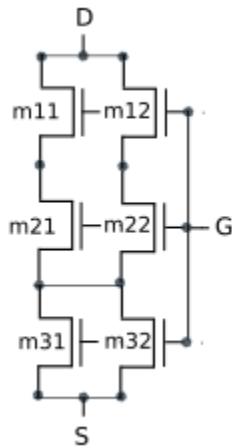
The output total property value for each device will be 2 because each row of devices has the same W and L properties. But if the first column of devices has a different width from the second column, then the output total property value will be 1 for each device.

Multi-Gate, Series-Parallel, and Dangling Pin Structures

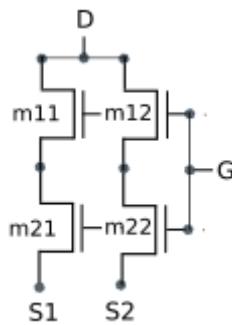
In some layouts, there can be different gate pin connections for parallel stacked devices, such as here:



Another configuration is series-parallel. In the following schematic, the upper four devices are in parallel stacks that are in series with the bottom two devices, which are in parallel with each other.



A third configuration is when terminal pins of stacked branches are not connected together, such as the S1 and S2 nets here:

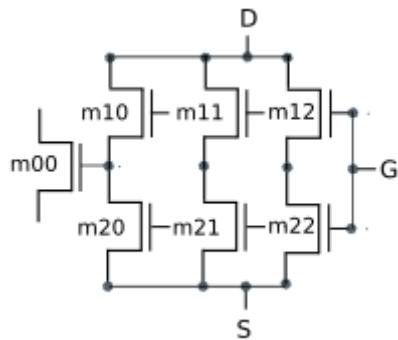


Ordinarily, these devices would not be considered to be connected in parallel. However, with the `erc::setup_device_parameters` -series option specified, they are.

When -series is used with -total to derive the number of total parallel connected devices in any of the preceding three cases, the output property value is 2 for each of the devices (2 per row), as with the post-layout schematic in [Figure 6-2](#).

Quasi-Stacked Devices

The following figure shows devices in a quasi-stacked configuration:



Devices M10 and M20 appear as though they may be stacked, but in reality, they are not because devices M00, M10, and M20 cannot be derived from a single device as in [Figure 6-1](#). The remaining devices are in parallel stacks.

When -series is used with -total to derive the number of total parallel connected devices, the output property value is 2 for M11, M12, M21, and M22, and 1 otherwise.

Property Definitions and Netlist Behavior for CCI Flows

Device properties appear in the extracted layout netlist used in LVS-H and in netlists produced by CCI. There are differences in how these properties are specified and the circumstances under which they appear in these netlists. This topic is of particular concern when LVS Push Devices SEPARATE PROPERTIES YES is specified in the rules (the PDSP flow).

For this discussion, “layout netlist” refers to the SPICE netlist extracted for use in LVS comparison. “CCI netlist” refers to the netlist produced by `dfm::write_spice_netlist` or its standard mode equivalent, LAYOUT NETLIST WRITE. If no distinction is made between these two types, then the term “netlist” refers to either type.

The term “PDSP NO” refers to when [LVS Push Devices](#) SEPARATE PROPERTIES NO is specified (the default) in the rule file. “PDSP YES” refers to when SEPARATE PROPERTIES YES is specified.

Device properties that appear in a netlist come from three sources: a [Device](#) statement, an [ERC TVF](#) function block, or a [Device Layer](#) ANNOTATE operation. The Device Layer ANNOTATE operation is principally of interest in an LVS Annotate Devices flow and is not discussed further here.

This is an abbreviated Device syntax showing the constructs that define properties:

```
DEVICE <element> <seed layer> <pin layers and names> ...
  '[' <property_specification> ']'      // properties defined here can appear
                                             // in any netlist
  '<'annotation_auxiliary_layer'>'
  '[' <device_annotation_program> ']' // properties defined here only appear
                                             // in a CCI netlist
```

Notice that only properties from a *property_specification* block can appear in a layout netlist. In a PDSP YES flow, certain properties from this block might not appear in the layout netlist, but do appear in a CCI netlist. Properties from a *device_annotation_program* (if any) can only appear in a CCI netlist.

Properties referenced in a ERC TVF function block can only appear in a CCI netlist, and they behave similarly to properties declared in a *device_annotation_program*.

Given the preceding ideas and terms, these definitions are given:

- **Layout netlist property** — A [Device](#) built-in (default) property or a property declared in a *property_specification* block's PROPERTY statement.

In a PDSP NO flow, all such properties appear in a layout netlist. In a PDSP YES flow, only the properties required for LVS comparison appear in the layout netlist. Properties defined in a *device_annotation_program* or an ERC TVF function are never layout netlist properties. (See the “Layout netlist properties” columns in the tables at the end of this section.)
- **LVS-actionable property** — A layout netlist property actively used in LVS comparison.

See [LVS Netlist Property](#) for a detailed definition of LVS-actionable. Such properties always appear in both layout and CCI netlists. Layout netlist properties that are not required for LVS comparison are called *non-LVS-actionable*. (Layout Netlist Property can be used to change the LVS-actionable status of a layout netlist property.)
- **Separated property** — In a PDSP YES flow, a non-LVS-actionable property.

In a PDSP NO flow, properties that are non-LVS-actionable appear in CCI netlists, but these are not called separated properties because they appear in the layout netlist, which has no separated properties.

- **CCI netlist property** — A property appearing in a CCI netlist. (See the “CCI netlist properties” columns in the tables at the end of this section.)

CCI netlist properties are in these classes:

LVS-actionable properties (always present).

Non-LVS-actionable properties (when present), except in a PDSP YES flow and `dfm::set_layout_netlist_options -separated_properties NO` is active. (This is the default and recommended setting in a PDSP YES flow.)

Properties defined in a *device_annotation_program* block’s PROPERTY statement or an ERC TVF function block. Properties in this class, when defined, only appear in CCI netlists when `dfm::set_layout_netlist_options -separated_properties YES` is active.

Consider the following rule file excerpt:

```

LVS PUSH DEVICES SEPARATE PROPERTIES YES // PDSP YES FLOW
MASK SVDB DIRECTORY svdb CCI QUERY

sd_cont_p = DFM PROPERTY sd cont [cont_count = COUNT(cont)]

DEVICE M gate gate(g) sd(s) sd(d)
  [ PROPERTY L, W, AS, AD
    L = PERIM_OUT(gate, sd) * 0.5
    W = PERIM_CO_OUT(gate, sd) * 0.5
    AS = AREA(s)
    AD = AREA(d) ]
<sd_cont_p>
  [ PROPERTY ccnt
    ccnt = DFM_NUM_VAL(sd_cont_p, cont_count) ]

TRACE PROPERTY M L L 0
TRACE PROPERTY M W W 0

```

The properties in the code are classified as follows in a PDSP YES flow.

Layout netlist: L, W

LVS actionable: L, W

Separated: AD, AS

CCI netlist: L, W (always)

AD, AS (except for PDSP YES and
`dfm::set_layout_netlist_options -separated_properties NO`)

ccnt (with -separated_properties YES)

The following tables summarize how device properties are specified and in which netlists they appear. The -separated_properties entries refer to the `dfm::set_layout_netlist_options` argument of that name.

Property Origin	PDSP NO (default)	
	Layout netlist properties	CCI netlist properties ¹
<i>property_specification</i> block	all	all
<i>device_annotation_program</i> block or ERC TVF function	none	-separated_properties NO: none -separated_properties YES: any

1. CCI netlists are not typically of concern outside of PDSP YES or LVS Annotate Devices flows.

Property Origin	PDSP YES	
	Layout netlist properties	CCI netlist properties
<i>property_specification</i> block	LVS-actionable	-separated_properties NO ¹ : LVS-actionable -separated_properties YES: all
<i>device_annotation_program</i> block or ERC TVF function	none	-separated_properties NO ¹ : none -separated_properties YES: any

1. The recommended setting in a PDSP YES flow.

Generating CCI Output Files

CCI files are used by third-party tools for post-LVS flows. This procedure shows how to generate the files.

If using pushdown separated properties ([LVS Push Devices SEPARATE PROPERTIES YES](#)), see “[Customized Files for Pushdown Separated Properties \(PDSP\) Flow](#)” on page 514 for a discussion of files generated for that flow.

Prerequisites

- A [Mask SVDB Directory](#) generated with the CCI option during a hierarchical comparison run.
- A Calibre Query Server and Calibre Connectivity Interface license.

Procedure

1. Create an *output* directory in your working directory.
2. In a text editor, create the following file:

```
#####
##### Generate the Annotated GDSII File
#####
### set the value used for the PROPATR record in GDSII BOUNDARY
### element to indicate node id
qs::set_agf_options -netprop_number 1

### set the value used for the PROPATR record in GDSII SREF
### element to indicate placement name
qs::set_agf_options -placeprop_number 2

### set the value used for the PROPATR record in GDSII BOUNDARY
### element to indicate device name
qs::set_agf_options -devprop_number 3

### set layer name to number and datatype mappings, if desired.
# qs::agf_map -remap RedLayer 2 1
# qs::agf_map -remap BlueSeed 3 1
# qs::agf_map -remap GreenPin 3 1

### write out the AGF MAP showing layer name to number mappings
qs::agf_map -write output/gds_map

### write the agf file in annotated GDSII format
qs::write_agf output/svdb.agf

#####
##### Generate the Layout Netlist
#####
### include trivial pins and empty cells in the layout netlist
dfm::set_layout_netlist_options -trivial_pins YES -empty_cells YES

### use only node numbers for netlists
dfm::set_layout_netlist_options -names NONE

### write the layout netlist
dfm::write_spice_netlist output/svdb.spi

#####
##### write the net to name mapping file (LNN)
#####
qs::write_layout_netlist_names output/svdb.lnn
```

```
#####
##### Generate Cross Reference Information
#####
### write the source and layout placement hierarchy files (sph,lph)
dfm::write_sph output/svdb.sph
dfm::write_lph output/svdb.lph

### write the net and instance cross reference
dfm::write_nxf output/svdb.lnxf -lnxf
dfm::write_ixf output/svdb.ixf

#####
##### Generate Port Table File
#####
### write the port table
qs::port_table -write output/svdb.ports

#####
##### Generate Rules Report
#####
### write a rule file settings file
qs::write_lvs_settings_report output/rules.report
qs::quit

#####
##### Optional. Equivalent script for use
##### with calibre -query standard shell.
#####
####
# AGF NETPROP NUMBER 1
# AGF PLACEPROP NUMBER 2
# AGF DEVPROP NUMBER 3
# AGF MAP RedLayer 2 1
# AGF MAP BlueSeed 3 1
# AGF MAP GreenPin 4 1
# RESPONSE FILE output/gds_map
# AGF MAP
# RESPONSE DIRECT
# AGF WRITE output/svdb.agf
# LAYOUT NETLIST TRIVIAL PINS YES
# LAYOUT NETLIST EMPTY CELLS YES
# LAYOUT NETLIST NAMES NONE
# LAYOUT NETLIST WRITE output/svdb.spi
# LAYOUT NAMETABLE WRITE output/svdb.lnn
# SOURCE HIERARCHY WRITE output/svdb.sph
# LAYOUT HIERARCHY WRITE output/svdb.lph
# NET XREF WRITE output/svdb.lnxf LNXF
# INSTANCE XREF WRITE output/svdb.ixf
# PORT TABLE WRITE output/svdb.ports
# LVS SETTINGS REPORT WRITE output/rules.report
# QUIT
```

3. Save the file as *cci_script*.
4. Run the following command:

```
calibre -qs -svdb svdb -exec cci_script
```

The generated files are saved in the *output* directory.

5. As an alternative to the script in Step 1, you can use this one for writing out device seed shapes having net ID properties or instance name properties and additional files:

```

# Define property numbers in annotated GDSII
qs::set_agf_options -netprop_number 5 -placeprop_number 6 \
-devprop_number 7

# Output seed polygons with either net ids or with instance names
qs::set_agf_options -seed_property DEVICE ORIGINAL

# Output annotated GDSII mapping file
qs::agf_map -remap ngate 4 1 -original
qs::agf_map -remap ngate 4 2 -device
qs::agf_map -remap pgate 4 3 -original
qs::agf_map -remap pgate 4 4 -device
qs::agf_map -write ./DEVICES.gds_map

# Output annotated GDSII file
qs::write_agf ./DEVICES.agds

# Output device table file containing descriptions of device layers
qs::device_table -write ./DEVICES.table

# Output ideal layout netlist
dfm::set_layout_netlist_options -trivial_pins YES -empty_cells YES \
-primitive_device_subckts YES -device_templates YES -hierarchy AGF \
-names NONE
dfm::write_spice_netlist ./DEVICES.spi

# Output net and device instance cross-referencing tables
qs::write_layout_netlist_names ./DEVICES.agf_lnn -expand_cells
dfm::write_nxf ./DEVICES.lnxf -lnxf
dfm::write_ixf ./DEVICES.ixf

# Output ports file
qs::port_table -cells -write ./DEVICES.ports

# Rules settings report file
qs::write_lvs_settings_report ./rules.report
qs::quit

```

6. Save the file as *cci_device_script* and run Step 4 using that filename.

Results

The files produced by the scripts are used in third-party tools for parasitic extraction and other purposes.

The *cci_script* causes these files to be written in the *output* directory.

Table 6-2. Files in output Directory

Name	Description
<i>gds_map</i>	qs::agf_map

Table 6-2. Files in output Directory (cont.)

Name	Description
<i>svdb.agds</i>	AGF layout file
<i>svdb.spi</i>	“Customized SPICE Netlist Format” on page 497
<i>svdb.lnn</i>	“Layout Netlist Names File Format” on page 502
<i>svdb.lph</i>	“Placement Hierarchy (LPH/SPH) File Format” on page 487
<i>svdb.sph</i>	
<i>svdb.ixf</i>	“Instance and Net Cross-Reference File Formats” on page 490
<i>svdb.lnxf</i>	
<i>svdb.ports</i>	“Port Table File Format” on page 504
<i>rules.report</i>	“qs::write_lvs_settings_report” on page 459

The *ccidevices_script* causes these files to be written in the current directory:

Table 6-3. Files in Current Directory

Name	Description
<i>DEVICES.gds_map</i>	Descriptions as in Table 6-2 for files with the same extensions.
<i>DEVICES.agds</i>	
<i>DEVICES.spi</i>	
<i>DEVICES.ixf</i>	
<i>DEVICES.ports</i>	
<i>rules.report</i>	
<i>DEVICES.table</i>	qs::device_table
<i>DEVICES.agf_lnn</i>	“Layout Netlist Names File Format” on page 502 This is generated using qs::write_layout_netlist_names with the EXPAND_CELLS option. (dfm::set_layout_netlist_options -hierarchy AGF is set.)
<i>DEVICES.lnxf</i>	“Layout Net Cross-Reference File (LNXF) Format” on page 495

Rule file query commands are frequently useful in a CCI flow. See [“Rule File Query Commands” on page 462](#) for more information.

Related Topics

[Property Definitions and Netlist Behavior for CCI Flows](#)

CCI Generated Files Reference

The Calibre Connectivity Interface generates a number of formatted files used by downstream simulation tools. These files are produced by Tcl shell and standard interface commands.

Cross-Reference System File Formats	486
Customized SPICE Netlist Format	497
Layout Netlist Names File Format	502
Port Table File Format	504
Reduction Data File Format	506
Separated Properties File Format	509

Cross-Reference System File Formats

This section describes how individual instances and nets are identified in the IXF, NXF, LPH, and SPH cross-reference files.

Untexted layout nets are identified by net number. Texted layout nets are identified with their user-specified names. Layout cell instances are identified with the letter X followed by the instance number, for example, X25. Layout device instances are identified with a letter designating their device type (for example, M, R, C, D, Q, or X), followed by the instance number. Source instances and nets are identified with names as they appear in the original source netlist.

This identification scheme is somewhat different from the one used in the AGF output or customized layout netlist outputs. In particular, note that all nets in the AGF file and the customized layout netlist are identified solely by their numbers. The correspondence between numbers and user-specified names for texted layout nets is provided in the Layout Netlist Name file ([qs::write_layout_netlist_names](#)).

Placement Hierarchy (LPH/SPH) File Format.....	487
Instance and Net Cross-Reference File Formats.....	490

Placement Hierarchy (LPH/SPH) File Format

Output for:[dfm::write_lph](#), [dfm::write_sph](#), and [HIERARCHY WRITE](#)

The layout placement hierarchy file (LPH) and the source placement hierarchy file (SPH) describe the hierarchical structure of the layout and source, respectively. They map cell instances to cell names in the layout and source. These files complement the IXF and NXF cross-reference files. The terms placement and instance are used interchangeably in this discussion.

Format

General format:

```
# SVDB: header_line
# SVDB: header_line
...
# SVDB: header_line
% cell_name pin_count
instance_identifier cell_or_device_name pin_count
instance_identifier cell_or_device_name pin_count
...
%cell_name pin_count
instance_identifier cell_or_device_name pin_count
instance_identifier cell_or_device_name pin_count
...
```

Each placement hierarchy file begins with a number of SVDB header lines indicated with a pound sign (#). The header format is described under “[SVDB Header Description](#)” on page 495. Following that there is a section for each cell in the hierarchy (all cells). The first line in each section consists of a percent (%) sign, a space, the cell name, a space, and finally the number of pins of the cell. The format is this:

```
% cell_name pin_count
```

Each of the following lines represents a cell instance (placement) or device instance within the current cell and consists of the instance identifier (local to the cell), a space, the name of the cell or device being instantiated, a space and the number of pins on the instance. The format is this:

```
instance_identifier cell_or_device_name pin_count
```

Note that no connectivity information is present other than pin counts. Also, cells that do not contain devices or placements of other cells (no vias) are not represented. Also, parameterized subcircuits in SPICE are flattened in this listing.

Parameters

- % cell_name
 - A % character followed by the name of a cell.

- pin_count
Integer specifying a pin count.
- instance_identifier
A string that defines a SPICE instance.
- cell_or_device_name
The name of the instance. For devices, this is given for M elements but not for others.

Examples

Example 1

```
# SVDB: ...
...
# SVDB: ...
% NAND 5
M1 MP 4
M2 MP 4
M3 MN 4
M4 MN 4
%PARMSUB 3
CO 2
C1 2
% TOP 0
X1 NAND 5
X2 NAND 5
X3/X0 PARMSUB 3
```

The [dfm::xref_xname](#) command configures the use of the “X” character on subcircuit calls in these tables. When dfm::xref_xname is set to -off, the listing for cell TOP becomes this:

```
% TOP 0
1 NAND 5
2 NAND 5
3/0 PARMSUB 3
```

Example 2 — Original model name (OMN) in the SPH and LPH files

The LPH and SPH files can be generated with original model name information. Original model names are model names of devices as they appear in an input SPICE netlist before any processing of *.EQUIV substitutions. This information is also available through the Instance Cross-Reference (IXF) file (see “[Original Model Name Information \(OMN\) in IXF File](#)” on page 492). This capability is enabled through the [Mask SVDB Directory OMN](#) option and the [dfm::write_lph](#) or [dfm::write_sph -omn](#) option.

For example, the rule file could have this statement:

```
MASK SVDB DIRECTORY "svdb" CCI OMN
```

To generate the information in the LPH/SPH files, use the -omn option as shown here:

```
dfm:::write_lph svdb.lph -omn
dfm:::write_sph svdb.sph -omn
```

This information is written into the placement hierarchy report in the following form:

```
instance_identifier cell_or_device_name number_of_pins [OMN=mod]
```

where mod is the original model name of the instance. The brackets are not part of the output and indicate an optional string. Here is an excerpt of an SPH file when -omn has been used:

```
# SVDB: Source Placement Hierarchy (sph) (File format 1)
...
% top 4
C4 C 3 OMN=CA
...
```

This capability is only available for hierarchical LVS.

Instance and Net Cross-Reference File Formats

Output for: `dfm::write_ixf`, `dfm::write_nxf`, `dfm::xref_xname`, **INSTANCE XREF WRITE**, **NET XREF WRITE**, and **LAYOUT NET XREF WRITE**.

Cross-reference files show correspondences between layout and source objects as determined by LVS.

There are three hierarchical cross-reference files:

IXF: Instance Cross-Reference File

NXF: Net Cross-Reference File

LNXF: Layout Net Cross-Reference File

Format

General form:

```
# SVDB: header_line
# SVDB: header_line
...
# SVDB: End of header.
% layout_cell layout_pin_count source_cell source_pin_count
layout_id layout_path source_id source_path
layout_id layout_path source_id source_path
...
% layout_cell layout_pin_count source_cell source_pin_count
layout_id layout_path source_id source_path
layout_id layout_path source_id source_path
...
...
```

Each hierarchical cross-reference file begins with a number of SVDB header lines indicated with a pound sign (#). The remainder of each file contains sections for individual correspondence cells or hcells. These cells exist in both layout and source netlists. There is one section for each hcell. Each hcell section begins with a percent sign (%) line specifying the layout cell name and pin count and the corresponding source cell name and pin count. This is followed by lines of corresponding layout and source elements in the cell.

The layout_path and source_path fields contain hierarchical pathnames rooted in the particular hcell that owns them. Note that the IXF and NXF files always have layout ID and source ID values of 0. More information about individual instance and net lines is provided in the following sections.

Instance Cross-Reference File (IXF) Format

This format is output by the `dfm::write_ixf` command. The instance cross-reference file contains matched instances. This includes device instances as well as cell instances. There is one line per instance in the following form:

```
layout_id layout_path source_id source_path [ SL | SS ] [ X ]
```

The layout_path and source_path fields identify corresponding layout and source instances. Instances are identified by hierarchical pathnames rooted in the particular hcell to which the current object belongs. An instance pathname consists of zero or more cell instance identifiers, followed by a device instance identifier, separated by “/” characters. The layout_id and source_id fields are for internal use and should be ignored. In particular, note that the layout_id field does not contain layout instance numbers. For example:

```
% ALU 25 ALU 25
0 X2/X3/M1 0 M1
0 D2 0 X3/X1/D8
```

Reduced devices (such as those created by series or parallel device reduction) are represented in the file by all their original constituents. The original devices appear in consecutive lines in the file, with the corresponding device from the other design repeated in each line. When a reduced layout device is matched to a reduced source device, then all the original layout devices are listed in consecutive lines on the left, with a representative original source device repeated on the right; all the other original source devices are listed in consecutive lines on the right, with a representative original layout device repeated on the left. The representative devices are chosen arbitrarily. Reduced layout devices (other than the representative) are indicated with the string SL, which stands for smashed layout. Reduced source devices (other than the representative) are indicated with the string SS, which stands for smashed source. For example:

```
0 R10 0 R1
0 R11 0 R1 SL
0 R10 0 R2 SS
0 R10 0 R3 SS
```

In this example, layout devices R10 and R11 were reduced to a single device and correspond to source devices R1, R2, and R3 that were also reduced to a single device. Layout device R10 and source device R1 were chosen as representatives.

Capacitor and resistor devices with swapped positive and negative pins are indicated with the trailing letter X. For example:

```
0 RR1 0 RRR1 X
0 CC1 0 CCC1 X
```

MOS devices with swapped source and drain pins are indicated with the trailing letter X. For example:

```
0 M10 0 M1 X
```

The X indicates that the source pin of the layout device corresponds to the drain pin of the source device and vice-versa. Lines that represent reduced devices (SL or SS) have X indicators as well, with respect to the two devices reported on that particular line. LVS logic gates are represented by the original transistors that form them. All matched transistors in the layout gate are listed, along with the corresponding transistors in the source gate.

The `dfm::xref_xname` command configures the use of the X character on subcircuit calls in these tables. When `dfm::xref_xname -off` is used, flattened instance names are changed. The following transistor (M) and User Defined (X) primitive devices normally represented as this:

```
0 X0/X0/M0 0 XA/XA/MA  
0 X0/X1/X1 0 XA/XB/XB
```

are instead represented as this:

```
0 0/0/M0 0 A/A/MA  
0 0/1/X1 0 A/B/XB
```

An hcell instance normally represented as this:

```
0 X0/X2/X2 0 XA/XC/XC
```

is instead represented as this:

```
0 0/2/2 0 A/C/C
```

If `dfm::xref_xname -box_cells` is used, the LVS Box cells that are matched in layout and source are annotated like this immediately after the SVDB header:

```
% layout_cell pin_count source_cell pin_count BOX
```

Original Model Name Information (OMN) in IXF File

Original model name (OMN) information can be generated through the IXF file. Note that, while the SPH and LPH files reflect the original design hierarchy, the IXF file reflects the design hierarchy after expansion of non-hcells. Because of this, communicating original model name information through the SPH and LPH files, rather than the IXF file, may save significant file space.

This information is generated similarly to the OMN information generated in the SPH/LPH files (see “[Example 2 — Original model name \(OMN\) in the SPH and LPH files](#)” on page 488).

The OMN option of the Mask SVDB Directory rule file statement must be used.

```
MASK SVDB DIRECTORY "svdb" OMN CCI
```

The `-layout_omn` or `-source_omn` options for `dfm::write_ixf` trigger the writing of the original model names in CCI. For example:

```
dfm::write_ixf svdb.ixf -layout_omn -source_omn
```

This information is written to the instance cross-reference report in the following form:

```
layout_id layout_path source_id source_path [SL|SS] [X] [LOMN=lmod]  
[SOMN=smod]
```

where lmod and smod are layout original model name and source original model name respectively. The brackets and vertical bar are syntax meta-characters and are not part of the output. For example:

```
# SVDB: Instance Cross Reference (ixvi) (File format 1)
...
0 X0/C0 0 XA/CA SOMN=CA LOMN=C1
...
```

Net Cross-Reference File (NXF) Format

This is the output format of the `dfm::write_nxf` command without the `-lnxf` option. The net cross-reference file contains matched nets. There is one line per net in the following form:

```
layout_id layout_path source_id source_path
```

The layout_path and source_path fields identify corresponding layout and source nets. Nets are identified by hierarchical pathnames rooted in the particular hcell to which this object belongs. A net pathname consists of zero or more cell instance identifiers, followed by a net identifier, separated by “/” characters. The layout_id and source_id fields are for internal use and should be ignored. In particular, note that the layout_id field does not contain layout net numbers. For example:

```
% ALU 25 ALU 25
0 X2/X3/7 0 SIG1
0 13 0 X3/X1/8
```

In certain situations, Calibre nmLVS may match several layout nets to one source net, or several source nets to one layout net, or a group of several layout nets (together) to a group of several source nets. This may occur, for example, after split gate reduction or device filtering, or when open or short circuit discrepancies exist. In these cases, a representative net is chosen for the layout side and a representative net is chosen for the source side. The representative pair appears in the net cross-reference file. In addition, each of the remaining layout nets appears with the source representative, and each of the remaining source nets appears with the layout representative. In the following example, layout nets 1, 2, and 3 were matched (as a group) to source nets n1, n2, and n3.

```
0 1 0 n1
0 1 0 n2
0 1 0 n3
0 2 0 n1
0 3 0 n1
```

Net 1 is matched to n1, n2, and n3. Net 2 is matched to n1 explicitly, but because n1 is matched to net 1, which is matched to n2 and n3, net 2 is implicitly matched to n2 and n3 as well. A similar argument holds for net 3.

Note that some nets are unmatched even when LVS is successful. Examples of nets that are never matched are nets internal to certain types of logic gates formed during comparison and

nets removed because of series device reduction. Unmatched nets do not appear in the net cross-reference file.

Tip

i If a many-to-one net correspondence is the result of an [LVS Filter](#) SHORT transformation, the selected net name for the shorted pins may be specified using [LVS Prefer Nets](#). Such a net name is listed first in the “many” list of the net correspondence.

The dfm::xref_xname command configures the use of the “X” character on expanded subcircuit calls in these tables. When dfm::xref_xname -off is used, expanded net names are changed as follows. Names like this:

```
0 X0/X0/0 0 XA/XA/1
0 X0/X1/N 0 XA/XB/N
```

are represented as this:

```
0 0/0/0 0 A/A/1
0 0/1/N 0 A/B/N
```

If the dfm::write_nxf -box_cells option is specified, [LVS Box](#) cells in the NXF file are reported in this form:

```
% layout_box_cell layout_port_count source_box_cell source_port_count BOX
0 port1 0 port1
0 port2 0 port2
0 port3 0 port3
```

Logic Injection and NXF File Contents

By default, logic injection ([LVS Inject Logic](#) YES) is performed during a Calibre nmLVS-H run. Logic injection is a netlist transformation just like parallel reduction. When a group of individual devices in parallel is replaced by a single reduced device, the internal nets of the parallel group are removed and are not individually compared. Such nets do not appear in the net cross-reference (NXF) file. The same behavior is true of injected instances.

CCI produces an exact record of the matches that it made during the LVS comparison phase. CCI does not report nets that were not compared. Transformations are controlled by the rule file and CCI reflects what the rule file contains.

Logic injection for the CCI and backannotation flow can cause nets to “disappear” because injected components have taken the place of individual devices. If the absence of the original nets is a problem, then the rule file used for CCI output should be set to LVS Inject Logic NO. However, for LVS comparison, YES is the preferred setting (and the default) for performance reasons.

Layout Net Cross-Reference File (LNXF) Format

This is the output format of the dfm::write_nxf -lnxf option. The format of the LNXF is similar to the NXF. Recall that the Net Cross-reference File (NXF) report discussed previously in this topic also maps layout nets to source nets, but takes its information only from an LVS comparison. The LNXF report maps layout nets from the persistent hierarchical database (PHDB) to source nets.

LNXF format contains a super-set of the cross-reference information that is provided by the default dfm::write_nxf command. Specifically, it adds cross-reference information for some nets that have a valid cross reference in one cell and extend upward in the hierarchy but are not considered relevant for LVS comparison purposes because they are not connected to devices outside of the cell.

When `dfm::set_layout_netlist_options -names LAYOUT` is used, the LNXF file handles nets that have been combined during comparison (due to various LVS comparison features like the LVS Filter SHORT keyword and open circuit error recovery) by showing representative nets paired together followed by the combined nets shown only with the representative net from the other design. For example, if layout nets L1, L2, and L3 are all combined and matched with similarly combined source nets S1 and S2, the relevant section of the LNXF file appears as follows:

```
0 L1 0 S1
0 L2 0 S1
0 L3 0 S1
0 L1 0 S2
```

Since L1 corresponds to S2 and S1, L2 and L3 (which correspond to S1) implicitly correspond to S2. This is the same as the NXF file format and can be desirable for certain flows.

If `dfm::set_layout_netlist_options -names NONE` is used, the LNXF file follows this convention only for combined *source nets*. Combined layout nets are not included in the customary grouping order, but repeated for the same group of source nets for each layout net, like this:

```
0 L1 0 S1
0 L1 0 S2
0 L2 0 S1
0 L2 0 S2
0 L3 0 S1
0 L3 0 S2
```

SVDB Header Description

The IXF, NXF, LPH and SPH files each begin with a number of SVDB header lines. The header identifies the type of file and the source of the information used to create the file. The header

consists of ten lines. Each line begins with the string # SVDB:. Here is an example of the SVDB header from a IXF file:

```
# SVDB: Instance Cross Reference (ixvi) (File format 1)
# SVDB: Layout Primary mix
# SVDB: Rules -0 play.rules Wed Dec 10 10:07:38 1997
# SVDB: GDSII -0 (nonv) (nonv)
# SVDB: SNL -0 (nonv) (nonv)
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
```

The first line in the header identifies the type of file and its format version, and is the only line which differs between files representing the same design. Here are the first lines from each of the four different files mentioned previously:

```
# SVDB: Layout Placement Hierarchy (lph) (File format 1)
# SVDB: Source Placement Hierarchy (sph) (File format 1)
# SVDB: Instance Cross Reference (ixvi) (File format 1)
# SVDB: Net Cross Reference (nxvi) (File format 1)
```

The second line in the header gives the name of the top (primary) cell of the design. The third, fourth, and fifth lines identify the rule file, layout GDSII file, and source netlist file used in creating the file. The entries on these lines are the type of file (Rules, GDSII, or SNL, respectively, the latter standing for source netlist) followed by a checksum for the file (or -0 if no checksum is present), the pathname to the file at the time the information was captured (or “(none)” if a pathname is not present), and a time stamp of the file which was used (or “(none)” if a time stamp is not present). A time stamp, when specified, is of the form:

week_day month day hh:mm:ss year

Parameters

- % layout_cell
 - A % character followed by the name of a layout cell.
- layout_pin_count
 - Integer specifying a pin count for the layout cell.
- source_cell
 - The name of a source cell corresponding to the layout_cell.
- source_pin_count
 - Integer specifying a pin count for the source cell.
- layout_id layout_path source_id source_path
- layout_id
 - An internally assigned integer to identify the layout object.

- layout_path
The netlist path to the layout object.
- source_id
An internally assigned integer to identify the source object.
- source_path
The netlist path to the source object.
- BOX
String that indicates matched LVS Box cells. This appears in IXF files when dfm::write_ixf -box_cells is specified. This appears in NXF files when dfm::write_nxf -box_cells is specified.
- LOMN=lmod
LOMN= followed by an original layout model name. This appears in IXF files when the dfm::write_ixf -layout_omn is specified.
- SOMN=smod
SOMN= followed by an original source model name. This appears in IXF files when the dfm::write_ixf -source_omn is specified.
- SL
String that indicates the layout device participated in reduction.
- SS
String that indicates the source device participated in reduction.
- X
String that indicates a device that has undergone pin swapping.

Related Topics

[Cross-Reference System File Commands](#)

Customized SPICE Netlist Format

[dfm::write_spice_netlist](#) and [LAYOUT NETLIST WRITE](#) output format.

CCI generates a custom SPICE netlist for use by downstream tools.

The custom netlist conforms to the extended Calibre SPICE netlist format described in the “[SPICE Format](#)” chapter of the *Calibre Verification User’s Manual*. Comment-coded fields appearing in such netlists are ignored by most SPICE simulators.

This file is used in conjunction with an AGF file and a [Layout Netlist Names File Format \(LNN\)](#) file.

By default, the netlist follows conventions used in netlists produced by calibre -spice. Various configuration options alter the structure of the netlist produced.

- Subcircuit definitions

.SUBCKT definitions in the layout netlist represent original cells in the input database or pseudo-cells created by Calibre hierarchical pre-processing. For original database cells, subcircuit names in the layout netlist are identical to cell names in the input database. An exception occurs when a cell name in the input database begins with a dollar sign (\$) character. Cell names that start with \$ usually have the prefix ___. In general, to avoid conflicts with database names, the prefix consists of N underscores, where N is one larger than the number of leading underscores in any database cell named ___*\$\$ (that is, any database cell whose name begins with at least two underscores followed by a \$). This prefix is necessary to make those cell names valid in SPICE. Recall that a leading \$ denotes a comment in SPICE.

Subcircuit definitions indicate which nets in the cell serve as pins of the cell. All nets in a cell that make connections to nets outside of the cell appear as pins in the .SUBCKT line for the cell.

For example:

```
.SUBCKT ABC 1 2
R1 1 2 50
.ENDS
```

describes cell ABC with two pins: 1, 2. These numbers are net numbers in the cell.

- Subcircuit calls

Subcircuit calls are represented by standard SPICE subcircuit call notation X. Comments provide some additional information. For example, the line:

```
X3 5 7 ABC $T=-375 13750 1 0 $X=2250 $Y=2000
```

describes a placement of cell ABC with nets 5, and 7 connected to the first and second pins of ABC respectively. The comment field \$T represents the transform of the placement: X translation, Y translation, reflection (0 or 1, indicating false or true, with the X-axis being the axis of symmetry) and rotation (0, 90, 180, or 270 degrees, counter-clockwise); \$X and \$Y represent coordinates of the placement.

- Devices

When possible, devices in the layout netlist are represented with regular SPICE elements (M, R, C, D, Q, and so on). In other cases (for example, when the device is not a standard SPICE element), devices in the layout netlist are represented with subcircuit calls. In the latter case, the layout netlist contains an empty subcircuit definition describing the device. Device lines provide the type of device, pin connections, model name, and parameters or properties. Location of the device is presented as an X Y coordinate pair in database units encoded as a comment at the end of the device.

For example, this line:

```
M2 4 5 6 7 p L=1 W=2 $X=-6000 $Y=1000
```

describes a MOS device instance of type P (in SVRF, DEVice MP) with drain, gate, source and bulk connected to nets 4, 5, 6, and 7 respectively and W/L as specified at the coordinates (-6000, 1000) in the current cell. This line:

```
X2 7 8 9 10 11 Q3E $X=1000 $Y=1000
```

describes a device instance of type Q3E with five pins. In this case, the netlist also contains a subcircuit definition such as this:

```
.SUBCKT Q3E C B E1 E2 E3
.ENDS
```

Actual seed and pin device shapes that make up the device may occur at various levels above and below the device coordinates. Their location is governed by the Calibre nmLVS circuit extraction seed promotion functionality. The device coordinates appear at a vertex of the seed shape after translation into the current cell. Device coordinates can also be configured to appear at the center of the seed shape when the coordinates of the center lie on the seed shape using the [dfm::set_layout_netlist_options -device_location](#) command.

Some built-in devices make use of optional substrate pins defined by Calibre. They are specified as:

```
R0 1 2 50 $SUB=3
```

where R0 is a resistor with terminals on nets 1 and 2 and substrate pin on net 3. This feature is controlled with the [LVS Netlist Comment Coded Substrate](#) specification statement. If LVS Netlist Comment Coded Substrate NO is specified in the rule file, the resistor is output as a call to a subcircuit:

```
.SUBCKT R pos neg sub
.ENDS
```

```
X0 1 2 3 r=50
```

Devices included in the netlist are affected by the [dfm::create_filter -device_ids](#) or [-device_names](#) setting currently active in the Query Server. Only devices of FILTERED types are included in the netlist.

- Nets

The nets in the layout netlist can be specified in one of three ways:

- Using a combination of layout net names available from text in the layout database (default) and net numbers for untexted nets.
- Pure net numbers.

- Source names where source names are available, layout names prefixed by a string where matching names are not available. The prefix is typically `_Layout_`, but if there are source names that begin with `_Layout_`, additional `_` characters are added to the prefix to ensure that it is unique. When devices are reduced through series or parallel reduction, the first device in the group is named with the corresponding source name. Other devices in the group are named with a name like `_Layout_<layout_name>_Source_<source_name>`.
- Name format is controlled with the `dfm::set_layout_netlist_options -names` command.
- Net numbers

Net numbers are determined arbitrarily by Calibre during circuit extraction. The combination of subcircuit calls and subcircuit definitions in the layout netlist describes how electrical nets traverse levels of hierarchy in the design.

- Expanded placements

The `dfm::set_layout_netlist_options -hierarchy` command can be used to expand cells in place within the netlist. When cells are expanded, the names of cell placements are of the form $XX_i/[X_j\dots]/X_k$.

Note that all device and cell placement names have exactly one extra letter at the front when compared with names in the unexpanded netlist (`dfm::set_layout_netlist_options -hierarchy ALL`). This extra letter ensures that the generated netlist is compatible with SPICE readers and that the generated names can all be easily mapped to regular Calibre hierarchical names by simply removing the first letter. The extra letter is applied within all cells (those that have expanded placements as well as those that do not).

For example, consider the following non-flattened circuit:

```
.SUBCKT CELL_1 14 13
M0 11 IN 10 13 p L=1e-06 W=1.8e-05 $X=3000 $Y=9500
M1 9 IN 8 14 n L=1e-06 W=1.8e-05 $X=3000 $Y=-10500
X2 23 13 24 14 26 CELL_2
.ENDS

.SUBCKT CELL_2 I1 13 OUT 14 I2
M0 13 I1 OUT 13 p L=1e-06 W=1e-05 $X=-20000 $Y=20500
M1 13 I2 OUT 13 p L=1e-06 W=1e-05 $X=0 $Y=24500
M2 6 I2 14 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-50500
M3 OUT I1 6 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-29500
.ENDS

.SUBCKT TOP 14 13
M0 13 4 5 13 p L=3e-06 W=1.3e-05 $X=47000 $Y=-2000
M1 3 2 14 14 n L=1e-06 W=9e-06 $X=-52000 $Y=-35000
M2 5 4 14 14 n L=2e-06 W=1e-05 $X=49000 $Y=-42000
X3 14 13 CELL_1 $T=-104000 -58000 0 0 $X=-108000
$Y=-76000
.ENDS
```

when CELL_1 is expanded, the circuit is represented as this:

```
.SUBCKT CELL_2 I1 13 OUT 14 I2
MM0 13 I1 OUT 13 p L=1e-06 W=1e-05 $X=-20000 $Y=20500
MM1 13 I2 OUT 13 p L=1e-06 W=1e-05 $X=0 $Y=24500
MM2 6 I2 14 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-50500
MM3 OUT I1 6 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-29500
.ENDS

SUBCKT TOP 14 13
MM0 13 4 5 13 p L=3e-06 W=1.3e-05 $X=47000 $Y=-2000
MM1 3 2 14 14 n L=1e-06 W=9e-06 $X=-52000 $Y=-35000
MM2 5 4 14 14 n L=2e-06 W=1e-05 $X=49000 $Y=-42000
MX3/M0 X3/11 X3/1 X3/10 6 p L=1e-06 W=1.8e-05 $X=3000 $Y=9500
MX3/M1 X3/9 X3/1 X3/8 1 n L=1e-06 W=1.8e-05 $X=3000 $Y=-10500
XX3/X2 X3/23 13 X3/24 14 X3/26 CELL_2 $T=0 -22000 0 0 $X=-40000
$Y=-102000
.ENDS
```

See the [dfm::set_layout_netlist_options](#)-hierarchy command for more details on types of flattening available.

Layout Netlist Names File Format

Output for: [qs::write_layout_netlist_names](#) and [LAYOUT NAMETABLE WRITE](#).

Specifies a correspondence between net number annotations in an AGF file and user-names in the layout netlist.

The Layout Netlist Name file is required to translate layout net numbers (of texted layout nets) into net names for lookup in the hierarchical Net Cross-Reference file (.nxr suffix). See “[Layout Netlist Names File Command](#)” on page 607 for additional details.

Format

General form:

```
# SVDB: header_line
# SVDB: header_line
...
# SVDB: header_line
% cell_name pin_count
number name
number name
...
...
```

The file begins with a number of SVDB header lines indicated with pound characters (#). Following that, there is a section for each cell in the layout hierarchy (not just corresponding cells or hcells, but all cells). The first line in each section consists of a percent (%) sign, a space, the cell name, a space, and finally the number of pins of the cell. For non-standard SPICE cell names (such as those starting with a dollar sign “\$”, and so forth), the SPICE name (with prefix underscores) is included after the pin count. The format is this for regular cells:

```
% cell_name pin_count
```

and this for cell names that are not SPICE compatible:

```
% cell_name pin_count [spice_compatible_cell_name]
```

Each of the following lines represents a net within the current cell and consists of the net number (local to the cell), a space, and the user-specified net name. The format is this:

```
number name
```

Only texted nets have entries in the Layout Netlist Name file.

Parameters

- % cell_name
 - A % character followed by the name of a cell.

- pin_count
Integer specifying a pin count.
- spice_compatible_cell_name
Alternate name of a cell for names that start with \$. This name complies with SPICE naming conventions as \$ indicates a comment.
- number
An integer corresponding to a net number. The net number is cell-specific.
- name
A name of a net.

Examples

```
# SVDB: ....  
....  
# SVDB: ....  
% NAND 5  
4 I1  
5 I2  
6 OUT  
% TOP 0  
2 VDD  
3 VSS  
17 CLOCK  
% $VIA2 1 [__$VIA2]
```

Related Topics

[Customized SPICE Netlist Format](#)

Port Table File Format

Output for: [qs::port_table -write](#) and [PORT TABLE WRITE](#).

Shows locations and layers of port objects.

Format

The file contains one line for each port. (Unattached ports are not output.) Each line has the following fields:

```
port-name node-number node-name location layer-attached [cell-name]
```

Parameters

- port-name

Layout name of the port object, for example the text object string when using [Port Layer Text](#), or <UNNAMED> for [Port Layer Polygon](#) when [PORT TABLE NAME POLYGON PORTS NO](#) or [qs::port_table -name_polygon_ports NO](#) is used. The YES keyword for either of these latter commands causes polygon ports to have the names of their underlying nets.

- node-number

Layout node number to which the port is connected.

- node-name

Layout node name to which the port is connected, or layout node number if the node is unnamed.

- location

Form: x y; in database units. This is the location of the database text object when using Port Layer Text, or a vertex on the port polygon marker when using Port Layer Polygon.

- layer-attached

Layer of the polygon to which the port is attached. It is the rule file layer name or rule file layer number if the layer is unnamed. This layer appears in a [Connect](#) or [Sconnect](#) operation by default. If [qs::port_table -db_connectivity](#) or [PORT TABLE WRITE DB_CONNECTIVITY](#) is used, then the layer can appear in an [LVS DB Connectivity Layer](#) statement.

- cell-name

Cell name field is present only when the file is produced by the [qs::port_table](#) command with the [-cells](#) option or the [PORT TABLE CELLS WRITE](#) command. It specifies the name of the cell in which the port resides. The coordinate space context is the top-level cell.

Examples

qs::port_table produces the following output:

```
CONF3 5 CONF -98000 -90000 metal
CONF3 5 5 -98000 -90000 metal
<UNNAMED> 5 5 -98000 -90000 metal
CONF 5 CONF -98000 -90000 17
```

qs::port_table -cells produces the following output:

```
CONF3 5 CONF -98000 -90000 metal cellA
CONF3 5 5 -98000 -90000 metal cellA
<UNNAMED> 5 5 -98000 -90000 metal cellB
CONF 5 CONF -98000 -90000 17 CHIP
```

Reduction Data File Format

Output for: [dfm::write_reduction_data](#) and [REDUCTION DATA WRITE](#)

The reduction data file contains information about the types of reduction transformations that have occurred in the layout and source designs during LVS comparison.

The records pertaining to each cell are written in an order that preserves the logical order of reductions among related objects. For any set S of records within a cell's section, if all records in S refer to object X, those records are ordered such that X is the survivor in all records except the last, and in that last record, X may or may not be the survivor. In other words, if you are traversing the records for a cell in order from first to last, once an object appears as deleted in a record, thereafter that object does not appear in any record. This ordering allows the following algorithm to be used to find the ultimate survivor for object X in a given cell:

Let “X” be the survivor. For each cell reduction record from first to last, if the deleted object path for a record is the survivor, then set the survivor to be a surviving object path for the record.

It is necessary to examine all of a cell's records in order to find the ultimate survivor for any given object in the cell. Accordingly, the algorithm above functions differently if the data has been filtered to include only records with a specified reason code.

There are two supported output formats: hierarchical and flat. In the hierarchical format, one section is written to the output file for each hcell correspondence from the LVS-H run. In the flat format, the output file contains one section. If the LVS-H run that logged the data was a flat run, or if the whole design was flattened during the LVS-H run, the flat output format is necessarily produced.

When flattening is performed, each cell's own reduction records precede records for lower-level cells flattened into it. This applies only to flattening performed by reduction data commands. When flattening occurs during the LVS-H run, the logical order of reductions in the resulting data is preserved. Ordering considerations do not apply to the flattening done by reduction data commands because reduction chains are constrained by hcell boundaries. That is, all reduction operations pertaining to a given object must occur within the hcell containing that object, and therefore no requirements are imposed on the ordering of an hcell's reduction records with respect to records from any other hcell.

Format

The output file begins with the customary SVDB header for Query Server files. The sections of reduction data are grouped by corresponding cells in layout and source, and these sections are repeated as often as necessary. Each section is preceded by a percent (%) character. Cell correspondence lines are always present in the report for every hcell match regardless of whether any reduction took place.

```
# SVDB: <header_line>
# SVDB: <header_line>
. .
# SVDB: End of header.
% <layout_cell> <layout_pin_count> <source_cell> <source_pin_count>
<deleted_object_path> <surviving_object_path> <reason_code>
<deleted_object_path> <surviving_object_path> <reason_code>
. . .
```

Flat files have the same format as hierarchical files but with the added stipulation that only a single section is produced—for the top-level cell correspondence—and any reduction data from subordinate hcells are flattened into this top-level section.

Parameters

- <layout_cell>
The name of a layout hcell that corresponds to the <source_cell>.
- <layout_pin_count>
The number of ports for the <layout_cell>.
- <source_cell>
The name of a source hcell that corresponds to the <layout_cell>.
- <source_pin_count>
The number of ports for the <source_cell>.
- <deleted_object_path>
A path of the net or instance that was deleted in the reduction that the current report line pertains to. This path is relative to the cell of the given design (source or layout) identified in the nearest prior hcell correspondence line (the nearest prior line beginning with a percent sign (%)). This parameter only appears if a reduction occurred.
- <surviving_object_path>
A path of the net or instance that survived in the reduction that the current report line pertains to. This path is relative to the same cell that the <deleted_object_path> on the same line is relative to. This parameter only appears if a reduction occurred.
- <reason_code>
A keyword identifying the type of reduction process that resulted in the reduction in question. This is one of the following:
 - DEEPSHORT — Deep short resolution. This also encompasses the shorting of nets due to device filtering, as well as propagation of shorts due to *.CONNECT and its equivalent SPICE directives.
 - HIGHSHORT — High short resolution.
 - PARALLEL — Parallel device reduction.

- PORTFLATTEN — The reduction occurred due to flattening. The deleted object is a port net in the inner cell, and the surviving object is the corresponding pin net in the outer cell. Not all such reductions are logged.
- SEMISERIES — [LVS Reduce Semi Series MOS](#) reduction.
- SEN — [LVS Short Equivalent Nodes](#) reduction.
- SERIES — Series device reduction.
- SPLITGATE — [LVS Reduce Split Gates](#) reduction.

This parameter only appears if a reduction occurred.

Examples

This shows a report excerpt where the cells either have none of the requested reductions or series device reductions.

```
# SVDB: LVS Reduction Data (lrdf) (File format 1)
# SVDB: Layout Primary top
# SVDB: Rules -0 lvs.header Fri Sep  8 16:06:25 2017
# SVDB: GDSII -0 /project/design/fullchip.oas Thu Feb  6 12:21:40 2014
# SVDB: SNL -0 ../source.net
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
% ADC_5bit_comp_sc_5 37 ADC_5bit_comp_sc_5 35
% ADC_5bit_sc_5 9 ADC_5bit_sc_5 7
% AND2X1 5 AND2X1 5
% AND2X2 5 AND2X2 5
% AOI21X1 6 AOI21X1 6
...
% res_network 11 res_network 11
X86/R0 X74/R0 SERIES
X83/R0 X74/R0 SERIES
X84/R0 X74/R0 SERIES
X89/R0 X74/R0 SERIES
X77/R0 X74/R0 SERIES
X80/R0 X74/R0 SERIES
X79/R0 X74/R0 SERIES
X78/R0 X75/R0 SERIES
% Switch_new 7 Switch_new 7
% top 10 top 6
```

Separated Properties File Format

Output for: `qs::write_separated_properties` and **LAYOUT SEPARATED PROPERTIES WRITE**.

The separated properties file is generated for the PDSP flow and shows separated properties for affected devices.

Built-in devices are recognized according to the usual device recognition heuristics. A generic, user-defined, MOS device can only be formed during device recognition from a **Device** statement that satisfies all of the following conditions:

- The Device statement must contain one of each of the following pins, in any order: G, S, D.
- The S and D pins must be swappable.
- There must be only one swap group in the Device statement; that is, no other pins besides S/D can be swappable with any other pin.
- There can be any number of additional pins specified in the Device statement.
- There can be any number of auxiliary layers specified.
- The Device statement can use any user-defined element name and any model name (model name is optional).

The following Device statement is interpreted as a generic user-defined MOS device. The appropriate swapping is performed during device pushdown:

```
DEVICE userMOS seedLayer(G) SD(S) SD(D) bulkLayer(B)
```

Properties that are not separated, but are used in LVS comparison, appear in a comment-coded section like this in the file:

```
* These lvs-actionable properties, indexed by DEVICE element(model) will
be kept in the netlist:
*
* KEEP_PROP_BEGIN
* KEEP_PROP mn L W
* KEEP_PROP mp L W
* KEEP_PROP_END
```

These properties appear in the netlist generated by `dfm::write_spice_netlist`.

Pin Swapping Behaviors

For built-in MOS devices, Calibre nmLVS swaps \$PIN_XY coordinate values for source and drain pins when devices have their source/drain pins swapped during device pushdown. (Recall that Calibre nmLVS also swaps known built-in property values such as AS/AD, PS/PD, if present, when the related source/drain pins are swapped.)

For generic user-defined MOS devices, built-in swappable properties (AS/AD, PS/PD and so on)—the same set supported for built-in MOS—are supported. When S/D pins are swapped during pushdown, these property values are swapped just as with built-in MOS devices.

LVS also detects other generic devices. Fully generic swappable user devices can only be formed during Calibre device recognition from a Device statement that satisfies all of the following conditions:

- The Device statement must contain exactly two pins of any name that are swappable with one another.
- There must be only one swap group in the Device statement; that is, no other pins besides the pair satisfying the first requirement can be swappable with any other pin.
- There can be any number of additional pins specified in the Device statement.
- There can be any number of auxiliary layers specified.
- The Device statement can use any user-defined element name and any model name (model name is optional).

The following Device statement is interpreted as a fully generic swappable user device. The appropriate swapping is performed during device pushdown:

```
DEVICE userR seedLayer pinLayer(POS) pinLayer(NEG)
```

Format

Pin locations are written in a format similar to the following. Note the \$PIN_XY field:

```
<device><n> <properties> $PIN_XY=<x1>,<y1>,<x2>,<y2>,<x3>,<y3>,<x4>,<y4>  
$X=<x> $Y=<y> $D=<n>
```

Parameters

- device
The name of a device.
- n
A non-negative integer signifying the ordinal index of the device.
- properties
Property names and values computed for the device. Properties are represented using similar conventions as netlists produced using calibre -spice.
- \$PIN_XY=<x1>,<y1>,<x2>,<y2>,...,<xN>,<yN>
\$PIN_XY= followed by comma-separated integers, which represent the vertices of pins (possibly swapped). The order of the coordinates follows the pins of the corresponding netlist device element.

- \$X=<x>
\$X= followed by an integer representing the x-coordinate of the device body. By default, it is the lower-left vertex. It can also be the center if [dfm::set_layout_netlist_options -device_location CENTER](#) is specified.
- \$Y=<y>
\$Y= followed by an integer representing the y-coordinate of the device body. By default, it is the lower-left vertex. It can also be the center if [-device_location CENTER](#) is specified.
- \$D=<n>
\$D= followed by a non-negative integer. The integer gives the ordinal of the Device statement that was used in the rule file. A 0 means the first Device statement.

Examples

```
* Calibre Connectivity Interface Layout netlist
* Calibre VERSION: v2014.1 Mon Jan 6 12:44:50 PST 2014
* Type: SPICE
* Hierarchy: AGF
* Empty cells: YES
* Trivial pins: YES
* Device pin locations: NO
* String Properties: YES
* Primitive device subckts: NO
* HSPICE capacitors/resistors: NO
* HSPICE user devices: NO
* Comment Properties: NO
* Net name type: NONE
* Instance name type: LAYOUT
* Device location: CENTER
* Filter devices: (all)
* Device lowercase: NO
* Device templates: YES
* Separated properties: NO
* Calibre pushdown separate-properties (back-annotation) data
*****
*
* These lvs-actionable properties, indexed by DEVICE element(model) will
be kept in the netlist:
*
* KEEP_PROP_BEGIN
* KEEP_PROP mn L W
* KEEP_PROP mp L W
* KEEP_PROP_END
*
.SUBCKT nand
M0 as=1.075e-10 ad=1.25e-10 ps=5.075e-05 pd=5.25e-05
$PIN_XY=6875,61000,5125,61000,5125,61000,5125,61000 $X=6000 $Y=61000 $D=1
M1 as=1.25e-10 ad=1.075e-10 ps=5.25e-05 pd=5.075e-05
$PIN_XY=14875,61000,13125,61000,13125,61000,13125,61000 $X=14000 $Y=61000
$D=1
M2 as=8.4375e-11 ad=1.0125e-10 ps=4.125e-05 pd=4.35e-05
$PIN_XY=6625,21500,5375,21500,5375,21500,5375,21500 $X=6000 $Y=21500 $D=0
.ENDS
*****
.SUBCKT CellA
* nothing transcribed
.ENDS
*****
.SUBCKT inv
M0 as=1.075e-10 ad=1.075e-10 ps=5.075e-05 pd=5.075e-05
$PIN_XY=6875,61000,5125,61000,5125,61000,5125,61000 $X=6000 $Y=61000 $D=1
M1 as=8.4375e-11 ad=8.4375e-11 ps=4.125e-05 pd=4.125e-05
$PIN_XY=6625,21500,5375,21500,5375,21500,5375,21500 $X=6000 $Y=21500 $D=0
.ENDS
*****
.SUBCKT CellB
* nothing transcribed
.ENDS
*****
.SUBCKT TOPCELL
```

```
X0/X0/M3 as=1.0125e-10 ad=1.179e-10 ps=4.35e-05 pd=4.575e-05
$PIN_XY=14625,21500,13375,21500,13375,21500,13375,21500 $X=14000 $Y=21500
$D=0
.ENDS
*****
```

Customized Files for Pushdown Separated Properties (PDSP) Flow

Pushdown separated properties is an LVS flow that allows for efficient handling of device properties used by third-party parasitic extraction tools. These properties are not typically used in LVS comparison, and they do not appear in the extracted layout netlist. CCI employs a number of commands specific to this flow.

The separated properties saved in the [Mask SVDB Directory](#) can be used to write a customized layout netlist or to write a data file as a part of property backannotation. These commands assume the use of [LVS Push Devices](#) SEPARATE PROPERTIES YES in the rule file.

The principal CCI command of interest in the PDSP flow is [qs::write_separated_properties](#). It generates a file containing netlist elements with separated properties. The format is discussed under “[Separated Properties File Format](#).” The procedure for calculating PDSP properties is discussed under “[Writing PDSP Properties in CCI](#)” on page 515.

Commands to configure and write a backannotated netlist using CCI netlist writer include the following.

Table 6-4. PDSP Netlist Configuration Commands

Command or Option	Description
ERC TVF Commands	Support special device parameter calculations that are unavailable in LVS.
dfm::create_filter -device_ids or -device_names	Controls which devices are queried.
dfm::set_layout_netlist_options Configures the dfm::write_spice_netlist output, called the CCI netlist for brevity. Options follow:	
-annotated_devices	Specifies that devices annotated with DFM properties are used for writing device properties in the CCI netlist.
-cell_prefix	Specifies a cell prefix for the CCI netlist.
-device_location	Controls device location information in the CCI netlist.
-device_templates	Controls output of device template information in the CCI netlist.
-empty_cells	Controls writing of subcircuits for device-less cells.
-hierarchy	Specifies the type of hierarchy to use in the CCI netlist.

Table 6-4. PDSP Netlist Configuration Commands (cont.)

Command or Option	Description
-hier_separator	Changes the hierarchical separator character used in various files.
-names	Defines the origin of the net names used in the CCI netlist.
-primitive_device_subckts	Controls writing of primitive subcircuits for user-defined devices.
-separated_properties	Controls the output of separated properties to the CCI netlist.
-string_properties	Controls the output of string properties.
-trivial_pins	Controls the writing of trivial pins to the CCI netlist.
dfm::write_spice_netlist	Writes the CCI netlist.
qs::write_separated_properties	Causes separated properties to be written to a file.

Consult your third-party tool vendor's documentation for specific settings of these commands.

Rule file query commands are frequently desirable in a PDSP flow. See “[Rule File Query Commands](#)” on page 462 for more information.

Writing PDSP Properties in CCI

Pushdown Separated Properties Flow

This procedure describes a basic configuration of a rule file and CCI script to get PDSP properties for downstream simulation tools.

Prerequisites

- Knowledge of when to use [LVS Push Devices SEPARATE PROPERTIES YES](#).
- Knowledge of which device properties in your rules are used for simulation purposes but not LVS comparison. Typically, these are properties that do not appear in [Trace Property](#) statements.
- Calibre nmLVS-H, Calibre Query Server, and Calibre Connectivity Interface licenses.

Procedure

1. Ensure these statements are in your rules:

```
LVS PUSH DEVICES SEPARATE PROPERTIES YES
MASK SVDB DIRECTORY svdb CCI QUERY
```

2. (Required if the device extraction rules do not contain these derivations already.)
As needed, derive DFM Property layers that contain all simulation properties you are interested in, similar to this:

```
nsd_prop = DFM PROPERTY nsd [ A = AREA( nsd ) ] [ P = PERIM( nsd ) ]  
psd_prop = DFM PROPERTY psd [ A = AREA( psd ) ] [ P = PERIM( psd ) ]
```

These layers must interact with pin layers for the devices. Alternatively, the properties can be assigned to copies of device seed shapes, like this:

```
ngate_p = DFM PROPERTY ngate <pin_or_aux_layer> ...  
[ simprop = ... ] ...  
pgate_p = DFM PROPERTY pgate <pin_or_aux_layer> ...  
[ simprop = ... ] ...
```

3. Write Device statements in your rules that use the layers from Step 2 as annotation auxiliary layers. Include the standard property computation block and the device annotation program block as follows:

```
device mn(n) ngate ngate(g) nsd(s) nsd(d) pwell(b)  
[ PROPERTY L, W  
  L = perimeter_outside(ngate,nsd) * 0.5  
  W = perimeter_co(ngate,nsd) * 0.5  
]  
<nsd_prop>  
[  
  PROPERTY AS, AD, PS, PD  
  AS = DFM_NUM_VAL(nsd_prop, A, s) * UNIT_LENGTH() * UNIT_LENGTH()  
  AD = DFM_NUM_VAL(nsd_prop, A, d) * UNIT_LENGTH() * UNIT_LENGTH()  
  PS = DFM_NUM_VAL(nsd_prop, P, s) * UNIT_LENGTH()  
  PD = DFM_NUM_VAL(nsd_prop, P, d) * UNIT_LENGTH()  
]  
  
device mp(p) pgate pgate(g) psd(s) psd(d) nwell(b)  
[ PROPERTY L, W  
  L = perimeter_outside(pgate,psd) * 0.5  
  W = perimeter_co(pgate,psd) * 0.5  
]  
<psd_prop>  
[  
  PROPERTY AS, AD, PS, PD  
  AS = DFM_NUM_VAL(psd_prop, A, s) * UNIT_LENGTH() * UNIT_LENGTH()  
  AD = DFM_NUM_VAL(psd_prop, A, d) * UNIT_LENGTH() * UNIT_LENGTH()  
  PS = DFM_NUM_VAL(psd_prop, P, s) * UNIT_LENGTH()  
  PD = DFM_NUM_VAL(psd_prop, P, d) * UNIT_LENGTH()
```

When writing these, ensure the device type/subtype combinations are unique to these Device statements. Also, ensure that no other Device statements have annotation program blocks that share the same seed layers as these statements.

An alternative to using UNIT_LENGTH() for scaling is to set [Unit Length 1](#) in the LVS rules.

4. Run circuit extraction:

```
calibre -spice layout.sp -turbo rules
```

If there are any problems indicated in the transcript or circuit extraction report, fix them and re-run this step.

5. (Tcl shell interface.) Write a PDSP CCI script, as follows:

```
dfm::set_layout_netlist_options -device_location CENTER \
    -trivial_pins YES -empty_cells YES -names NONE \
    -primitive_device_subckts NO -separated_properties NO \
    -device_templates YES -hierarchy AGF
qs::write_separated_properties cci_props.data
dfm::write_spice_netlist cci.netlist
qs::quit
```

Save it as *pdsp_script.tcl*.

6. Run the Query Server Tcl shell:

```
calibre -qs -svdb svdb -turbo -exec pdsp_script.tcl
```

If there are any runtime errors, fix the problems and re-run this step.

7. (Standard interface.) Write a PDSP CCI script, as follows:

```
LAYOUT NETLIST DEVICE LOCATION CENTER
LAYOUT NETLIST TRIVIAL PINS YES
LAYOUT NETLIST EMPTY CELLS YES
LAYOUT NETLIST NAMES NONE
LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS NO
LAYOUT NETLIST SEPARATED PROPERTIES NO
LAYOUT NETLIST DEVICE TEMPLATES YES
LAYOUT NETLIST HIERARCHY AGF
LAYOUT SEPARATED PROPERTIES WRITE cci_props.data
LAYOUT NETLIST WRITE cci.netlist
TERMINATE
```

Save it as *pdsp_script*.

8. (Standard interface.) Run the Query Server:

```
calibre -query svdb < pdsp_script
```

If there are any ERROR or NOK responses, fix the problems and re-run this step.

Results

The *cci.netlist* file contains the customized SPICE netlist for downstream tool use. It lacks the PDSP properties. The *cci_props.data* file contains a netlist with the PDSP properties.

Related Topics

[Separated Properties File Format](#)

[Customized Files for Pushdown Separated Properties \(PDSP\) Flow](#)

[ERC TVF Commands](#)

[Property Definitions and Netlist Behavior for CCI Flows](#)

Generating a Flat Netlist for a Specific Device

This procedure shows how to generate a flat CCI netlist for one device type. This procedure can be useful for constructing things like fuse tables.

To generate a flat netlist consisting only of a specific device using source net names, with device locations at the center of the seed shape, use the following steps.

Prerequisites

- Ensure that you have access to a Calibre Connectivity Interface license, as well as Calibre Query Server, Calibre nmLVS, and Calibre nmLVS-H licenses. Refer to the [Calibre Administrator's Guide](#) for license information.

Procedure

1. Use the following rule file statement to enable full CCI functionality within the PHDB:

```
MASK SVDB DIRECTORY "svdb" CCI
```

2. Run Calibre nmLVS-H as you normally would:

```
calibre -spice lay.net -turbo -lvs -hier -hcell cells rules
```

3. Run the Query Server Tcl shell on svdb generated in Step 2.

```
calibre -qs -svdb svdb
```

4. At the Tcl shell prompt, enter these commands (here, a diode is assumed):

```
set dfltr [dfm::create_filter -device_names "D(probe)"]
dfm::set_layout_netlist_options -filter $dfltr" -hierarchy FLAT \
    -device_location CENTER -names SOURCE NETS
dfm::write_spice_netlist probe.net
qs::quit
```

5. (Optionally using the standard interface.) Run the Query Server on svdb generated in Step 2.

```
calibre -query svdb
```

6. (Optionally using the standard interface.) After getting the “OK: Ready to serve.” prompt, issue the following commands:

```
FILTER DEVICENAMES D(probe)
LAYOUT NETLIST HIERARCHY FLAT
LAYOUT NETLIST DEVICE LOCATION CENTER
LAYOUT NETLIST NAMES SOURCE NETS
LAYOUT NETLIST WRITE probe.net
QUIT
```

Results

This creates the desired netlist in the file *probe.net*. This is used in conjunction with an AGF file produced by the `qs::write_agf` (or **AGF WRITE**) command and an LNN file produced by the `qs::write_layout_netlist_names` (or **LAYOUT NAMETABLE WRITE**) command.

Viewing Connect and Device Layers

Connect and Device layers are available in the PHDB. This procedure discusses how to output them to an AGF file.

The `qs::agf_map` command provides a simple way to view **Connect** and **Device** layers used in an LVS run.

Prerequisites

- LVS connectivity extraction rule file.
- The Mask SVDB Directory CCI option is used in the rule file.
- Calibre nmLVS-H, Calibre Query Server, and Calibre Connectivity Interface licenses.

Procedure

1. Run LVS circuit extraction:

```
calibre -spice lay.net -turbo rules
```

2. Use the following commands to echo the layer map list to the terminal, write an SVRF rule file with the layer statement configuration, and write the AGF design:

```
calibre -qs -svdb svdb
qs::agf_map
qs::agf_map -svrf gds_map.rules
qs::write_agf svdb.agf
qs::quit
```

3. Load the *svdb.agf* file into Calibre DESIGNrev:

```
calibredrv svdb.agf
```

Then load the input layer name information in the *gds_map.rules* file using **Layer > Load Input SVRF Layer Names**.

Results

The *svdb.agf* file contains design layers matching the SVRF rule file. When the SVRF layer names are loaded, you can see the corresponding named layers in the AGF.

The *gds_map.svrf* file shows the layer mapping information. The two files allow you to see the connectivity and device layers.

Standard Interface CCI Command Reference

The standard interface dates to the origin of the Query Server. It is superseded by the Tcl shell interface because of the latter's flexibility when it comes to scripting and command coverage in Calibre YieldServer.

All CCI commands require a Calibre Connectivity Interface license in addition to the Query Server license.

Annotated Device Commands	521
Annotated Geometry Format Commands	525
Cell Extents Report Command	565
Customized Layout SPICE Netlist Commands	568
Layout Netlist Names File Command	607
Port Table File Commands	610
Reduction Data File Command	614
Cross-Reference System File Commands	618

Annotated Device Commands

Annotated device layers have DFM properties attached to them, which are used to annotate devices for the CCI flow. The LVS Annotate Devices specification statement must be present in the rule file for these commands to be useful.

Table 6-5. Annotated Device Commands

Command	Description
ANNOTATED DEVICE LAYER MAP <code>qs::annotated_device_layer_map</code>	Prints a report of any layers that are used for annotations on particular devices.
FILTER {IN OUT} ANNOTATED LAYERS	Adds or removes annotated device layers from the CCI system.
AGF ANNOTATED DEVICES <code>qs::set_agf_options -annotated_devices</code>	Causes annotated device seed layers to be output instead of original seed layers.
LAYOUT NETLIST ANNOTATED DEVICES <code>dfm::set_layout_netlist_options -annotated_devices</code>	Controls whether annotated devices are used in generating a SPICE netlist.
LAYOUT NETLIST REPORT BAD ANNOTATED DEVICES	Reports bad annotated devices.

ANNOTATED DEVICE LAYER MAP

CCI command. Corresponding Tcl shell command: [qs::annotated_device_layer_map](#).

Prints a report of any layers that are used for annotations on particular devices.

Usage

ANNOTATED DEVICE LAYER MAP

Arguments

None.

Description

Prints a report of any layers that are used for annotations on particular devices. Each line of the report contains the device type and subtype (if a subtype is present), the device table index (see the [DEVICE TABLE](#) command), and the DFM property annotation layer name for any layer that is currently mapped. [LVS Annotate Devices](#) must be specified in the rules for this command to be useful.

If conflicting layers are present as discussed under “[Conflicting Annotated Device Layers](#)” on page 435, only one layer of the set of conflicting layers is reported by this command.

Output can be sent to a [RESPONSE FILE](#).

Response

```
Annotated_Device_Layer_map <precision>
Device_Layer_Mappings:
0 0 <k> <date>                                // k layer entries
<dev> <dev_idx> <layer_name>
...
END OF RESPONSE
0 0 0 Oct 15 10:43:47 2010
OK.
```

Acknowledgments

OK., NOK(40), ERROR(130)

Examples

```
Annotated_Device_Layer_map 1000
Devices_Layer_Mappings:
0 0 7 Oct 15 10:43:47 2010
mn(n) 0 DLAYER_MN_PROP
mn(ns) 1 DLAYER_MN_PROP
mn(nvt1) 2 DLAYER_MN_PROP
mp(p) 3 DLAYER_MP_PROP
mp(ps) 4 DLAYER_MP_PROP
mp(pvt1) 5 DLAYER_MP_PROP
mn(ncap) 20 DLAYER_MN_PROP
END OF RESPONSE
0 0 0 Oct 15 10:43:47 2010
OK.
```

Related Topics

[Annotated Device Commands](#)

FILTER {IN | OUT} ANNOTATED LAYERS

CCI command.

Adds or removes annotated device layers from the CCI system.

Usage

FILTER IN ANNOTATED LAYERS {ALL | *layer* [*layer* ...]}

FILTER OUT ANNOTATED LAYERS {ALL | *layer* [*layer* ...]}

Arguments

- **IN**

A keyword that specifies annotated layers are included. This is the default.

- **OUT**

A keyword that specifies annotated layers are excluded.

- **ALL**

A keyword that specifies all annotated layers are acted upon by the command. This is the default. Either **ALL** or *layer* must be specified when this command is used.

- ***layer***

A name of a DFM property annotated layer. More than one layer may be specified. Either **ALL** or *layer* must be specified when this command is used.

Description

Adds or removes layers from the set of annotated layers when executing annotated device commands. [LVS Annotate Devices](#) must be specified in the rules for this command to be useful.

When **IN** and **ALL** are specified (the defaults), all non-conflicting annotated device layers are read in. When **IN** and *layer* are specified, all non-conflicting layers from the set of specified layers are read in.

If any layers conflict, the first one of the conflicting set that appears in an LVS Annotate Devices statement is used, and all other layers in the conflicting set are ignored. Conflicting layers are discussed under “[Conflicting Annotated Device Layers](#)” on page 435.

Acknowledgments

OK., NOK(40), ERROR(130), ERROR(161), ERROR(162), ERROR(163)

Related Topics

[Annotated Device Commands](#)

Annotated Geometry Format Commands

Annotated Geometry Format (AGF) files are GDSII stream files (version 6) or OASIS (version 1.0) files created from the layout database. The Calibre Connectivity Interface generates such layouts for use in downstream simulation tools.

To generate AGF files, your rule file must include the [Mask SVDB Directory](#) statement with the GDSII or CCI option. A basic script for generating all CCI files needed for downstream flows is discussed under “[Generating CCI Output Files](#)” on page 480.

You create the AGF file using the [AGF WRITE](#) command. This file is used in conjunction with the outputs of the [LAYOUT NETLIST WRITE](#), [LAYOUT NAMETABLE WRITE](#), and [LAYOUT NET XREF WRITE](#) commands to provide a complete set of cross-references for layout connectivity annotations.

GDS output uses the PROPATTR and PROPVVALUE fields to specify device ids, node numbers, and instance placement names. OASIS output uses the CELLNAME, PROPNAME, PROPSTRING, and LAYERNAME records in STRICT mode. Each CELLNAME record is followed by a S_CELL_OFFSET property record to support arbitrary access. This allows multi-threaded stream-in by Calibre tools.

The following table contains command summaries. Equivalent Tcl shell commands (qs::: or dfm::: scopes) are shown when applicable.

Note

 Historically, many of these command names began with the string “GDS”. However, with the introduction of OASIS output, GDS was changed to AGF in Calibre 2019.4. The GDS forms of the original command names may still be used, but that should be limited to cases where the output is actually GDS to prevent possible confusion.

Table 6-6. AGF Commands

Command	Description
AGF ANNOTATED DEVICES qs:::set_agf_options -annotated_devices	Causes DFM property annotated device seed layers to be output instead of original seed layers.
AGF CLEAR FILTER BOX LAYERS dfm:::create_filter -box_layers	Restores the default behavior of including the same set of layers in LVS Box cells that appear in non-box cells. The present configuration of the FILTER LAYERS command is enforced.
AGF CLEAR MERGE LAYERS qs:::set_agf_options -clear_merge_layers	Clears the current list of layers specified for merging by the AGF MERGE LAYER or AGF MERGE PIN LAYERS commands.

Table 6-6. AGF Commands (cont.)

Command	Description
AGF CLEAR PCELL PROMOTE LAYERS qs::set_agf_options -clear_pcell_promote_layers	Clears the current AGF PCELL PROMOTE LAYERS list.
AGF FILTER BOX LAYERS dfm::create_filter -box_layers	Controls the inclusion of layers in the AGF WRITE output for LVS Box cells. This command acts independently from FILTER LAYERS .
AGF FILTER {IN OUT} BOX LAYERS	Controls the inclusion of layers in the AGF WRITE output for LVS Box cells. Layers that are previously included or excluded have their inclusion status changed by this command. This command acts independently from FILTER LAYERS.
AGF FILTER MAP qs::agf_map -filter	Controls whether the AGF MAP command outputs only filtered layers as specified in the FILTER LAYERS command.
AGF FORMAT qs::set_agf_options -format	Controls the layout format used by AGF WRITE.
AGF HIERARCHY qs::set_agf_options -hierarchy	Controls the hierarchy output by the AGF WRITE (or qs::write_agf) command.
AGF HIERARCHY CLEAR EXPAND CELLS qs::set_agf_options -hierarchy_clear_expand_cells	Clears the current list of cells specified with AGF HIERARCHY EXPAND CELL (or qs::set_agf_options -hierarchy_expand_cell) commands.
AGF HIERARCHY EXPAND CELL qs::set_agf_options -hierarchy_expand_cell	Causes expansion of cells in AGF output.
AGF HIERARCHY EXPAND DEVICELESS CELLS qs::set_agf_options -hierarchy_expand_deviceless_cells	Controls automatic expansion of cells that contain no devices in them or in the lower-level placements within them.
AGF MAGNIFY USER UNITS qs::magnify_user_units	Controls whether non-default MAGNIFY RESULTS factors are applied.
AGF MAP qs::agf_map	Returns or configures the layer map scheme of AGF output.

Table 6-6. AGF Commands (cont.)

Command	Description
AGF MERGE LAYER qs::set_agf_options -merge_layer	Causes a layer to be merged in AGF output.
AGF MERGE PIN LAYERS qs::set_agf_options -merge_pin_layers	Causes selected device pin layers to be merged in AGF output.
AGF {DEVPROP NETPROP PLACEPROP} NAME qs::set_agf_options -devprop_name, -netprop_name, -placeprop_name	Configures the PROPNAMES record values corresponding to PROPSTRING records in the AGF output when it is OASIS.
AGF {DEVPROP NETPROP PLACEPROP} NUMBER qs::set_agf_options -devprop_number, -netprop_number, -placeprop_number	Configures the PROPATR record numbers for PROPVALUE records in the AGF output when it is GDS.
AGF PCELL PROMOTE LAYERS qs::set_agf_options -pcell_promote_layers	Specifies pcell layers in the PHDB that are available for promotion in a subsequent AGF WRITE command.
AGF RESET qs::set_agf_options -reset	Sets the AGF commands to their default values.
AGF SEED PROPERTY qs::set_agf_options -seed_property	Configures the type of device seed properties that can appear in AGF output.
AGF SVRF MAP qs::agf_map -svrf	Configures the format of AGF MAP (or qs::agf_map).
AGF UNITS qs::set_agf_options -units	Specifies the values of the UNITS record in the output of the AGF WRITE (or qs::write_agf) command.
AGF WRITE qs::write_agf	Writes an AGF file.

Layer Output Control for AGF Files

The annotated geometry format (AGF) file created by AGF WRITE can serve as input to a parasitic extraction tool. You may want to output only certain layers (perhaps the layers involved in top-level connectivity) to the stream file to simplify extraction.

The following things apply:

- Only the layers involved in connectivity and in **Device** operations are output in the AGF by default. The **LVS DB Layer** and **LVS DB Connectivity Layer** specification statement add other layers to the set of output layers.
- For general layer filtering, use the **FILTER LAYERS** command after the **AGF MAP** command and before the **AGF WRITE** command. The order of the commands is important. This guidance also applies for **AGF FILTER BOX LAYERS** and **AGF FILTER {IN | OUT} BOX LAYERS**.
- If the layer is assigned a name in the rule file, then the layer number cannot be used in the FILTER LAYERS command.
- If you do not explicitly map the layers to some layer numbers, the layers get mapped as layer numbers 1, 2, 3, and so forth. You can control datatypes and texttypes by using the AGF MAP command. For example, this maps metal1 to layer 12 datatype 0:

```
AGF MAP metal1 12 0
```

AGF ANNOTATED DEVICES

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -annotated_devices](#).

Controls whether annotated device seed layers are output instead of original seed layers.

Usage

AGF ANNOTATED DEVICES {NO | YES}

Alternate Command Name

GDS ANNOTATED DEVICES

Arguments

- **NO**

A keyword that specifies annotated device seed layers are not to be output. This is the default.

- **YES**

A keyword that specifies annotated device seed layers are to be output.

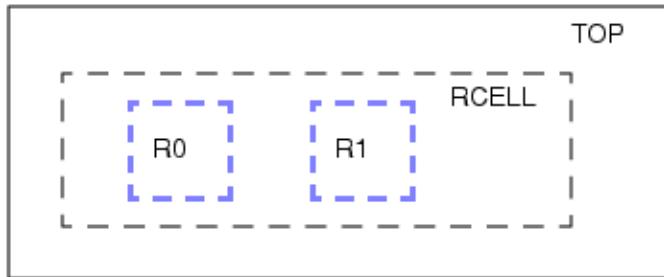
Description

Configures the output of the [AGF WRITE](#) command for annotated device seed layers in place of the originally recognized device seed layers. The property annotations contain device instance paths as values. The default is **NO**.

To use the **YES** keyword, the [LVS Annotate Devices](#) specification statement must be present in the rule file. When **YES** is specified, the DFM property annotated devices are represented by promoted seed shapes containing the paths to the originally-recognized devices in higher-level circuits of the design.

Assume you have a resistor cell called RCELL with two device instances, R0 and R1. Assume RCELL is placed once in cell TOP as shown in [Figure 6-3](#).

Figure 6-3. Resistor Cell Placed in TOP



Assume the AGF file for the preceding design is represented by two device seed shapes contained in RCELL. Suppose that a property P can be measured using [DFM Property](#).

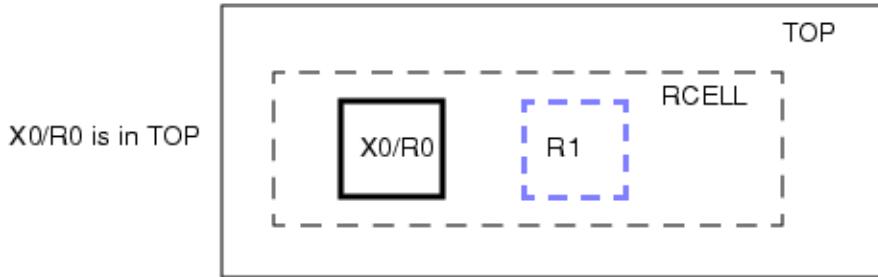
Furthermore, assume the resistor R0 must be promoted to the top level in order to make a meaningful measurement for P. Assume this is the original netlist:

```
.SUBCKT RCELL 1 2
R0 1 3 2 40
R1 1 3 2 50
.ENDS

.SUBCKT TOP
X0 RCELL A B
.ENDS
```

If AGF ANNOTATED DEVICES YES is used when the AGF file is generated, the new promoted location of R0 with its extended pathname is generated as a property in the file. RCELL now contains only instance R1. TOP contains both the promoted instance R0 and RCELL as shown in [Figure 6-4](#).

Figure 6-4. Resistor Instance Promoted in AGF



The annotated SPICE netlist might appear like this:

```
.SUBCKT RCELL 1 2
RR1 1 3 2 50 $P=1.8e-07
.ENDS

.SUBCKT TOP
RX0/R0 A B X0/3 40 $P=2.3e-06
X0 RCELL A B
.ENDS
```

Property P is the DFM annotated property for the instance.

Acknowledgments

OK., NOK(57)

Related Topics

[LAYOUT NETLIST ANNOTATED DEVICES](#)

[Annotated Device Commands](#)

[Annotated Geometry Format Commands](#)

AGF CLEAR FILTER BOX LAYERS

CCI command. Corresponding Tcl shell command: [dfm::create_filter -box_layers](#).

Restores the default behavior of including the same set of layers in LVS Box cells that appear in non-box cells.

Usage

AGF CLEAR FILTER BOX LAYERS

Alternate Command Name

GDS CLEAR FILTER BOX LAYERS

Arguments

None.

Description

Clears the layer inclusion and exclusion lists for LVS Box cells established by [AGF FILTER BOX LAYERS](#) and [AGF FILTER {IN | OUT} BOX LAYERS](#) commands. The present configuration of the FILTER LAYERS is unaffected and still applies, however.

Acknowledgments

OK.

Examples

```
# exclude all layers from box cells
AGF FILTER BOX LAYERS NONE
# write output
AGF WRITE devices_no_box_layers.gds.gz
# remove layer filtering
AGF CLEAR FILTER BOX LAYERS
# write output again, with all layers included in box cells
AGF WRITE devices.gds.gz
```

Related Topics

[Annotated Geometry Format Commands](#)

AGF CLEAR MERGE LAYERS

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -clear_merge_layers](#).
Clears the current list of layers specified in CCI for merging.

Usage

AGF CLEAR MERGE LAYERS

Alternate Command Name

GDS CLEAR MERGE LAYERS

Arguments

None.

Description

When this command is executed, after clearing the current list of merged layers, the [AGF MERGE LAYER](#) and [AGF MERGE PIN LAYERS](#) commands are restored to their defaults.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

AGF CLEAR PCELL PROMOTE LAYERS

CCI command. Corresponding Tcl shell command:

`qs::set_agf_options -clear_pcell_promote_layers.`

Clears the current AGF PCELL PROMOTE LAYERS list.

Usage

AGF CLEAR PCELL PROMOTE LAYERS

Alternate Command Name

GDS CLEAR PCELL PROMOTE LAYERS

Arguments

None.

Return Values

Acknowledgment: OK.

AGF FILTER BOX LAYERS

CCI command. Corresponding Tcl shell command: [dfm::create_filter -box_layers](#).

Controls the inclusion of layers in the AGF WRITE output for LVS Box cells. This command acts independently from FILTER LAYERS settings.

Usage

AGF FILTER BOX LAYERS {ALL | NONE | *layer* [*layer* ...]}

Alternate Command Name

GDS FILTER BOX LAYERS

Arguments

- **ALL**

A keyword that specifies all layers are included in the output of [LVS Box](#) cells. If any layers are in the exclusion set at the time AGF FILTER BOX LAYERS ALL is called, they are included.

- **NONE**

A keyword that specifies no layers are included in the output of LVS Box cells. If any layers are in the inclusion set at the time AGF FILTER BOX LAYERS NONE is called, they are excluded.

- ***layer***

A layer name to be included in the output of LVS Box cells. All other layers are excluded.
More than one layer name may be specified.

Description

By default, the set of layers written to LVS Box cells is the same as for non-box cells. This command allows the list of layers written by [AGF WRITE](#) to LVS Box cells to be different from non-box cells. Any [FILTER LAYERS](#) settings are ignored for LVS Box cells when AGF FILTER BOX LAYERS is used.

If you have regular LVS Box cells with subcells that are also placed in non-box cells, using [LVS Clone By LVS Box Cells](#) YES in your rule file can be desirable to get only the filter layers in the AGF output.

Acknowledgments

OK., ERROR(114)

Examples

```
# write the layer map
RESPONSE FILE gds.map
AGF MAP
RESPONSE DIRECT
# exclude all layers from box cells
AGF FILTER BOX LAYERS NONE
# write output
AGF WRITE devices_no_box_layers.gds.gz
```

Related Topics

[AGF FILTER {IN | OUT} BOX LAYERS](#)

[AGF CLEAR FILTER BOX LAYERS](#)

[Annotated Geometry Format Commands](#)

AGF FILTER {IN | OUT} BOX LAYERS

CCI command.

Controls the inclusion of layers in the AGF WRITE output for LVS Box cells. Layers that are previously included or excluded have their inclusion status changed by this command. This command acts independently from FILTER LAYERS.

Usage

AGF FILTER IN BOX LAYERS *layer* [*layer* ...]

AGF FILTER OUT BOX LAYERS *layer* [*layer* ...]

Alternate Form

GDS FILTER ...

Arguments

- **IN**

A keyword that specifies layers are included in the output of [LVS Box](#) cells, regardless of their inclusion status before the command is called.

- **OUT**

A keyword that specifies layers are excluded from the output of LVS Box cells, regardless of their inclusion status before the command is called.

- ***layer***

A required name of a layer. More than one layer may be specified.

Description

By default, the set of layers written to LVS Box cells is the same as for non-box cells. This command allows the list of layers written by [AGF WRITE](#) to LVS Box cells to be different from non-box cells.

A [FILTER LAYERS](#) setting that is opposite of an IN or OUT specification is ignored for LVS Box cell layer output. For example, if FILTER LAYERS includes a layer, but AGF FILTER OUT BOX LAYERS specifies the same layer, then the layer is omitted in the AGF output for box cells.

If you have regular LVS Box cells with subcells that are also placed in non-box cells, using [LVS Clone By LVS Box Cells](#) YES in your rule file can be desirable to get only the filter layers in the AGF output.

There is no Tcl shell command that matches this command. However, you can get similar behavior by defining a filter object for each set of layers you want to use with [dfm::create_filter_box_layers](#).

Acknowledgments

OK., ERROR(114)

Examples

```
# write the layer map
RESPONSE FILE gds.map
AGF MAP
RESPONSE DIRECT
# exclude output layers from box cells
AGF FILTER BOX LAYERS NONE
# include only pin layers in box cells
AGF FILTER IN BOX LAYERS ngate pgate nsd psd res_pin cap_pos cap_neg
# write output
AGF WRITE devices.gds.gz
```

Related Topics

[AGF FILTER BOX LAYERS](#)

[AGF CLEAR FILTER BOX LAYERS](#)

[Annotated Geometry Format Commands](#)

AGF FILTER MAP

CCI command. Corresponding Tcl shell command: [qs::agf_map -filter](#).

Controls whether the AGF MAP command outputs only filtered layers as specified in the FILTER LAYERS command.

Usage

AGF FILTER MAP {NO | YES}

Alternate Command Name

GDS FILTER MAP

Arguments

- **NO**

A keyword that specifies all layers that contain geometry in the PHDB are output by the AGF MAP command. This is the default.

- **YES**

A keyword that specifies only filtered layer geometry is to be output.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

[AGF MAP](#)

[FILTER LAYERS](#)

AGF FORMAT

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -format](#).

Controls the layout format used by AGF WRITE.

Usage

AGF FORMAT {GDSII | {OASIS [NOSTRICT | STRICT]}}

Arguments

- **GDSII**

A keyword that specifies GDSII is the output format. This is the default.

- **OASIS [NOSTRICT | STRICT]**

A keyword set that specifies OASIS is the output format. OASIS output uses CBLOCK compression. All records to which strict-mode formatting applies are output using that mode except for PROPSTRING records, to which these keywords apply:

NOSTRICT — An optional keyword that specifies PROPSTRING records are output in non-strict mode. This is the default when **OASIS** is specified. NOSTRICT can reduce OASIS file output memory consumption by up to 3×. It may also substantially reduce memory consumption of downstream tools that use the output OASIS file as an input.

STRICT — An optional keyword that specifies PROPSTRING records are output in strict mode.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

AGF HIERARCHY

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -hierarchy](#).

Controls the hierarchy output by the AGF WRITE command.

Usage

AGF HIERARCHY {AGF | HCELL | FLAT}

Alternate Command Name

GDS HIERARCHY

Arguments

- **AGF**

A keyword that specifies to generate the native AGF hierarchy with no additional expansion of cells and is consistent with normal GDSII output hierarchy. This is the default.

- **HCELL**

A keyword that specifies to expand all non-hcells into the containing hcell. It also expands unbalanced hcells that were expanded during LVS comparison.

- **FLAT**

A keyword that specifies to expand all lower-level cells into the top-level cell.

Acknowledgments

OK., ERROR(122)

Related Topics

[Annotated Geometry Format Commands](#)

[AGF WRITE](#)

AGF HIERARCHY CLEAR EXPAND CELLS

CCI command. Corresponding Tcl shell command:

`qs::set_agf_options -hierarchy_clear_expand_cells.`

Clears the current list of cells specified with AGF HIERARCHY EXPAND CELL commands.

Usage

AGF HIERARCHY CLEAR EXPAND CELLS

Alternate Command Name

GDS HIERARCHY CLEAR EXPAND CELLS

Arguments

None.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

[AGF HIERARCHY EXPAND CELL](#)

AGF HIERARCHY EXPAND CELL

CCI command. Corresponding Tcl shell command:

`qs::set_agf_options -hierarchy_expand_cell.`

Causes expansion of cells in AGF output.

Usage

AGF HIERARCHY EXPAND CELL *cell_name*

Alternate Command Name

GDS HIERARCHY EXPAND CELL

Arguments

- *cell_name*

A required name of a cell. The name may contain the asterisk (*) wildcard, which matches zero or more characters.

Description

Causes the [AGF WRITE](#) command to expand the cell specified by *cell_name* by one level.

Cell names can be original cell names or FAUX BIN cells as identified in the LVS run transcript.

See also [AGF HIERARCHY EXPAND DEVICELESS CELLS](#) and [AGF HIERARCHY CLEAR EXPAND CELLS](#).

Acknowledgments

OK., NOK(36)

Related Topics

[Annotated Geometry Format Commands](#)

AGF HIERARCHY EXPAND DEVICELESS CELLS

CCI command. Corresponding Tcl shell command:

`qs::set_agf_options -hierarchy_expand_deviceless_cells.`

Controls automatic expansion of cells that contain no devices in them or in the lower-level placements within the cells.

Usage

AGF HIERARCHY EXPAND DEVICELESS CELLS {NO | YES}

Alternate Command Name

GDS HIERARCHY EXPAND DEVICELESS CELLS

Arguments

- **NO**
A keyword that disables automatic expansion of device-less cells. This is the default.
- **YES**
A keyword that enables automatic expansion of device-less cells.

Acknowledgments

OK.

Related Topics

[AGF HIERARCHY EXPAND CELL](#)

[Annotated Geometry Format Commands](#)

AGF MAGNIFY USER UNITS

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -magnify_user_units](#).
Controls whether non-default MAGNIFY RESULTS factors are applied.

Usage

AGF MAGNIFY USER UNITS {YES | NO}

Arguments

- **YES**

A keyword that specifies magnification of user units occurs when a non-default [MAGNIFY RESULTS](#) factor is in effect.

- **NO**

A keyword that specifies magnification of user units does not occur.

Acknowledgments

OK.

Examples

These commands cause magnification of AGF output by a factor of 0.5:

```
MAGNIFY RESULTS 0.5
AGF MAGNIFY RESULTS YES
...
AGF WRITE out.agds
```

Related Topics

[Annotated Geometry Format Commands](#)

AGF MAP

CCI command. Corresponding Tcl shell command: [qs::agf_map](#).

Returns or configures the layer map scheme of AGF output.

Usage

AGF MAP [*calibre_layer design_layer [design_datatype]*]

AGF MAP *calibre_layer design_layer design_datatype* {**DEVICE** | **ORIGINAL**}

Alternate Command Name

GDS MAP

Arguments

- *calibre_layer*

The name (not number) of a Calibre rule file layer. This argument is required if **DEVICE** or **ORIGINAL** is specified.

- *design_layer*

The number of an AGF output layer. This argument is required if **DEVICE** or **ORIGINAL** is specified.

- *design_datatype*

The number of an AGF layer datatype. This argument is required if **DEVICE** or **ORIGINAL** is specified.

- **DEVICE**

A required keyword in the second syntax form of the command. This keyword configures the output layer as a device seed layer. May not be specified with **ORIGINAL**.

- **ORIGINAL**

A required keyword in the second syntax form of the command. This keyword configures the output layer as an original layer. May not be specified with **DEVICE**.

Description

Without any optional arguments, this command returns the current Calibre layer name to layer number and datatype mapping for all Device layers output by a subsequent [AGF WRITE](#) command. Output layer numbers and datatypes are assigned by the system by default. Output can be sent to a [RESPONSE FILE](#) in this case.

If any of the optional arguments are specified, the command does not return the layer mapping information. Instead, the additional parameters configure the AGF WRITE command output. If *design_datatype* is specified, then the output uses that datatype; otherwise, the datatype is 0. The GDSII standard specifies layers and datatypes are numbers between zero and 63; however, values between -32767 and +32767 are accepted by the Query Server.

If SVRF layer statement output is preferred (this is useful in Calibre DESIGNrev for annotating layer names in the layer palette), specify **AGF SVRF MAP YES**.

If **AGF SEED PROPERTY DEVICE ORIGINAL** is specified, then either the **DEVICE** or **ORIGINAL** keyword must be supplied in a AGF MAP command in order to configure the desired output layer. Separate AGF MAP commands are needed to produce each **DEVICE** or **ORIGINAL** output layer type.

The precision value in the output response header is multiplied by the **MAGNIFY RESULTS** factor. Magnification factors can be useful in certain third-party timing analysis tool flows.

Response

```
Gds_Map <precision>          // AGF_Map and rule file precision
Layers:                      // "Layers:"
0 0 <n>           <date>    // n lines of text follow
// mapped layer names to AGF layer numbers and datatypes
// annotations dependent on AGF SEED PROPERTY configuration
<name> <number> <datatype> [DEVICE | ORIGINAL | TENTATIVE_PROMOTION]
<name> <number> <datatype> [DEVICE | ORIGINAL | TENTATIVE_PROMOTION]
...
...
```

Acknowledgments

OK., ERROR(114), ERROR(123), ERROR(124), ERROR(133)

Related Topics

[Annotated Geometry Format Commands](#)

[Generating CCI Output Files](#)

[AGF FILTER MAP](#)

[AGF SVRF MAP](#)

[Results Transformation Commands](#)

AGF MERGE LAYER

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -merge_layer](#).

Causes a layer to be merged in AGF output.

Usage

AGF MERGE LAYER *layer*

Alternate Command Name

GDS MERGE LAYER

Arguments

- *layer*

A required polygon layer name that exists in the Mask SVDB Directory PHDB.

Description

Causes the *layer* to be merged by a subsequent [AGF WRITE](#) command. Layers that are merged are present in the Mask SVDB Directory PHDB that is loaded by the Query Server. This command can be invoked multiple times in order to merge multiple layers.

Geometries in merged layers are promoted as necessary to higher levels of the hierarchy in order to ensure that each shape is a single polygon contained entirely in one cell. Merging large layers, such as bulk pin layers or metal layers that form resistor pins, can degrade performance because of the flattening of the layers.

By default, device seed shapes are fully merged on output by AGF WRITE. However, device seed layers might not be merged if the ORIGINAL keyword of [AGF SEED PROPERTY](#) is used in the flow. Also, the Calibre device extraction module does not require pin layers to be merged in order to extract a device successfully. However, certain parasitic extraction tools do require merged pin layers in order to generate accurate results. This command is useful in generating AGF output for such tools.

An empty input layer is processed with no effect on the output. An error occurs if any selected layer is unsuitable for merging.

Acknowledgments

OK, ERROR(185)

Examples

This shows a command sequence where layer merging is needed for gate pin layers:

```
# output seed polygons with either net ids or with instance names
AGF SEED PROPERTY DEVICE ORIGINAL

# output annotated GDSII mapping file.
# output the gates on layer 4 using datatypes 1 through 4.
# output instance names for DEVICE, net names for ORIGINAL.
RESPONSE FILE gds.map
AGF MAP ngate 4 1 ORIGINAL
AGF MAP ngate 4 2 DEVICE
AGF MAP pgate 4 3 ORIGINAL
AGF MAP pgate 4 4 DEVICE
AGF MAP
RESPONSE DIRECT
# merge the gate layers
AGF MERGE LAYER pgate
AGF MERGE LAYER ngate
AGF WRITE devices.agds.gz
AGF RESET
```

Related Topics

[AGF MERGE PIN LAYERS](#)

[AGF CLEAR MERGE LAYERS](#)

[Annotated Geometry Format Commands](#)

AGF MERGE PIN LAYERS

CCI command. Corresponding Tcl shell command: `qs::set_agf_options -merge_pin_layers`.
Causes selected device pin layers to be merged in AGF output.

Usage

AGF MERGE PIN LAYERS *element_name* [‘(‘*model_name*‘)’] *pin_name*

Alternate Command Name

GDS MERGE PIN LAYERS

Arguments

- ***element_name***
A required SPICE device element name. An asterisk (*) wildcard matches zero or more characters.
- ***pin_name***
A required device pin name. An asterisk (*) wildcard matches zero or more characters.
- ‘(‘*model_name*‘)’
An optional device model (or subtype) name enclosed in parentheses. An asterisk (*) wildcard matches zero or more characters. If this argument is unspecified, then all available models of the ***element_name*** are matched.

Description

Causes selected device pin layers to be merged by a subsequent **AGF WRITE** command. The pin layers that get merged correspond to device pins matched by the AGF MERGE PIN LAYERS arguments. Device types, pins, and models (if specified) are selected from among the Device statements present in the rule file used during connectivity extraction. The corresponding pin layers that are merged are present in the Mask SVDB Directory PHDB that is loaded by the Query Server. An error occurs if any selected pin layer is unsuitable for merging.

This command can be invoked multiple times to merge multiple layers. Using asterisk wildcards in argument names can have a similar effect to specifying the command more than once.

Geometries in merged layers are promoted as necessary to higher levels of the hierarchy in order to ensure that each shape is a single polygon contained entirely in one cell. Merging large layers, such as bulk pin layers or metal layers that form resistor pins, can degrade performance because of the flattening of the layers.

By default, device seed shapes are fully merged on output by AGF WRITE. However, device seed layers might not be merged if the ORIGINAL keyword to **AGF SEED PROPERTY** is used in the flow. Also, the Calibre device extraction module does not require pin layers to be merged in order to extract a device successfully. However, certain parasitic extraction tools do require

merged pin layers in order to generate accurate results. This command is useful in generating AGF output for such tools.

If the command's arguments do not match any device, an NOK(73) acknowledgment is given. It is not an error if the *pin_name* does not match any of the pin names of any particular device matching the given wildcard for an element/model pair as long as some device eventually matches that *pin_name*.

Acknowledgments

OK., NOK(73), ERROR(185)

Examples

This shows a command sequence in which seed layer merging is useful.

```
# output seed polygons with either net ids or with instance names
AGF SEED PROPERTY DEVICE ORIGINAL

# output annotated GDSII mapping file.
# output the gates on layer 3 using datatypes 1 through 4.
# output instance names for DEVICE, net names for ORIGINAL.
RESPONSE FILE gds.map
AGF MAP pgate 3 1 DEVICE
AGF MAP pgate 3 2 ORIGINAL
AGF MAP ngate 3 3 DEVICE
AGF MAP ngate 3 4 ORIGINAL
AGF MAP
RESPONSE DIRECT
# merge g pin layers
AGF MERGE PIN LAYERS mp g
AGF MERGE PIN LAYERS mn g
AGF WRITE devices.agds.gz
AGF RESET
```

Related Topics

[AGF MERGE LAYER](#)

[AGF CLEAR MERGE LAYERS](#)

[Annotated Geometry Format Commands](#)

AGF {DEVPROP | NETPROP | PLACEPROP} NAME

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -devprop_name, -netprop_name, -placeprop_name](#).

Configures the PROPNAME record values corresponding to PROPSTRING records in the AGF output when it is OASIS.

Usage

AGF DEVPROP NAME *name*

AGF NETPROP NAME *name*

AGF PLACEPROP NAME *name*

Arguments

- **DEVPROP**

A keyword that specifies the PROPNAME record corresponds to device instance names.

- **NETPROP**

A keyword that specifies the PROPNAME record corresponds to net numbers.

- **PLACEPROP**

A keyword that specifies the PROPNAME record corresponds to placement paths.

- ***name***

A required name of a property. By default, the property names are Device_Property, Net_Property, and Placement_Property, which correspond to the three keywords mentioned previously.

Description

Changes the value of the PROPNAME record inserted in the output of the [AGF WRITE](#) command to the specified ***name***.

Note

 This command does not apply to annotated GDS output.

The PROPSTRING records controlled by the command's keywords are assigned to output polygons. **DEVPROP** properties appear on device seed shapes and correspond to instance names. **NETPROP** properties appear on interconnect shapes and correspond to node numbers. **PLACEPROP** properties appear on placements and correspond to placement paths.

See [AGF SEED PROPERTY](#) for a means to control the **DEVPROP** property value.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

AGF {DEVPROP | NETPROP | PLACEPROP} NUMBER

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -devprop_number, -netprop_number, -placeprop_number](#).

Configures the PROPATTR record values corresponding to PROPVALUE records in the AGF output when it is GDS.

Usage

AGF DEVPROP NUMBER *number*

AGF NETPROP NUMBER *number*

AGF PLACEPROP NUMBER *number*

Alternate Forms

GDS {DEVPROP | NETPROP | PLACEPROP} NUMBER

Arguments

- **DEVPROP**

A keyword that specifies the PROPVALUE record corresponds to device instance names.

- **NETPROP**

A keyword that specifies the PROPVALUE record corresponds to net numbers.

- **PLACEPROP**

A keyword that specifies the PROPVALUE record corresponds to placement names.

- ***number***

A required positive integer that specifies the value of a PROPATTR record associated with the PROPVALUE record. The default is 1 for all record types.

Description

Changes the value of the PROPATTR record inserted in the output of the [AGF WRITE](#) command to the specified ***number***.

Note

 This command does not apply to annotated OASIS output.

The PROPVALUE records controlled by the command's keywords are assigned to output polygons. **DEVPROP** properties appear on device seed shapes and correspond to instance names. **NETPROP** properties appear on interconnect shapes and correspond to node numbers. **PLACEPROP** properties appear on placements and correspond to placement names.

See [AGF SEED PROPERTY](#) for a means to control the **DEVPROP** property value.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

AGF PCELL PROMOTE LAYERS

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -pcell_promote_layers](#).
Specifies pcell layers in the PHDB that are available for promotion in a subsequent AGF WRITE command.

Usage

AGF PCELL PROMOTE LAYERS *layer* [*layer* ...]

Alternate Command Name

GDS PCELL PROMOTE LAYERS

Arguments

- *layer*

A pcell layer name to be available for promotion in the AGF WRITE output. At least one layer name must be specified.

Description

Specifies a list of layer names from PEX xcells and cells specified in a [PEX Preserve Cell List](#) statement. The specified layers are promoted in the output from a subsequent AGF WRITE command. This promotion can be advantageous in certain parasitic modeling flows.

If a *layer* is a Device seed layer, then the [AGF SEED PROPERTY ORIGINAL](#) keyword must also be specified, otherwise the seed shapes are not promoted. “TENTATIVE_PROMOTION” information produced by the AGF SEED PROPERTY TENTATIVE keyword is unaffected by pcell layer promotion.

If a *layer* is also specified by [AGF FILTER BOX LAYERS](#), then the promotion occurs first, followed by the box layer filtering.

To clear the promotion layer list, use [AGF CLEAR PCELL PROMOTE LAYERS](#).

Acknowledgments

OK., ERROR(114)

AGF RESET

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -reset](#).

Sets the AGF configuration commands to their default values.

Usage

AGF RESET

Alternate Command Name

GDS RESET

Arguments

None.

Description

Causes default AGF command settings to take effect for the [AGF WRITE](#) command.

Does not affect the [MAXIMUM VERTEX COUNT](#) command setting.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

AGF SEED PROPERTY

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -seed_property](#).

Configures the type of device seed properties that can appear in AGF output.

Usage

AGF SEED PROPERTY {DEVICE | ORIGINAL | **DEVICE ORIGINAL} [TENTATIVE]**

Alternate Command Name

GDS SEED PROPERTY

Arguments

- **DEVICE**

A keyword that enables device instance name properties to be output on device seed shapes. This is the default behavior. If this keyword is not specified, then **ORIGINAL** must be.

- **ORIGINAL**

A keyword that enables net name properties to be output on device seed shapes. If this keyword is not specified, then **DEVICE** must be.

- **TENTATIVE**

An optional keyword that specifies the output from [AGF WRITE](#) contains tentatively promoted device seed shapes, and the output from [AGF MAP](#) contains "TENTATIVE_PROMOTION" entries for layers with tentatively promoted shapes.

Description

Controls properties attached to device seed shapes when AGF WRITE is executed.

The **DEVICE** keyword enables device instance names to be written, which is done by default. The **ORIGINAL** keyword enables net name properties (if available on seed layers) to be written. Using both **DEVICE** and **ORIGINAL** enables both forms to be written.

The property numbers that get used are controlled by the [AGF {DEVPROP | NETPROP | PLACEPROP} NUMBER](#) command. DEVPROP is for device instances, NETPROP is for net names.

If this command is used with both **DEVICE** and **ORIGINAL**, then associated AGF MAP commands must use these keywords (one command for each keyword), depending on which

output property is desired on a given output layer. For example, this sequence outputs properties of both types:

```
AGF SEED PROPERTY DEVICE ORIGINAL
RESPONSE FILE ./DEVICES.gds_map
# output the gates on layer 3 using datatypes 1 through 4.
# output instance names for DEVICE, net names for ORIGINAL.
AGF MAP pgate 3 1 DEVICE
AGF MAP pgate 3 2 ORIGINAL
AGF MAP ngate 3 3 DEVICE
AGF MAP ngate 3 4 ORIGINAL
AGF MAP
RESPONSE DIRECT
AGF WRITE ./DEVICES.agf
```

Due to seed promotion, original layer shapes may occur at a different level than the recognized seed shapes. In such cases, the [AGF MERGE LAYER](#) or [AGF MERGE PIN LAYERS](#) command may be useful to merge the shapes.

The **ORIGINAL** keyword must be used in order for seed layers specified by [AGF PCELL PROMOTE LAYERS](#) to be promoted.

Acknowledgments

OK.

Examples

This example shows a method for analyzing tentative device seed promotion. In an AGF flow, tentative seed promotion information appears in the circuit extraction transcript when the [Mask SVDB Directory](#) CCI NOPINLOC options are used or when [LVS Push Devices SEPARATE PROPERTIES YES](#) is used.

When the AGF SEED PROPERTY TENTATIVE keyword is used, the AGF writer can output geometries at the locations of tentative seed promotions. These geometries appear on a dedicated layer in the AGF file. In the AGF MAP output, these layers are labeled "<layer>_TENTATIVE_PROMOTION" where <layer> is the name of the original seed layer of the devices. This output is helpful in determining the cause of seed promotion during device extraction. Here is a typical command sequence:

```
AGF SEED PROPERTY DEVICE ORIGINAL TENTATIVE
AGF SVRF MAP YES
RESPONSE FILE agf.gds.svrf
AGF MAP
RESPONSE DIRECT
```

The output AGF file can then be loaded into Calibre DESIGNrev. The output SVRF rule file can be loaded using the **Layer > Load Input SVRF Names** menu item. The layer palette will then contain the annotated layer names, along with the "*_TENTATIVE_PROMOTION" layers. You can toggle the visibility of layers as desired to debug seed promotion issues.

Related Topics

[Annotated Geometry Format Commands](#)

AGF SVRF MAP

CCI command. Corresponding Tcl Shell command: [qs::agf_map -svrf](#).

Configures the format of AGF MAP command output.

Usage

AGF SVRF MAP {NO | YES}

Alternate Command Name

AGF SVRF MAP

Arguments

- **NO**

A keyword that causes the [AGF MAP](#) response format to be enabled. This is the default setting.

- **YES**

A keyword that enables use of SVRF format.

Description

Controls the AGF MAP command's response format. The SVRF format makes transferring layer names into Calibre DESIGNrev and viewing the AGF file easier. See “[Viewing Connect and Device Layers](#)” on page 519.

The following shows the difference between the default AGF MAP format and the SVRF format when saved using [RESPONSE FILE](#).

AGF SVRF MAP NO	AGF SVRF MAP YES
Gds_Map 1000	//Gds_Map 1000
Layers:	//Layers:
0 0 13 Jan 14 14:31:05 2015	// 0 0 13 Jan 14 14:20:58 2015
pwell 1 0	LAYER pwell 1
nwell 2 0	LAYER nwell 2
poly 3 0	LAYER poly 3
psd 4 0	LAYER psd 4
pgate 6 0	LAYER pgate 6 // DEVICE
ptap 7 0	LAYER ptap 7
nsd 8 0	LAYER nsd 8
ngate 10 0	LAYER ngate 10 // DEVICE
ntap 11 0	LAYER ntap 11
metal1 12 0	LAYER metal1 12
contact 13 0	LAYER contact 13
via 14 0	LAYER via 14
metal2 15 0	LAYER metal2 15
END OF RESPONSE	// END OF RESPONSE
0 0 0 Jan 14 14:31:05 2015	// 0 0 0 Jan 14 14:20:58 2015

If YES is specified and non-zero datatypes are used, then the associated [Layer Map](#) and [Layer](#) statements are included in the output.

If the **AGF SEED PROPERTY ORIGINAL** keyword is used, then “_ORIGINAL” suffixes are added to layer names in the output. If the AGF SEED PROPERTY TENTATIVE keyword is used, then “_TENTATIVE_PROMOTION” suffixes are added to layer names in the output for layers having tentatively promoted seed shapes.

Acknowledgments

OK.

Related Topics

[Annotated Geometry Format Commands](#)

AGF UNITS

CCI command. Corresponding Tcl shell command: [qs::set_agf_options -units](#).

Specifies the values of the UNITS record in the output of the AGF WRITE command. AGF UNITS has no other effect on CCI output. Magnification of output is controlled by the MAGNIFY RESULTS and AGF MAGNIFY USER UNITS commands.

Usage

AGF UNITS *precision dbu_size*

Alternate Command Name

GDS UNITS

Arguments

- *precision*

A required floating-point number that specifies the physical precision of the output database. The value is typically on the interval (0, 0.001]. The reciprocal of the rule file [Precision](#) is the default.

- *dbu_size*

A required number that specifies the database unit in meters. The number may be floating-point or in scientific notation. The *dbu_size* is the ratio of the [Unit Length](#) rule file parameter to the rule file Precision by default.

Acknowledgments

OK., ERROR(117)

Examples

Assuming the default Precision of 1000 and the default Unit Length of 1e-06, this command specifies the default output values:

```
AGF UNITS 0.001 1e-09
```

Related Topics

[Annotated Geometry Format Commands](#)

[MAGNIFY RESULTS](#)

[AGF WRITE](#)

AGF WRITE

CCI command. Corresponding Tcl shell command: [qs::write_agf](#).

Writes an annotated geometry format (AGF) file.

Usage

AGF WRITE *filename*

Alternate Command Name

GDS WRITE

Arguments

- *filename*

A require pathname of a file. If the *filename* ends in the .Z or the .gz suffix, the output file is compressed using the compress or gzip utility, respectively. Using compressed output suffixes assumes you have the appropriate utilities in your environment. Directories in a pathname are created if they do not exist.

Description

Writes layout geometry in AGF to the specified *filename*. Other commands in the AGF family configure the output before AGF WRITE is used.

The layout format used for the output design is specified by [AGF FORMAT](#). The default is GDSII.

By default, device seed shapes are fully merged on output by AGF WRITE. However, device seed layers might not be merged if the ORIGINAL keyword of [AGF SEED PROPERTY](#) is used in the flow. The Calibre device extraction module does not require pin layers to be merged in order to extract a device successfully. However, certain parasitic extraction tools do require merged pin layers in order to generate accurate results. The [AGF MERGE LAYER](#) and [AGF MERGE PIN LAYERS](#) commands can be used to manage layer merging in the AGF output.

For GDS output, device instance names, net IDs, and placement names are written to property records PROPATTR and PROPVVALUE. By default, the PROPATTR record value is 1 for all properties. The value for PROPATTR records may be changed using the [AGF {DEVPROP | NETPROP | PLACEPROP} NUMBER](#) command.

For OASIS output, device instance names, net IDs, and placement names are written to property records PROPNAME and PROPSSTRING. The value for the PROPNAME records may be changed using the [AGF {DEVPROP | NETPROP | PLACEPROP} NAME](#) command.

Device seed shape annotations are configured through the [AGF SEED PROPERTY](#) command.

Layer mapping information for the AGF layout is made available through the [AGF MAP](#) command.

By default, the maximum vertex count is 200 (same as the GDSII standard). It may be adjusted with the [MAXIMUM VERTEX COUNT](#) command.

By default, the database precision is the reciprocal of the rule file [Precision](#) setting. The database precision can be changed with the [AGF UNITS](#) command.

This command requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license. Refer to the [Calibre Administrator's Guide](#) for license information.

The AGF file is used in conjunction with the outputs of the [LAYOUT NETLIST WRITE](#) and [LAYOUT NAMETABLE WRITE](#) commands.

The [AGF HIERARCHY](#) and [AGF RESET](#) are also frequently useful in the context of using AGF WRITE.

Filters

These commands control filtering of layer output:

FILTER LAYERS	FILTER {IN OUT} LAYERS
FILTER DEVICELAYERS	FILTER DEVICES
AGF FILTER BOX LAYERS	AGF FILTER {IN OUT} BOX LAYERS
AGF CLEAR FILTER BOX LAYERS	AGF RESET

See “[Layer Output Control for AGF Files](#)” on page 527 for details about layer output control.

Response

None. Writes an AGF file.

Acknowledgments

OK., NOK(33), ERROR(102), ERROR(103), ERROR(130)

Related Topics

- [Annotated Geometry Format Commands](#)
- [Generating CCI Output Files](#)
- [Results Transformation Commands](#)

Cell Extents Report Command

The CELL EXTENTS WRITE command (qs::write_cell_extents in the Tcl shell) creates a file listing the coordinates for the rectangular extents of cells in the PHDB, in each cell's coordinate space.

A basic script for generating all CCI files needed for downstream flows is discussed under “[Generating CCI Output Files](#)” on page 480.

CELL EXTENTS WRITE **566**

CELL EXTENTS WRITE

CCI command. Corresponding Tcl shell command: [qs::write_cell_extents](#).

Writes a cell extents file.

Usage

CELL EXTENTS WRITE *filename*

Arguments

- *filename*

A required pathname of a file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

Description

Writes the extent of each cell in cell coordinate space to the specified *filename*.

Requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license. Refer to the [Calibre Administrator's Guide](#) for license information.

Response

Writes a cell extent report. Format is as follows:

```
# SVDB: Cell Extents (File format 1)
# SVDB: Layout Primary mix
# SVDB: Rules -0 rules Mon Mar  3 16:09:17 2003
# SVDB: GDSII -0 mix.gds Fri Oct  9 12:19:16 1998
# SVDB: SNL -0 mix.net.src
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
mix -134000 -102000 80000 32000
nand -40000 -80000 30000 42000
inv -4000 -18000 23000 18000
```

The names of the cells follow the SVDB header information. Each cell name is followed by the x y coordinates of the lower-left and upper-right corners of the cell extent.

Acknowledgments

OK., ERROR(101), ERROR(102)

Related Topics

[Results Transformation Commands](#)

Generating CCI Output Files

Customized Layout SPICE Netlist Commands

The LAYOUT NETLIST WRITE command writes a hierarchical SPICE netlist representing network connectivity in the AGF file written by AGF WRITE. The netlist and the AGF file are designed to be used together. The layout netlist is configured with the commands in the LAYOUT NETLIST family.

To generate a layout netlist, your rule file must include the Mask SVDB Directory statement with the NETLIST, GDSII, or CCI keyword.

A basic script for generating all CCI files needed for downstream flows is discussed under “[Generating CCI Output Files](#)” on page 480.

The following table lists the layout netlist family of commands in CCI. The corresponding command in the Query Server Tcl shell for configuring a layout netlist is [dfm::set_layout_netlist_options](#).

Table 6-7. Customized Layout SPICE Netlist Commands

Command	Description
LAYOUT NETLIST ANNOTATED DEVICES	Controls whether DFM property annotated devices are used in generating a SPICE netlist.
LAYOUT NETLIST CELL PREFIX	Specifies a prefix for subcircuit names in the output netlist.
LAYOUT NETLIST NO CELL PREFIX	Overrides LAYOUT NETLIST CELL PREFIX.
LAYOUT NETLIST COMMENT PROPERTIES	Specifies whether non-standard SPICE properties are written as comments.
LAYOUT NETLIST DEVICE LOCATION	Controls device location information in the output netlist.
LAYOUT NETLIST DEVICE LOWERCASE	Controls the text case used for device names.
LAYOUT NETLIST DEVICE TEMPLATES	Controls output of device template information in the output netlist.
LAYOUT NETLIST EMPTY CELLS	Specifies whether calls are made to empty subcircuits.
LAYOUT NETLIST HIERARCHY	Specifies the type of hierarchy to use in the output netlist.
LAYOUT NETLIST HIERARCHY CLEAR EXPAND CELLS	Clears the LAYOUT NETLIST HIERARCHY EXPAND CELL list.
LAYOUT NETLIST HIERARCHY EXPAND CELL	Expands the specified cell by one level in an output layout netlist.

Table 6-7. Customized Layout SPICE Netlist Commands (cont.)

Command	Description
LAYOUT NETLIST HIERARCHY PREFIX	Specifies whether to write a cell prefix for expanded cell names.
LAYOUT NETLIST HSPICE CR	Specifies whether capacitor and resistor element model names use the comment-coded format.
LAYOUT NETLIST HSPICE USER	Specifies whether user-defined devices use model names as subcircuit reference names.
LAYOUT NETLIST NAMES	Defines the origin of the net names used in the output netlist.
LAYOUT NETLIST PIN LOCATIONS	Controls the output of pin location information.
LAYOUT NETLIST PORT PADS	Controls the output of port information comments.
LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS	Specifies whether primitive subcircuits are written in the output netlist.
LAYOUT NETLIST REPORT BAD ANNOTATED DEVICES	Reports bad DFM property annotated devices.
LAYOUT NETLIST RESET	Returns the CCI netlist generation system to its default values.
LAYOUT NETLIST SEPARATED PROPERTIES	Controls the output of separated properties to the CCI netlist.
LAYOUT NETLIST STRING PROPERTIES	Specifies whether string properties are written to the output netlist.
LAYOUT NETLIST TRIVIAL PINS	Controls the writing of trivial pins to the output netlist.
LAYOUT NETLIST UNIQUE NAMES	Controls the output of reduced layout instance and net names.
LAYOUT NETLIST WRITE dfm::write_spice_netlist	Writes the customized layout netlist from the CCI system.
LAYOUT SEPARATED PROPERTIES WRITE qs::write_separated_properties	Writes separated (PDSP) properties to a file.
STATUS LAYOUT NETLIST dfm::list_layout_netlist_options	Returns current layout netlist generation settings.

LAYOUT NETLIST ANNOTATED DEVICES

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -annotated_devices.`

Controls whether DFM property annotated devices are used in generating a SPICE netlist.

Usage

LAYOUT NETLIST ANNOTATED DEVICES {NO | YES}

Arguments

- **NO**
A keyword that specifies annotated devices are not used. This is the default.
- **YES**
A keyword that specifies annotated devices are used.

Description

Controls whether the [LAYOUT NETLIST WRITE](#) command generates a SPICE netlist based upon DFM property annotated device layers in place of the originally-recognized device layers.

To use the **YES** keyword, the [LVS Annotate Devices](#) specification statement must be present in the rule file. When **YES** is specified, the annotated devices are represented as promoted device paths in higher-level circuits of the design. Pins of the promoted devices may also require new pins that route through the design.

For example, an original netlist containing two resistors might look like this:

```
.SUBCKT RCELL 1 2
R0 1 3 2 40
R1 1 3 2 50
.ENDS

.SUBCKT TOP
X0 A B RCELL
.ENDS
```

Suppose that a property P can be measured using [DFM Property](#) and you want to create a netlist with P properties annotated for some downstream tool. Furthermore, assume the resistor R0 must be promoted to the top level in order to make a meaningful measurement for P. However, R1 can be annotated in place inside cell RCELL.

After netlist extraction and use of the LAYOUT NETLIST ANNOTATED DEVICES YES command in the Query Server, the following netlist might result:

```
.SUBCKT RCELL 1 2
RR1 1 3 2 50 $P=1.8e-07
.ENDS

.SUBCKT TOP
RX0/R0 A B X0/3 40 $P=2.3e-06
X0 A B RCELL X0/3
.ENDS
```

Notice that the resistor R0 has been promoted to cell TOP. R0 retains its original hierarchical pathname (with an R prefixed in order to remain as a valid SPICE netlist). A hierarchical net path X0/3 has appeared in cell TOP and is routed down through to RCELL in order to provide proper connectivity between the two resistors.

Note

 LAYOUT NETLIST ANNOTATED DEVICES YES is mutually exclusive with [LAYOUT NETLIST SEPARATED PROPERTIES YES](#).

Acknowledgments

OK., OK: LAYOUT NETLIST SEPARATED PROPERTIES was set to NO, NOK(57)

Related Topics

[LAYOUT NETLIST REPORT BAD ANNOTATED DEVICES](#)

[Customized Layout SPICE Netlist Commands](#)

[Annotated Device Commands](#)

LAYOUT NETLIST CELL PREFIX

CCI command. Corresponding Tcl shell command:
`dfm::set_layout_netlist_options -cell_prefix`.

Configures the LAYOUT NETLIST WRITE command to add the specified prefix in front of all cell subcircuit names in the netlist. This applies to subcircuit calls and .SUBCIRCUIT statements.

Usage

LAYOUT NETLIST CELL PREFIX *prefix*

Arguments

- *prefix*

A required string used as a cell name prefix.

Acknowledgments

OK.

Examples

A LAYOUT NETLIST CELL PREFIX P1_ command would cause this output:

```
.SUBCKT sub GROUND VCC
```

to become this:

```
.SUBCKT P1_sub GROUND VCC
```

and this subcircuit call:

```
X8 GROUND VCC sub $T=-104000 2000 0 0 $X=-108000 $Y=-16000
```

to become this:

```
X8 GROUND VCC P1_sub $T=-104000 2000 0 0 $X=-108000 $Y=-16000
```

Related Topics

[LAYOUT NETLIST NO CELL PREFIX](#)

[Customized Layout SPICE Netlist Commands](#)

[LAYOUT NETLIST WRITE](#)

LAYOUT NETLIST NO CELL PREFIX

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -no_cell_prefix.`

Cancels the current LAYOUT NETLIST CELL PREFIX command.

Usage

LAYOUT NETLIST NO CELL PREFIX

Arguments

None.

Acknowledgments

OK.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

[LAYOUT NETLIST CELL PREFIX](#)

LAYOUT NETLIST COMMENT PROPERTIES

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -comment_properties.`

Controls whether all non-standard properties are written in comment format to the output netlist. *Non-standard* means not standard in regular SPICE syntax.

Usage

LAYOUT NETLIST COMMENT PROPERTIES {NO | YES}

Arguments

- **NO**

A keyword that specifies non-standard properties are not written. This is the default. This keyword may not be used with LVS Netlist Builtin Property Conversion QD in the rule file.

- **YES**

A keyword that specifies non-standard properties are written.

Acknowledgments

OK., ERROR(175)

Examples

Using **YES** causes devices normally written as this:

```
C6 22 25 5 areac=9e-012 $SUB=42 $X=-7000 $Y=0
C7 23 26 5 areac=9e-012 $X=-7000 $Y=9000
```

to be written as this:

```
C6 22 25 5 $SUB=42 $X=-7000 $Y=0 $areac=9e-012
C7 23 26 5 $X=-7000 $Y=9000 $areac=9e-012
```

Property `areac` is not standard for capacitors in SPICE, hence it is written out as a comment.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST DEVICE LOCATION

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -device_location.`

Controls device location information in the output netlist.

Usage

LAYOUT NETLIST DEVICE LOCATION {VERTEX | CENTER}

Arguments

- **VERTEX**

A keyword that specifies a device shape's lower-left vertex is used for the device location. This is the default. If [LVS Center Device Location YES](#) is specified in the rule file, this keyword may not be used.

- **CENTER**

A keyword that specifies the centroid of a shape is used for the device location. If the centroid lies outside the polygon (it is concave), then a vertex is arbitrarily chosen.

Description

Controls the netlist device locations generated by the [LAYOUT NETLIST WRITE](#) command. The netlist is generated with \$X= \$Y= comments to indicate the location of the device seed shape.

This command requires use of either the CCI or ANNOTATE DEVICES keyword of the [Mask SVDB Directory](#) rule file statement.

See also [LAYOUT NETLIST PIN LOCATIONS](#).

Acknowledgments

OK., ERROR(130)

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST DEVICE LOWERCASE

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -device_lowercase.`

Controls the text case used for device names.

Usage

LAYOUT NETLIST DEVICE LOWERCASE {NO | YES}

Arguments

- **NO**

A keyword that specifies normal netlisting behavior is used for device name text case. This is the default.

- **YES**

A keyword that specifies device names are written in lower case.

Description

Controls whether all device element names, model names, and pin names are written in lowercase letters. See “[Customized SPICE Netlist Format](#)” on page 497 for netlist conventions.

The **YES** keyword causes the special names to be lowercase regardless of the [LVS Downcase Device](#) setting in the rule file.

Acknowledgments

OK.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST DEVICE TEMPLATES

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -device_templates.`

Controls output of device template information in the output netlist.

Usage

LAYOUT NETLIST DEVICE TEMPLATES {NO | YES}

Arguments

- **NO**

A keyword that specifies device template information is not written. This is the default.

- **YES**

A keyword that specifies device template information is written.

Description

Controls whether the netlist generated by the [LAYOUT NETLIST WRITE](#) command provides rule file [Device](#) template information similar to [LAYOUT NETLIST PIN LOCATIONS YES](#), but without the \$PIN_XY data.

The **YES** keyword is useful for using pushdown separated properties (PDSP) when pin location information is not desired in the generated netlist (the information is still provided in the PDSP file). See “[Customized Files for Pushdown Separated Properties \(PDSP\) Flow](#)” on page 514 for more information about this flow.

The **YES** keyword instructs the Query Server to write device template info in the form of SPICE comment coded extensions that are used by downstream tools. The header of the CCI netlist file contains this comment indicating that device template information was requested:

* Device templates: YES

Otherwise, this comment appears:

* Device templates: NO

If both [LAYOUT NETLIST PIN LOCATIONS YES](#) and [LAYOUT NETLIST DEVICE TEMPLATES](#) are specified, the former command’s format is used for the output.

Response

The device template comment coded extension has the following format:

```
* .DEVTMPLT <d> <el>([<mod>])<seed> <pl_1>(<pn_1>) [<pl_2>(<pl_2>...)]  
* .DEVTMPLT // indicates the device template comment record.  
<d> // represents the device template number. each device is  
// identified with a $D=<n> comment field that matches  
// it with a specific DEVTMPLT. this is similar to what  
// the extracted SPICE netlist provides.  
<el> // element name specified in the rule file DEVICE  
// statement.  
<mod> // model name, when present, specified in the rule  
// file DEVICE statement.  
<seed> // device seed layer specified in the DEVICE statement.  
<pl_n> // nth device pin layer specified in the DEVICE  
// statement.  
<pn_n> // nth device pin name specified in the DEVICE statement.
```

If a template device is not selected for circuit extraction, this warning is appended to the extension's line:

```
** WARNING: This device was not selected for extraction.
```

Acknowledgments

OK., NOK(40)

Examples

```
* .DEVTMPLT 0 mp() PSEED GPIN(g) SDPIN(s) SDPIN(d)
```

This line represents pin layer and ordering information for all netlist devices that include the \$D=0 comment. It represents an MP device with no model name. The seed layer is PSEED, the pin layers are GPIN, SDPIN, and SDPIN for the pin names g, s, and d, respectively. The following device instance in the netlist:

```
M0 2 4 1 p L=5e-06 W=3e-06 $X=-6000 $Y=1000 $D=0
```

corresponds with *.DEVTMPLT 0.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST EMPTY CELLS

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -empty_cells.`

Controls whether the netlist generated by the LAYOUT NETLIST WRITE command contains calls to empty cells, like via cells. This command does not pertain to primitive subcircuits for user-defined devices.

Usage

LAYOUT NETLIST EMPTY CELLS {NO | YES}

Arguments

- **NO**
A keyword that specifies empty subcircuits are not written. This is the default.
- **YES**
A keyword that specifies empty subcircuits are written.

Acknowledgments

OK.

Related Topics

[LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS](#)

[Customized Layout SPICE Netlist Commands](#)

[LAYOUT NETLIST WRITE](#)

LAYOUT NETLIST HIERARCHY

CCI command. Corresponding Tcl shell command: [dfm::set_layout_netlist_options -hierarchy](#).
Specifies the type of hierarchy to use in the output netlist.

Usage

LAYOUT NETLIST HIERARCHY {ALL | FLAT | AGF | HCELL}

Arguments

- **ALL**
A keyword that specifies no cells are expanded. This is the default.
- **FLAT**
A keyword that specifies all cell placements are expanded into the top-level cell.
- **AGF**
A keyword that specifies output is consistent with the default output of the [AGF WRITE](#) command.
- **HCELL**
A keyword that specifies all non-hcells are expanded into the containing hcell. It also expands unbalanced hcells that were expanded during LVS comparison.

Description

Specifies the netlist hierarchy generated by the [LAYOUT NETLIST WRITE](#) command. The netlist may be generated with certain cells expanded.

If [LAYOUT NETLIST NAMES SOURCE](#) is not specified, the layout netlist is written with all cells by default. If [LAYOUT NETLIST NAMES SOURCE](#) is specified, the layout netlist is written flat for the **ALL**, **FLAT**, and **AGF** options. The [LAYOUT NETLIST HIERARCHY](#) command with the corresponding keywords is invalid in this situation.

If **ALL** is not specified, then the [LAYOUT NAMETABLE WRITE](#) command should be used with the [EXPAND_CELLS](#) keyword.

Acknowledgments

OK., ERROR(122), ERROR(128)

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST HIERARCHY CLEAR EXPAND CELLS

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -clear_hierarchy_expand_cells.`

Clears the cell expansion list generated by the LAYOUT NETLIST HIERARCHY EXPAND CELL command.

Usage

LAYOUT NETLIST HIERARCHY CLEAR EXPAND CELLS

Arguments

None.

Acknowledgments

OK.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

[LAYOUT NETLIST HIERARCHY EXPAND CELL](#)

LAYOUT NETLIST HIERARCHY EXPAND CELL

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -hierarchy_expand_cells.`

Expands the specified cell by one level in an output layout netlist.

Usage

LAYOUT NETLIST HIERARCHY EXPAND CELL *cellname*

Arguments

- *cellname*

A required name of a cell. The *cellname* parameter can contain asterisk (*) wildcards. The * matches zero or more characters.

Acknowledgments

OK., NOK(36)

Related Topics

[LAYOUT NETLIST HIERARCHY CLEAR EXPAND CELLS](#)

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST HIERARCHY PREFIX

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -hierarchy_prefix.`

Specifies whether to write the prefix for expanded names as specified by the LAYOUT NETLIST HIERARCHY setting.

Usage

LAYOUT NETLIST HIERARCHY PREFIX {YES | NO}

Arguments

- **YES**

A keyword that specifies a prefix is written for expanded names. This is the default.

- **NO**

A keyword that specifies a prefix is not written for expanded names.

Description

This command is used for third-party tool compatibility.

The **NO** keyword creates an invalid SPICE netlist because the type of element is no longer designated by the first character in the line. However, some tools prefer not to have the extra character in the netlist. For example, the following netlist might be created by the **LAYOUT NETLIST WRITE** command with **LAYOUT NETLIST HIERARCHY** set to FLAT. By default, the netlist is generated with the extra call prefix characters, as shown here:

```
LAYOUT NETLIST HIERARCHY FLAT
LAYOUT NETLIST WRITE net.spi

* Calibre Connectivity Interface Layout netlist
* ...
* Hierarchy: FLAT
* ...
*****SUBCKT mix GROUND VCC
** N=7 EP=2 IP=11 FDC=11
MM0 VCC 4 5 VCC p L=3e-06 W=1.3e-05 $X=47000 $Y=-2000
MM1 3 2 GROUND GROUND n L=1e-06 W=9e-06 $X=-52000 $Y=-35000
MM2 5 4 GROUND GROUND n L=2e-06 W=1e-05 $X=49000 $Y=-42000
MX3/M0 X3/11 X3/IN X3/10 VCC p L=1e-06 W=1.8e-05 $X=-101000 $Y=-48500
MX3/M1 X3/9 X3/IN X3/8 GROUND n L=1e-06 W=1.8e-05 $X=-101000 $Y=-68500
MX4/M0 X4/11 X4/IN X4/10 VCC p L=1e-06 W=1.8e-05 $X=-101000 $Y=11500
MX4/M1 X4/9 X4/IN X4/8 GROUND n L=1e-06 W=1.8e-05 $X=-101000 $Y=-8500
MX5/M0 VCC 2 3 VCC p L=1e-06 W=1e-05 $X=-20000 $Y=-1500
MX5/M1 VCC 7 3 VCC p L=1e-06 W=1e-05 $X=0 $Y=2500
MX5/M2 X5/6 7 GROUND GROUND n L=1e-06 W=1e-05 $X=-10000 $Y=-72500
MX5/M3 3 2 X5/6 GROUND n L=1e-06 W=1e-05 $X=-10000 $Y=-51500
.ENDS
*****
```

If the hierarchy prefix keyword is set to **NO**, the extra call character at the beginning of each line is omitted:

```
LAYOUT NETLIST HIERARCHY PREFIX NO
LAYOUT NETLIST WRITE net.spi

* Calibre Connectivity Interface Layout netlist
* ...
* Hierarchy: FLAT
* ...
* Hierarchy Prefix: NO
* ...
*****  
  
.SUBCKT mix GROUND VCC
** N=7 EP=2 IP=11 FDC=11
M0 VCC 4 5 VCC p L=3e-06 W=1.3e-05 $X=47000 $Y=-2000
M1 3 2 GROUND GROUND n L=1e-06 W=9e-06 $X=-52000 $Y=-35000
M2 5 4 GROUND GROUND n L=2e-06 W=1e-05 $X=49000 $Y=-42000
X3/M0 X3/11 X3/IN X3/10 VCC p L=1e-06 W=1.8e-05 $X=-101000 $Y=-48500
X3/M1 X3/9 X3/IN X3/8 GROUND n L=1e-06 W=1.8e-05 $X=-101000 $Y=-68500
X4/M0 X4/11 X4/IN X4/10 VCC p L=1e-06 W=1.8e-05 $X=-101000 $Y=11500
X4/M1 X4/9 X4/IN X4/8 GROUND n L=1e-06 W=1.8e-05 $X=-101000 $Y=-8500
X5/M0 VCC 2 3 VCC p L=1e-06 W=1e-05 $X=-20000 $Y=-1500
X5/M1 VCC 7 3 VCC p L=1e-06 W=1e-05 $X=0 $Y=2500
X5/M2 X5/6 7 GROUND GROUND n L=1e-06 W=1e-05 $X=-10000 $Y=-72500
X5/M3 3 2 X5/6 GROUND n L=1e-06 W=1e-05 $X=-10000 $Y=-51500
.ENDS
*****
```

Acknowledgments

OK., ERROR(122)

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST HSPICE CR

CCI command. Corresponding Tcl shell command: [dfm::set_layout_netlist_options -hspice_cr](#).
Controls whether model names in capacitor and resistor elements use the standard HSPICE model name field instead of the default comment-coded format.

Usage

LAYOUT NETLIST HSPICE CR {NO | YES}

Arguments

- **NO**
A keyword that specifies to use the comment-coded form of model name. This is the default.
- **YES**
A keyword that specifies to use the HSPICE model name.

Acknowledgments

OK.

Examples

Setting LAYOUT NETLIST HSPICE CR YES causes devices normally written as this:

```
C6 22 25 5 areac=9e-012 $SUB=42 $[capa] $X=-7000 $Y=0
C7 23 26 5 areac=9e-012 $[capa] $X=-7000 $Y=9000
```

to be written as this:

```
C6 22 25 capa 5 areac=9e-012 $SUB=42 $X=-7000 $Y=0
C7 23 26 capa 5 areac=9e-012 $X=-7000 $Y=9000
```

Notice the change in how the model name capa is represented.

Related Topics

[LAYOUT NETLIST HSPICE USER](#)

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST HSPICE USER

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -hspice_user.`

Controls whether user-defined devices use model names as subcircuit reference names in addition to the comment-coded format.

Usage

LAYOUT NETLIST HSPICE USER {NO | YES}

Arguments

- **NO**

A keyword that specifies not to use model names as subcircuit reference names. This is the default.

- **YES**

A keyword that specifies to use model names as subcircuit reference names in addition to comment-coded names.

Acknowledgments

OK.

Examples

Specifying **YES** causes devices normally written as this:

```
X6 22 25 27 5 C $[capa] $X=-7000 $Y=0
X7 23 26 27 5 C $[capa] $X=-7000 $Y=9000
```

to be written as this:

```
X6 22 25 27 5 capa $[capa] $X=-7000 $Y=0
X7 23 26 27 5 capa $[capa] $X=-7000 $Y=9000
```

Related Topics

[LAYOUT NETLIST HSPICE CR](#)

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST NAMES

CCI command. Corresponding Tcl shell command: [dfm::set_layout_netlist_options -names](#).

Defines the origin of the net names used in the output netlist.

Usage

LAYOUT NETLIST NAMES {LAYOUT | NONE | SOURCE} [NETS | INSTANCES]

Arguments

- **LAYOUT**

A keyword that specifies net and instance names are output from the layout design, when names are available. Otherwise, numbers are output. This is the default.

- **NONE**

A keyword that specifies node and instance numbers are output from the layout design.

- **SOURCE**

A keyword that specifies net and instance names that correspond to layout names are output from the source design, when names are available. Otherwise, numbers are used.

- **NETS**

Optional keyword that causes net names to be output. Instance numbers are output instead of names. May not be explicitly specified with **INSTANCES**. Used by default.

- **INSTANCES**

Optional keyword that causes instance names to be output. Node numbers are output instead of names. May not be explicitly specified with **NETS**. Used by default.

Description

Controls the netlist names generated by the [LAYOUT NETLIST WRITE](#) command.

The **LAYOUT** keyword is frequently preferred when generating a Layout Net Cross-Reference (LNXF) file with [LAYOUT NET XREF WRITE](#).

The **NONE** keyword is used together with [LAYOUT NAMETABLE WRITE](#) to generate a Layout Netlist Names (LNN) file that maps net and instance numbers to names. The **NONE** setting causes this note to be issued:

NOTE: Backward compatibility mode used for LNXF file generation: LAYOUT NETLIST NAMES NONE was set.

when LAYOUT NET XREF WRITE is used.

Unmatched nets or instances are identified using a prefix when **SOURCE** is specified. The prefix is typically `_Layout_`, but if there are source names that begin with `_Layout_`, additional `_` characters are added to the prefix to insure that it is unique.

If LAYOUT NETLIST NAMES **SOURCE** is executed and **LAYOUT NETLIST HIERARCHY** is set to ALL, the Query Server automatically changes the ALL setting to FLAT and issues a note to that effect.

The NETS and INSTANCES keywords are used to configure the corresponding names separately.

Acknowledgments

OK., ERROR(111), ERROR(129)

Related Topics

[LAYOUT NETLIST UNIQUE NAMES](#)

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST PIN LOCATIONS

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -pin_locations.`

Controls the output of pin location information.

Usage

LAYOUT NETLIST PIN LOCATIONS {NO | YES}

Arguments

- **NO**

A keyword that specifies pin location information is not output. This is the default.

- **YES**

A keyword that specifies pin location information is output.

Description

Controls whether the netlist generated by the [LAYOUT NETLIST WRITE](#) command provides a \$PIN_XY comment after each device instance showing the coordinates of the pins. *.DEVTMPLT entries and IDs match those of the [DEVICE TABLE](#) command.

The pin location output of the YES keyword corresponds to the [LVS Center Device Pins](#) statement setting in the rule file. By default, the reported pin locations are chosen arbitrarily at the intersection of device seed shapes and pins.

The YES keyword should not be used for pushdown separated properties (PDSP) flows. Use [LAYOUT NETLIST DEVICE TEMPLATES YES](#) instead.

Response

The device template has the following format:

```
*.DEVTMPLT <dt> <el>([<mod> ] )<seed> <pl_1>(<pn_1>) [<pl_2>(pl_2)...]  
*.DEVTMPLT // indicates the device template comment record.  
<dt> // represents the device template number. each device is  
// identified with a $dt=<n> comment field that matches  
// it with a specific *.DEVTMPLT.  
<el> // element name specified in the rule file DEVICE statement.  
<mod> // model name, when present, specified in the rule  
// file DEVICE statement.  
<seed> // device seed layer specified in the DEVICE statement.  
<pl_n> // nth device pin layer specified in the DEVICE statement.  
<pn_n> // nth device pin name specified in the DEVICE statement.  
// device instances in this netlist have the  
// following additional comment line:  
<regular_device_record>  
+$dt=<d> $PIN_XY=X1,Y1[,X2,Y2 ...]  
$dt=<d> // shows that this is a device corresponding to *.DEVTMPLT <d>  
$PIN_XY= // specifies ordered X,Y coordinates for each pin in  
// SPICE netlist order.
```

Acknowledgments

OK.

Examples

This example shows the output when LAYOUT NETLIST PIN LOCATIONS YES is specified.

```
*.DEVTMPLT 0 mp() PSEED GPIN(g) SDPIN(s) SDPIN(d)
```

This line represents pin layer and ordering information for all netlist devices that include the \$dt=0 comment. It represents an MP device with no model name. The seed layer is PSEED, the pin layers are GPIN, SDPIN, and SDPIN for the pin names g, s, and d, respectively. The following device instance in the netlist:

```
M0 2 4 1 p L=5e-06 W=3e-06 $X=-6000 $Y=3500
+$dt=0 $PIN_XY=-4000,3500,-6000,3500,-6000,3500
```

corresponds with *.DEVTMPLT 0. Pin locations for this device are as follows (recall that SPICE syntax puts standard MOS pins in the order d g s):

```
SDPIN(d) connected to net 2: (-4000,2500)
GPIN(g) connected to net 4: (-6000,3500)
SDPIN(s) connected to net 1: (-6000,3500)
```

Related Topics

[LAYOUT NETLIST DEVICE LOCATION](#)

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST PORT PADS

CCI command. Corresponding Tcl shell command: [dfm::set_layout_netlist_options -port_pads](#).
Controls the output of port information comments.

Usage

LAYOUT NETLIST PORT PADS {NO | YES}

Arguments

- **NO**

A keyword that specifies port information is not output. This is the default.

- **YES**

A keyword that specifies port information is output.

Description

Controls whether the netlist generated by the [LAYOUT NETLIST WRITE](#) command contains “* PORT” comments for each cell containing port objects. Ports are declared in the rules using [Port Layer Text](#) or [Port Layer Polygon](#).

Information provided by this command is similar to that discussed under “[Port Table File Format](#)” on page 504.

Response

The netlist comments written by the command have this form:

```
* PORT <port_name> <net_name> <x> <y> <layer>
```

If the port object is a Port Layer Polygon object, then the *port_name* is <UNNAMED>, otherwise a user-given name appears. The *net_name* is either a user name or a numeric net ID. The *net_name* and *port_name* may differ, but often are the same. The [LAYOUT NETLIST NAMES](#) command can impact the net names that are used. The coordinates are in cell space, and the port’s layer is given.

Acknowledgments

OK.

Examples

This is an example subcircuit when YES is specified. Notice the “* PORT” comment lines.

Calibre Connectivity Interface
Customized Layout SPICE Netlist Commands

```
.SUBCKT inv VSS VDD IN Y
** N=5 EP=4 IP=0 FDC=2
* PORT VSS VSS 1300 3600 metal1
* PORT VDD VDD 1300 82400 metal1
* PORT IN IN 4100 36900 metal1
* PORT Y Y 12000 34000 metal1
M0 Y IN VSS VSS n L=1.25e-06 W=1.5e-05 $X=5375 $Y=14000 $D=0
M1 Y IN VDD VDD p L=1.75e-06 W=2e-05 $X=5125 $Y=51000 $D=1
.ENDS
```

LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -primitive_device_subckts.`

Controls whether the netlist generated by the LAYOUT NETLIST WRITE command provides primitive device subcircuits for user-defined devices.

Usage

LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS {YES | NO}

Arguments

- **YES**

A keyword that specifies primitive device subcircuits are written. This is the default.

- **NO**

A keyword that specifies primitive device subcircuits are not written.

Acknowledgments

OK.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

[LAYOUT NETLIST EMPTY CELLS](#)

[LAYOUT NETLIST WRITE](#)

LAYOUT NETLIST REPORT BAD ANNOTATED DEVICES

CCI command.

Reports bad DFM property annotated devices.

Usage

LAYOUT NETLIST REPORT BAD ANNOTATED DEVICES *filename*

Arguments

- *filename*

A required pathname of an output file. Directories in a pathname are created if they do not exist.

Description

Writes a bad DFM property annotated devices report to the *filename*. The format is similar to the circuit extraction report and contains bad device instances.

When using [LVS Annotate Devices](#), it is sometimes necessary to output device instances that are not found on the annotated layer. Instances associated with the annotated layer that do not interact with the layer are written to the *filename*. These instances are not output by [LAYOUT NETLIST WRITE](#).

Acknowledgments

OK., NOK(33), ERROR(130), ERROR(131)

Related Topics

[LAYOUT NETLIST ANNOTATED DEVICES](#)

[Annotated Device Commands](#)

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST RESET

CCI command. Corresponding Tcl shell command: [dfm::set_layout_netlist_options -reset](#).

Returns the CCI netlist generation system to its default values.

Usage

LAYOUT NETLIST RESET

Arguments

None.

Description

Resets layout netlist generation information to default values. This command does not reset **FILTER DEVICES** (or FILTER DEVICENAMES). Use FILTER DEVICES ALL to reset the device filter.

Acknowledgments

OK.

Related Topics

[STATUS LAYOUT NETLIST](#)

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST SEPARATED PROPERTIES

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -separated_properties.`

Controls the output of separated properties to the CCI netlist.

Usage

LAYOUT NETLIST SEPARATED PROPERTIES {NO | YES}

Arguments

- **NO**

A keyword that specifies separated properties are not written. This is the default.

- **YES**

A keyword that specifies separated properties are written.

Description

Controls whether properties saved by the [LVS Push Devices](#) SEPARATE PROPERTIES YES specification statement are written to the output netlist from the [LAYOUT NETLIST WRITE](#) command. NO is recommended for the PDSP flow.

See “[Property Definitions and Netlist Behavior for CCI Flows](#)” on page 477 for key definitions and details about various setting combinations. Be aware that the NO and YES keywords correspond to the `dfm::set_layout_netlist_options -separated_properties` keywords of the same names.

LAYOUT NETLIST SEPARATED PROPERTIES YES is mutually exclusive with [LAYOUT NETLIST ANNOTATED DEVICES](#) YES.

The [LAYOUT SEPARATED PROPERTIES WRITE](#) command causes separated properties to be written to a specified file, which is distinct from the LAYOUT NETLIST WRITE command.

See “[Customized Files for Pushdown Separated Properties \(PDSP\) Flow](#)” on page 514 for details about producing PDSP files.

Acknowledgments

OK., NOK(58)

Examples

Assume this is in the rule file:

```
LVS PUSH DEVICES YES
LVS PUSH DEVICES SEPARATE PROPERTIES YES
MASK SVDB DIRECTORY QUERY CCI
```

These commands in the Query Server shell can be used to write a separated properties file *props.data* and a backannotation netlist *cci_pdsp.netlist*:

```
LAYOUT NETLIST DEVICE LOCATION CENTER
LAYOUT NETLIST TRIVIAL PINS YES
LAYOUT NETLIST EMPTY CELLS YES
LAYOUT NETLIST NAMES NONE
LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS NO
LAYOUT NETLIST SEPARATED PROPERTIES NO
LAYOUT NETLIST DEVICE TEMPLATES YES
LAYOUT NETLIST HIERARCHY AGF
LAYOUT SEPARATED PROPERTIES WRITE props.data
LAYOUT NETLIST WRITE cci_pdsp.netlist
```

Related Topics

[Customized Layout SPICE Netlist Commands](#)

[Writing PDSP Properties in CCI](#)

LAYOUT NETLIST STRING PROPERTIES

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -string_properties.`

Controls the output of string properties. String properties produced with the device property computation language are output by the LAYOUT NETLIST WRITE command by default.

Usage

LAYOUT NETLIST STRING PROPERTIES {YES | NO}

Arguments

- **YES**

A keyword that specifies string properties generated by a [Device](#) property computation program are output. This is the default.

- **NO**

A keyword that specifies string properties are not output.

Acknowledgments

OK.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

[LAYOUT NETLIST WRITE](#)

LAYOUT NETLIST TRIVIAL PINS

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -trivial_pins.`

Controls the writing of trivial pins to the output netlist.

Usage

LAYOUT NETLIST TRIVIAL PINS {NO | YES}

Arguments

- **NO**
A keyword that specifies trivial pins are not written. This is the default.
- **YES**
A keyword that specifies trivial pins are written.

Description

Controls whether the netlist generated by the [LAYOUT NETLIST WRITE](#) command provides pins not connected to devices.

The YES keyword overrides [LVS Netlist Box Contents NO](#), [LVS Netlist Unnamed Box Pins NO](#), and the [LVS Netlist Internally Floating Pins NO](#) settings in the rule file. The purpose of LAYOUT NETLIST TRIVIAL PINS YES is to ensure that every place there is a pin needed in the AGF file, the AGF has an associated pin in the CCI netlist. (That is, if appropriate, a pin is represented in the CCI netlist where the net appears in more than one level of hierarchy, including box cells.) This is because the AGF has no mechanism for storing pins other than the CCI netlist.

If LAYOUT NETLIST TRIVIAL PINS NO is used, then the rule file settings for the aforementioned specification statements have precedence. In other words, the CCI generated netlist matches the calibre -spice netlist in that case.

Acknowledgments

OK.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST UNIQUE NAMES

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -unique_names.`

Controls the output of reduced layout instance and net names.

Usage

LAYOUT NETLIST UNIQUE NAMES {{NO | YES} [INSTANCES] [NETS]}

Arguments

- **NO**

A keyword that specifies reduced (smashed) layout net or instance names that are mapped to source names are not output as unique.

- **YES**

A keyword that specifies reduced (smashed) layout net or instance names that are mapped to source names are output as unique.

- **INSTANCES**

An optional keyword that causes instance names to be affected by the NO or YES setting.

- **NETS**

An optional keyword that causes net names to be affected by the NO or YES setting.

Description

Controls the output format of reduced (or smashed) layout instance and net names that are mapped to source names.

The defaults are these:

```
LAYOUT NETLIST UNIQUE NAMES NO NETS
LAYOUT NETLIST UNIQUE NAMES YES INSTANCES
```

Specifying only NO or YES without the INSTANCES or NETS options means the setting applies to both nets and instances.

The YES keyword causes combined layout instance and net names to appear in the form:

```
<element>_Layout_<l_name>_Source_<s_name>
```

where *<l_name>* is the layout name and *<s_name>* is the source name. For example, two layout resistors in series: R0 1 2, R1 2 3 when mapped to a single source resistor RA, are named as follows:

```
RA 1 2
R_Layout_1_Source_RA 2 3
```

This scheme guarantees a valid netlist when smashing of resistors occurs. The NO keyword causes the name of the smashed layout net or instance to be mapped directly to the source name. Using the previous example of series resistors, this is the format:

```
RA 1 2  
RA 2 3
```

Smashing of nets simply changes pin names of the netlist, which does not cause the netlist to become invalid. Hence, the default for NETS is NO.

See also [LAYOUT NETLIST NAMES](#).

Acknowledgments

OK.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

LAYOUT NETLIST WRITE

CCI command. Corresponding Tcl shell command: [dfm::write_spice_netlist](#).

Writes the customized layout netlist from the CCI system.

Usage

LAYOUT NETLIST WRITE *filename*

Arguments

- *filename*

A required pathname of an output netlist. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

Description

Writes a SPICE netlist describing devices, placements, and net connectivity present in the layout to the specified *filename*. Most of the other LAYOUT NETLIST family of commands affect what appears in the output netlist. This command must be used in a generated SPICE netlist flow.

Details of the layout netlist format are included under “[Customized SPICE Netlist Format](#)” on page 497. By default, the netlist written uses similar conventions as ones produced by calibre -spice.

This command preserves floating nets created by [LVS Preserve Floating Top Nets](#) YES and [LVS Netlist All Texted Pins](#) YES specification statements. Net names, including floating nets (which must be named) and named pins, are not written if [LAYOUT NETLIST NAMES](#) NONE is used.

Output coordinates are in database units as represented in the Calibre database (not necessarily the same as the physical layout). When used, magnification factors in the rule file are applied to the coordinates.

The [FILTER DEVICES](#) or [FILTER DEVICENAMES](#) commands affect the output.

When flattening of the netlist is required, the LAYOUT NETLIST WRITE command writes a *.STRIP_TYPE_CHAR directive to the netlist, which instructs the parser to strip the leading character from all names read in after deciding on the device type, but before any further processing. For example, the resistor “ROUT” would be treated as a resistor, but with the name “OUT”. The directive applies to all SPICE read for the current circuit, even SPICE read in previously. This is used in Calibre PERC flows to strip the leading characters from device names.

The **MAGNIFY RESULTS** factor is multiplied by the following output values:

- \$X \$Y cell placement and device coordinates.
- \$T transform values for cell placements.
- \$PIN_XY coordinates when **LAYOUT NETLIST PIN LOCATIONS YES** is used.

Requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license.
Refer to the *Calibre Administrator's Guide* for license information.

Acknowledgments

OK., NOK(33), ERROR(101), ERROR(102)

The command may issue warnings to STDOUT.

Examples

See “[Generating CCI Output Files](#)” on page 480.

Related Topics

[Customized Layout SPICE Netlist Commands](#)

[STATUS LAYOUT NETLIST](#)

[Results Transformation Commands](#)

LAYOUT SEPARATED PROPERTIES WRITE

CCI command. Corresponding Tcl shell command: [qs::write_separated_properties](#).

Writes pushdown separated properties (PDSP) to a file.

Usage

LAYOUT SEPARATED PROPERTIES WRITE *filename*

Arguments

- *filename*

A required pathname of a file to which separated properties are written. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

Description

Specifies a file to which the properties written by [LVS Push Devices SEPARATE PROPERTIES YES](#) are saved. The LVS Push Devices SEPARATE PROPERTIES YES statement must be in the rule file for this command to be used.

The [MAGNIFY RESULTS](#) factor is multiplied by the following output values:

- \$X \$Y coordinates.
- \$PIN_XY coordinates when present.

The *filename* format is shown under “[Separated Properties File Format](#)” on page 509.

See “[Customized Files for Pushdown Separated Properties \(PDSP\) Flow](#)” on page 514 for more details about the commands used with the PDSP flow.

Acknowledgments

OK., NOK(68), ERROR(102), ERROR(122), ERROR(131), ERROR(150)

Examples

Assume this is in the rule file:

```
LVS PUSH DEVICES YES
LVS PUSH DEVICES SEPARATE PROPERTIES YES
MASK SVDB DIRECTORY QUERY CCI
```

These commands in the Query Server shell can be used to write a separated properties file *cci_props.data* and a backannotation netlist *cci.netlist*:

```
LAYOUT NETLIST DEVICE LOCATION CENTER
LAYOUT NETLIST TRIVIAL PINS YES
LAYOUT NETLIST EMPTY CELLS YES
LAYOUT NETLIST NAMES NONE
LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS NO
LAYOUT NETLIST SEPARATED PROPERTIES NO
LAYOUT NETLIST DEVICE TEMPLATES YES
LAYOUT NETLIST HIERARCHY AGF
LAYOUT SEPARATED PROPERTIES WRITE cci_props.data
LAYOUT NETLIST WRITE cci.netlist
TERMINATE
```

Related Topics

[LAYOUT NETLIST SEPARATED PROPERTIES](#)

[Customized Layout SPICE Netlist Commands](#)

[Writing PDSP Properties in CCI](#)

[Results Transformation Commands](#)

STATUS LAYOUT NETLIST

CCI command. Corresponding Tcl shell command: [dfm::list_layout_netlist_options](#).
Returns current layout netlist generation settings. This command is useful prior to using LAYOUT NETLIST WRITE to check the current configuration of the netlisting environment.

Usage

STATUS LAYOUT NETLIST

Arguments

None.

Response

- The following response occurs (comments are not part of the report and indicate the setting of the corresponding command):

```
Status Layout Netlist 1000
Entries:
0 0 12 May 27 16:25:46 2021
Type: SPICE
Hierarchy: ALL | FLAT | AGF | HCELL // LAYOUT NETLIST HIERARCHY
Empty cells: NO | YES // LAYOUT NETLIST EMPTY CELLS
Trivial pins: NO | YES // LAYOUT NETLIST TRIVIAL PINS
Device pin locations: NO | YES // LAYOUT NETLIST PIN LOCATIONS
String Properties: YES | NO // LAYOUT NETLIST STRING PROPERTIES
Primitive device subckts: YES | NO // LAYOUT NETLIST PRIMITIVE
// DEVICE SUBCKTS
HSPICE capacitors/resistors: NO | YES // LAYOUT NETLIST HSPICE CR
HSPICE user devices: NO | YES // LAYOUT NETLIST HSPICE USER
Comment Properties: NO | YES // LAYOUT NETLIST COMMENT
// PROPERTIES
Net name type: LAYOUT | NONE | SOURCE // LAYOUT NETLIST NAMES NETS
Instance name type: LAYOUT | NONE | SOURCE // LAYOUT NETLIST NAMES
// SOURCE
Cell name type: LAYOUT | NONE | SOURCE //
// dfm::set_layout_netlist_option -names
Device location: VERTEX | CENTER // LAYOUT NETLIST DEVICE LOCATION
Filter devices: (all) | <indices> // FILTER DEVICES or DEVICENAMES
Device lowercase: NO | YES // LAYOUT NETLIST DEVICE LOWERCASE
Device templates: NO | YES // LAYOUT NETLIST DEVICE TEMPLATES
Separated properties: NO | YES // LAYOUT NETLIST SEPARATED
// PROPERTIES
END OF RESPONSE
```

Acknowledgments

OK.

Related Topics

[LAYOUT NETLIST RESET](#)

[LAYOUT NETLIST WRITE](#)

[Customized Layout SPICE Netlist Commands](#)

Layout Netlist Names File Command

The Layout Netlist Names (LNN) file provides mapping between net numbers and user-specified (texted) net names in the customized hierarchical layout netlist.

The LNN file is generated by [LAYOUT NAMETABLE WRITE](#) together with the [LAYOUT NETLIST NAMES NONE](#) command (corresponding Tcl shell commands:

`qs::write_layout_netlist_names` and `dfm::set_layout_netlist_options -names`). The LNN file is used with the [AGF WRITE](#) AGF file to correlate net numbers in the AGF file and customized SPICE netlist to the texted layout net names. See “[Customized SPICE Netlist Format](#)” on page 497 for details about the netlist and “[Layout Netlist Names File Format](#)” on page 502 for details about the LNN file.

To generate a Layout Netlist Names file, your rules file must include a Mask SVDB Directory statement with the NETLIST, GDSII, or CCI keyword.

A basic script for generating all CCI files needed for downstream flows is discussed under “[Generating CCI Output Files](#)” on page 480.

LAYOUT NAMETABLE WRITE

CCI Command. Corresponding Tcl shell command: [qs::write_layout_netlist_names](#).

Writes a Layout Netlist Names correspondence file.

Usage

LAYOUT NAMETABLE WRITE *filename* [EXPAND_CELLS]

Arguments

- *filename*

A required pathname of an output file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- EXPAND_CELLS

An optional keyword that specifies the output hierarchy adheres to a non-default [LAYOUT NETLIST HIERARCHY](#) setting.

Description

Writes a Layout Netlist Names (LNN) file describing correspondence of generated net numbers to original layout names. See “[Layout Netlist Names File Format](#)” for details. It is generally recommended that this file be generated in CCI flows.

This file is used in conjunction with the AGF file produced by [AGF WRITE](#) and a customized netlist produced by [LAYOUT NETLIST WRITE](#).

[LAYOUT NETLIST NAMES](#) NONE is used with LAYOUT NAMETABLE WRITE.

The EXPAND_CELLS keyword overcomes hierarchical inconsistency between the flattened output of the CCI netlist and the Layout Netlist Names (LNN) file. Such inconsistencies can occur when the LAYOUT NETLIST HIERARCHY setting is other than ALL. Note that when lower-level net names are expanded into higher-level cells, only top-level net names are printed at the lower levels. Net names that connect upward to other higher-level nets in the containing cell are not printed since they are represented by the higher-level net name in the CCI netlist and in the AGF file. For example, given a top-level cell contains a cell ICV_1 that is expanded in AGF output and has the following net structure:

```
.SUBCKT TOP
X0 net_A ICV_1
.ENDS

.SUBCKT ICV_1 AAA
M0 AAA BBB CCC
.ENDS
```

the net AAA in ICV_1 connects upward to net_A in cell TOP while nets BBB and CCC do not extend upward. In the LNN file, the nets BBB and CCC are represented while AAA is not:

```
% TOP
net_A 1
X0/BBB X0/1
X0/CCC X0/2
```

Requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license. Refer to the [Calibre Administrator's Guide](#) for license information.

Acknowledgments

OK., NOK(33), ERROR(102), ERROR(103), ERROR(130)

The command may generate warnings as it executes. These transcript to the standard out and are prefixed by the string WARNING.

Examples

The following example shows how to output using the default LAYOUT NETLIST HIERARCHY setting and the AGF setting.

```
# write the full LNN file--no cells expanded (default):
LAYOUT NAMETABLE WRITE cci_all.lnn

# change the hierarchy present in the LAYOUT NETLIST output:
LAYOUT NETLIST HIERARCHY AGF

# write the LAYOUT netlist
LAYOUT NETLIST WRITE cci_agf.nl

# write a nametable with the same hierarchy as the AGF netlist
LAYOUT NAMETABLE WRITE cci_agf.lnn EXPAND_CELLS
```

Port Table File Commands

The PORT TABLE WRITE and qs::port_table commands write a file that lists locations of ports identified in the design. PORT TABLE NAME POLYGON PORTS (or qs::port_table -name_polygon_ports) can modify the output to give names to Port Layer Polygon port objects.

To generate a port table, your rule file must include the [Mask SVDB Directory](#) statement with the NETLIST, GDSII, or CCI keyword.

A basic script for generating all CCI files needed for downstream flows is discussed under “[Generating CCI Output Files](#)” on page 480.

PORT TABLE WRITE.....	611
PORT TABLE NAME POLYGON PORTS.....	613

PORT TABLE WRITE

CCI command. Corresponding Tcl shell command: [qs::port_table](#).

Writes a port information file.

Usage

PORT TABLE [CELLS] [DB_CONNECTIVITY] WRITE *filename*

Arguments

- *filename*

A required pathname of a file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- CELLS

An optional keyword that specifies the port table represents all cells in the design. The default is top-level cells only. This keyword causes the table to have cell name fields, which by default, it does not.

- DB_CONNECTIVITY

An optional keyword that specifies to substitute layers from [LVS DB Connectivity Layer](#) rule file statements for Connect or Sconnect layers (which are the default). Port objects that intersect layers from LVS DB Connectivity Layer statements are then listed as if the ports are attached to those layers. (See the [qs::port_table](#) “[Description](#)” section for details about the -db_connectivity option, which is analogous to DB_CONNECTIVITY.)

Description

Writes a port table file to the *filename*.

When LVS DB Connectivity Layer is specified in the rule file but DB_CONNECTIVITY is not specified in the PORT TABLE WRITE command, then these messages appear in the CCI transcript:

NOTE: LVS DB CONNECTIVITY layers are present, but the DB_CONNECTIVITY option was not used.

NOTE: Ports are attached to the original CONNECT/SCONNECT layers.

Requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license. Refer to the [Calibre Administrator’s Guide](#) for license information.

Port Layer Polygon ports do not have names by default in the output table. [PORT TABLE NAME POLYGON PORTS YES](#) changes this behavior.

Response

Writes a [Port Table File Format](#) file.

Acknowledgments

OK., ERROR(101), ERROR(102), ERROR(130)

Related Topics

[Generating CCI Output Files](#)

[Port Table File Commands](#)

[Results Transformation Commands](#)

POR TABLE NAME POLYGON PORTS

CCI command. Corresponding Tcl shell command: [qs::port_table -name_polygon_ports](#).

Controls whether Port Layer Polygon objects are named in port tables.

Usage

POR TABLE NAME POLYGON PORTS {NO | YES}

Arguments

- **NO**

A keyword that specifies to use the default port table format, which treat polygon port objects as unnamed. This is the default.

- **YES**

A keyword that specifies treat polygon port objects as named.

Description

Controls the format of the [PORT TABLE WRITE](#) command output.

When YES is specified, [Port Layer Polygon](#) ports exist, and the [PORT TABLE CELLS WRITE](#) command is used, then unnamed ports that appear by default in the port table, such as this:

```
<UNNAMED> 5 5 -98000 -90000 metal cellB
```

are written like this instead:

```
NET1 5 5 -98000 -90000 metal cellB
```

If a net name is unavailable, a number is used.

Requires a Calibre Connectivity Interface license in addition to a Calibre Query Server license. Refer to the [Calibre Administrator's Guide](#) for license information.

Response

Writes a [Port Table File Format](#) file.

Acknowledgments

OK., ERROR(101), ERROR(102), ERROR(130)

Related Topics

[Generating CCI Output Files](#)

[Port Table File Commands](#)

Reduction Data File Command

Reduction data files contain information about reduction transformations that occur during LVS comparison. Each command producing this data allows selection of the design (source or layout) whose data is to be written, along with the type of reduction data (net or instance) to write, and optionally a single reduction reason to which the data is to be constrained.

To generate reduction data files, your rule file must include the [Mask SVDB Directory](#) statement with the CCI and XFORMS keywords.

This functionality requires a Calibre Connectivity Interface (CCI) license.

The following limitations exist:

- The forward slash (/) hierarchy separator character is always used for paths.
- Because of the structure inherent in reduction data (for example, many types of reductions occur after hcells are flattened), it is not possible for reduction data commands to reconstruct the original design hierarchy with greater refinement than the hcell hierarchy.
- There may be cases where an hcell correspondence is not uniquely identified by its cell names and pin counts—namely, cases where multiple hcell correspondences exist in an LVS-H run with the same combination of cell names and pin counts. In each such case, reduction data is written for only one of those hcell correspondences. This circumstance is rare, but it can happen when nested subcircuit definitions are used in the SPICE input to an LVS-H run.

LVS-H has a flat namespace for cells. (Nested subcircuit names are assigned directly to cells, unqualified by their containing subcircuit contexts.) Therefore, if the same combination of subcircuit name and pin count is found in two or more different SPICE contexts, although the SPICE parser constructs the correct circuit graph, the resulting cells are not distinguishable by name and pin count alone. Because captured reduction data, by contrast, are identified solely by cell names and pin counts, unique dummy cell names must be assigned to all but one of the non-uniquely-identified cells in each group. But such dummy cell names cannot be correlated with other XDB data, and therefore reduction data associated with them is not written by the reduction data commands.

- In any case where each hcell correspondence is not uniquely identified by its cell names and pin counts (see above), flattening of the data cannot be performed (as there is not sufficient information). In such a case, the command terminates with an error if flattening is requested.
- When flattening is requested, there may be certain many-to-one scenarios in which the data is ambiguous. For example, when the same source cell corresponds to multiple layout cells, LVS-H compares each layout cell against a separate representation of the source cell. These separate source cell representatives are not restricted from having different reduction characteristics from one another because they pair with a different

layout cell. When faced with a placement of such a source cell, the tool may be unable to determine which of the hcell correspondences to expand for lower-level reduction data. In such cases, a parameter may be supplied that directs the command to consult XREF data to resolve ambiguities. In the example just mentioned, if the placement in question were matched to a layout placement, the tool then uses the type of the latter to determine which hcell correspondence to choose.

Under the following circumstances, the flattened data is ambiguous: the requested design (source or layout) includes one or more hcells on the "one" side of a one-to-many correspondence, and reduction records of any requested type have been logged for any of these hcell correspondences or any of their children in the hierarchy. In such a case, an error message is issued and no output occurs. This behavior can be overridden with a parameter that directs the command to attempt disambiguation of the data by consulting XREF records in the XDB. When this parameter is specified, the correctness of the output from the command depends on the correctness of instance matching in the LVS run that produced the XDB. When this parameter is not specified, LVS matching results are always ignored by the command.

When flattening is requested, the data is ambiguous, and it has been requested that the command attempt to disambiguate the data, the following limitations apply:

- If a placement is encountered that cannot be disambiguated because it was not matched in the LVS-H run, the command produces an error message and generates no output.
- Because the command's processing relies on LVS results, if the status of the LVS run was not CORRECT, even on successful execution, the command may produce erroneous output.

For the Query Server Tcl shell, the [dfm::write_reduction_data](#) command is used and is documented in the *Calibre YieldServer Reference Manual*.

REDUCTION DATA WRITE **616**

REDUCTION DATA WRITE

CCI command. Corresponding Tcl shell command: [dfm::write_reduction_data](#).

Writes a file containing information about transformation reductions occurring in LVS comparison.

Usage

```
REDUCTION DATA WRITE filename {LAYOUT | SOURCE {NET | INSTANCE}}  
[FLAT [WITH XREF]] [reason_code]
```

Arguments

- ***filename***
A required pathname of an output file. Directories in the pathname are created if they do not exist. The file is overwritten if it already exists.
- **LAYOUT**
A keyword that specifies the reduction data from the layout design are written. Either **LAYOUT** or **SOURCE** must be specified.
- **SOURCE**
A keyword that specifies the reduction data from the source design are written. Either **LAYOUT** or **SOURCE** must be specified.
- **NET**
A keyword that specifies the reduction data from nets are written. Either **NET** or **INSTANCE** must be specified.
- **INSTANCE**
A keyword that specifies the reduction data from instances are written. Either **NET** or **INSTANCE** must be specified.
- **FLAT**
An optional keyword that specifies data are written in flat format instead of hierarchical.
- **WITH XREF**
An optional keyword specified with FLAT that causes the command to attempt to resolve ambiguous flattened reduction data using LVS matching results from the cross-reference data in the XDB. If WITH XREF is not specified, the command terminates with an error when ambiguous flattened reduction data are encountered.
- ***reason_code***
An optional keyword that specifies to report only reductions of a certain type. The allowed codes are: DEEPSHORT, HIGHSHORT, PARALLEL, PORTFLATTEN, SEMISERIES, SEN, SERIES, and SPLITGATE. The definitions of these codes are given in the parameters section under “[Reduction Data File Format](#)” on page 506. Only one code may be specified per command.

Description

Writes a data file containing information about reduction transformations for nets or instances in the layout or source design. A [Mask SVDB Directory](#) statement with the CCI and XFORMS keyword must be used for this command to work. A Calibre Connectivity Interface (CCI) license is also required.

The types of reductions include series and parallel devices; semi-series, split gate, and short equivalent nodes MOS devices; flattened cell pin nets; and deep and high shorts. The specific output of the data file can be tailored to one of these types by using a corresponding *reason_code*. If a *reason_code* is inconsistent with the **NET** or **INSTANCE** specification, the output file shows no transformations.

The hierarchy separator character for netlist object paths is controlled by the [HIERARCHY SEPARATOR](#) command.

Details of the limitations regarding use of the FLAT and WITH XREF keywords are discussed under “[Reduction Data File Command](#)” on page 614.

Acknowledgments

OK., ERROR(191) through ERROR(199), ERROR(202) through ERROR(205)

Cross-Reference System File Commands

Cross-reference files indicate correspondences between nets and instances in source and layout netlists and are used by various downstream simulation tools that use LVS comparison data.

To generate any cross-reference file, your rule file must include the [Mask SVDB Directory](#) statement with the XDB, QUERY, or CCI keywords.

A basic script for generating all CCI files needed for downstream flows is discussed under “[Generating CCI Output Files](#)” on page 480.

The following table describes the commands in this set and includes references to commands that work in the Tcl shell and Calibre YieldServer.

Table 6-8. Cross-Reference System File Commands

Command	Description
HIERARCHY SEPARATOR dfm::set_layout_netlist_options -hier_separator	Specifies the hierarchy delimiter character.
HIERARCHY WRITE dfm::write_lph , dfm::write_sph	Writes a Layout (LPH) or Source Placement (SPH) Hierarchy file.
INSTANCE XREF WRITE dfm::write_ixf	Writes an instance cross-reference file (IXF).
LAYOUT NET XREF WRITE dfm::write_nxf -lnxf	Writes a Layout Net Cross-Reference File (LNXF).
NET XREF WRITE dfm::write_nxf	Writes a Net Cross-reference File (NXF).
XREF XNAME dfm::xref_xname	Configures the subcircuit call format of cross-reference file commands.

HIERARCHY SEPARATOR

CCI command. Corresponding Tcl shell command:

`dfm::set_layout_netlist_options -hier_separator`

Specifies the hierarchy delimiter character.

Usage

HIERARCHY SEPARATOR *separator_character*

Arguments

- *separator_character*

A required single character that is used for a hierarchy delimiter. The following characters may be used for *separator_character*:

`~ ! # $ % ^ & * + ‘ = : ; , . ? | /`

Description

Changes the hierarchical separator reported in cross-reference files, AGF files, reduction data, and layout netlist files from the default “/” character to a specified separator character.

This command is useful in design flows that use the “/” character as a literal character in SPICE names through the [LVS Spice Slash Is Space NO](#) rule file specification statement. The literal forward slash “/” remains as is, but the hierarchy separators are changed to *separator_character*. For example, the following SPICE netlist generates a file output using the literal “/” character when used in conjunction with LVS Spice Slash Is Space NO:

```
.SUBCKT CELLA A/IN1 A/OUT
A/R0 A/IN1 A/OUT 50
.ENDS
```

The SVDB database must be from Calibre version 2003.4 or later. This command is not available to SVDB databases generated by a flat LVS run. This command does not affect the normal operation of non-CCI Query Server commands.

Acknowledgments

OK., ERROR(122), ERROR(153), ERROR(155)

Related Topics

[Cross-Reference System File Commands](#)

HIERARCHY WRITE

CCI command. Corresponding Tcl shell commands: [dfm::write_lph](#), [dfm::write_sph](#).

Writes a Layout or Source Placement Hierarchy file.

Usage

LAYOUT HIERARCHY WRITE *filename* [OMN]

SOURCE HIERARCHY WRITE *filename* [OMN]

Arguments

- **LAYOUT**

A keyword that specifies to write a Layout Placement Hierarchy (LPH) file.

- **SOURCE**

A keyword that specifies to write a Source Placement Hierarchy (SPH) file.

- ***filename***

A required pathname to an output file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- **OMN**

An optional keyword that specifies original model names are written.

Description

Writes a **LAYOUT** or **SOURCE** Placement Hierarchy (LPH or SPH) file describing the original netlist placement hierarchy to the specified *filename*.

Details of the Placement Hierarchy format are provided under “[Placement Hierarchy \(LPH/SPH\) File Format](#)” on page 487. A similar file is generated by Calibre when the Mask SVDB Directory SLPH keyword is set in the rule file. In flat mode, this note is generated:

An empty file is generated when the command is used in FLAT XDB mode.

If OMN is used, the [Mask SVDB Directory](#) statement in the rule file must have the OMN keyword specified. Original model names are names of devices as they appear in an input SPICE netlist before any processing of *.EQUIV statement and [LVS Map Device](#) substitutions. See “[Example 2 — Original model name \(OMN\) in the SPH and LPH files](#)” on page 488 for details of the output.

Acknowledgments

OK., NOK(33), ERROR(101), ERROR(102), ERROR(130), NOTE

Related Topics

[Cross-Reference System File Commands](#)

INSTANCE XREF WRITE

CCI command. Corresponding Tcl shell command: [dfm::write_ixf](#).

Writes an instance cross-reference file (IXF).

Usage

INSTANCE XREF WRITE *filename* [LAYOUT_OMN] [SOURCE_OMN] [BOX]

Arguments

- *filename*

A required pathname to an output file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- LAYOUT_OMN

An optional keyword that specifies layout original model names are written.

- SOURCE_OMN

An optional keyword that specifies source original model names are written.

- BOX

An optional keyword that specifies [LVS Box](#) cells that are matched in layout and source are written to the output.

Description

Writes an instance cross-reference file (IXF) to the specified *filename*. Details of the Instance Cross-Reference (IXF) format are provided under “[Instance and Net Cross-Reference File Formats](#)” on page 490. The format and content of this file is similar (but not identical) to the one generated by the calibre -ixf switch and the Mask SVDB Directory IXF keyword.

If the optional LAYOUT_OMN or SOURCE_OMN keywords are specified, then the [Mask SVDB Directory](#) statement in the rule file must have the OMN keyword specified. Original model names are names of devices as they appear in an input SPICE netlist before any processing of *.EQUIV statement and [LVS Map Device](#) substitutions. See “[Original Model Name Information \(OMN\) in IXF File](#)” on page 492.

If BOX is specified, then matched LVS Box cells are written in the output in this form:

```
% layout_cell pin_count source_cell pin_count BOX
```

Acknowledgments

OK., NOK(33), ERROR(101), ERROR(102), ERROR(122)

Related Topics

[Cross-Reference System File Commands](#)

LAYOUT NET XREF WRITE

CCI command. Corresponding Tcl shell command: [dfm::write_nxf -lnxf](#).

Writes a Layout Net Cross-Reference File (LNXF).

Usage

LAYOUT NET XREF WRITE *filename* [BOX]

Arguments

- *filename*

A required pathname to an output file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- BOX

An optional keyword that specifies [LVS Box](#) cells are added to the LNXF file.

Description

Writes a layout net cross-reference file (LNXF) with net correspondence in the PHDB to the specified *filename*.

The output of this command is often preferred to the default [NET XREF WRITE](#) output. The advantage of the LNXF format is that it provides additional net cross reference detail to downstream tools for all nets in the layout without the downstream tool having to follow connectivity up and down through cells in order to find valid cross reference nets. Details of the file format are provided under “[Layout Net Cross-Reference File \(LNXF\) Format](#)” on page 495.

If you want nets that have been combined during comparison (due to various LVS comparison features like the LVS Filter SHORT keyword and open circuit error recovery) shown with representative nets paired together followed by the combined nets shown only with the representative net from the other design, then use [LAYOUT NETLIST NAMES LAYOUT](#). This gets you the same net grouping as an NXF file, which is desirable for certain flows.

If LAYOUT NETLIST NAMES NONE is set, layout nets are identified by net number. If the LAYOUT or SOURCE keyword is set, layout nets are identified by name if available, else by number. Note that the NONE keyword does not provide the NXF format discussed in the previous paragraph. If the NONE keyword is used, then the following message is written to the transcript:

NOTE: Backward compatibility mode used for LNXF file generation: LAYOUT NETLIST NAMES NONE was set.

The PHDB must be created with Calibre versions 2009.2 and later.

Acknowledgments

OK., NOK(33), ERROR(101), ERROR(102)

Related Topics

[Cross-Reference System File Commands](#)

NET XREF WRITE

CCI command. Corresponding Tcl shell command: [dfm::write_nxf](#).

Writes a Net Cross-reference File (NXF).

Usage

NET XREF WRITE *filename* [BOX] [LNXF | UNMATCHED]

Arguments

- *filename*

A required pathname to an output file. If the output file *filename* ends in either the .Z or .gz suffix, the file is automatically compressed using the compress or gzip commands, respectively. This assumes these commands are available in your environment. Directories in a pathname are created if they do not exist.

- BOX

An optional keyword that specifies [LVS Box](#) cells are added to the NXF file.

- LNXF

An optional keyword that specifies to use the [LAYOUT NET XREF WRITE](#) command's output format—LNXF. The LNXF format is sometimes preferred due to the additional details it provides. See “[Layout Net Cross-Reference File \(LNXF\) Format](#)” on page 495.

- UNMATCHED

An optional keyword that specifies nets that would ordinarily be removed due to their lack of correspondence across the design are written to the output.

Description

Writes a net correspondence file (NXF) to the specified *filename*. The format and content of this file is similar to the one generated by the calibre -nxfs switch and the Mask SVDB Directory NXF keyword. Details of the Net Cross-Reference (N XF) format are provided under “[Net Cross-Reference File \(N XF\) Format](#)” on page 493.

A net may have no correspondence due to discrepancies in the design. These discrepancies can be unmatched nets or graph transformations like filtering, reduction, or logic injection. These unmatched nets are not written to the output by default. The UNMATCHED keyword changes this.

A net identified by the UNMATCHED keyword could have a trailing “R” representing the net was removed during device reduction. Furthermore, the set of characters “\$ _” indicates there is no matching device on a particular side of the comparison. The following example indicates that net 333 in the source netlist was removed due to device reduction and that there is no corresponding net in the layout:

```
0 $_ 0 333 R
```

A trailing “U” in the NXF file indicates that a net is unmatched and has been reported during comparison. In the following example, net 777 is unmatched in the source netlist:

```
0 777 0 $_ U
```

Acknowledgments

OK., NOK(33), NOK(62), NOK(63), NOK(64), NOK(65), ERROR(101), ERROR(102)

Related Topics

[Cross-Reference System File Commands](#)

XREF XNAME

CCI Command. Corresponding Tcl shell command: [dfm::xref_xname](#).

Configures the subcircuit call format of cross-reference file commands.

Usage

XREF XNAME {LAYOUT | SOURCE} {ON | OFF} [BOX] [DEVICE]

Arguments

- **LAYOUT**

A keyword that causes layout names to be configured. This keyword is used by default, but when the command is used explicitly, either **LAYOUT** or **SOURCE** must be specified.

- **SOURCE**

A keyword that causes source names to be configured. This keyword is used by default, but when the command is used explicitly, either **LAYOUT** or **SOURCE** must be specified.

- **ON**

A keyword that causes the normal hierarchical pathname format used in LVS and elsewhere in the Query Server to be output. This is the default for all subcircuit call types.

- **OFF**

A keyword that causes subcircuit calls in paths to drop the leading X character. By default, this does not include primitive subcircuit calls.

- **BOX**

An optional keyword that causes the command to process only [LVS Box](#) cell primitive subcircuit calls.

- **DEVICE**

An optional keyword that causes the command to process only primitive device subcircuit calls.

Description

Configures the hierarchical pathname format for the output of the following commands:

[INSTANCE XREF WRITE](#)

[LAYOUT HIERARCHY WRITE](#)

[LAYOUT NET XREF WRITE](#)

[NET XREF WRITE](#)

[SOURCE HIERARCHY WRITE](#)

When ON is specified (the default), this command produces the standard hierarchical pathname format used elsewhere in Calibre nmLVS and the Query Server for the aforementioned commands. Specifically, when operating on SVDB databases created by hierarchical circuit comparison, names of SPICE subcircuit calls include the preceding subcircuit-call designator “X”, like this:

```
0 a/Xb
```

Specifying OFF causes subcircuit call names to appear without preceding X characters, like this:

```
0 a/b
```

except for primitive subcircuit call names, which retain the X prefix. Primitive subcircuit calls are those referencing empty subcircuits, user-defined devices, or [LVS Box](#) cells. Note that primitive subcircuit call names can only appear at the end of a hierarchical pathname or as the only element in a pathname.

The BOX keyword used with OFF causes LVS Box subcircuit call names to appear without the preceding X character. Other primitive subcircuit names retain the X character, as usual. Similar semantics apply to the DEVICE keyword, which applies only to primitive device subcircuit calls.

Acknowledgments

OK., ERROR(122)

Examples

The default INSTANCE XREF WRITE command output might appear as follows for an inductor device (type L) in the source and an X0 call in the layout:

```
0 X0/X0/X0 0 69/42/XL0
```

Notice on the source side that an X appears before L0. If XREF XNAME SOURCE OFF DEVICE is used, the output would be this:

```
0 X0/X0/X0 0 69/42/L0
```

Related Topics

[Cross-Reference System File Commands](#)

Chapter 7

Query Server Runtime Messages

This section contains the runtime messages given by the Query Server.

Error and Warning Messages.....	631
Failure Messages	638
Note Messages	641

Error and Warning Messages

This section describes the errors and warnings that may be returned as acknowledgments. These conditions generally involve avoidable problems, such as unknown commands, missing arguments, or environment failures.

Table 7-1. Error and Warning Messages

Message	Description
ERROR(101)	File <response_filename> could not be opened for writing.
ERROR(102)	Problems writing to file <response_filename>.
ERROR(103)	There is no layout cell named <cell_name>. (Current context remains unchanged.)
ERROR(104)	Layout cell <viewing_cell> has no instance <query_instance>. (Current context remains unchanged.)
ERROR(105)	Due to problems parsing cross-reference files, the Query Server cannot be initialized.
ERROR(106)	This client ID is negative or malformed: client_id.
ERROR(107)	There is no client with ID: client_id.
ERROR(108)	Client 0 can never be disconnected.
ERROR(109)	The active client cannot be disconnected.
ERROR(110)	Marker size value is negative or malformed: size.
ERROR(111)	Unknown command or wrong number of arguments.
ERROR(112)	Command not yet implemented.
ERROR(113)	Filter distance is negative or malformed: distance.

Table 7-1. Error and Warning Messages (cont.)

Message	Description
ERROR(114)	Unknown layer name or number: <layer_name_or_number>.
ERROR(115)	Unknown device type number: <device_type_number>.
ERROR(116)	The x y location is malformed: <coordinate>.
ERROR(117)	Invalid or malformed window coordinates: <x1> <x2> <y1> <y2>
ERROR(118)	Invalid or malformed cull distances: <x> <y>
ERROR(119)	The maximum vertex count was malformed or less than 4.
ERROR(120)	Query instance argument <query_instance> not valid in XDB only mode.
ERROR(121)	Zoom factor is negative or malformed: <z_factor> (RVE only).
ERROR(122)	Command not valid in FLAT mode. Run -hier or with -flatten.
ERROR(123)	Invalid arguments to AGF MAP [layer][datatype].
ERROR(124)	Value specified <value> is not representable in GDSII.
ERROR(125)	Invalid cell name/pin count combination: <layout_cell> <pin_count> <source_cell> <pin_count>
ERROR(126)	Unknown device: <name>
ERROR(127)	(not currently used)
ERROR(128)	Layout netlist hierarchy must be FLAT when layout netlist source is specified.
ERROR(129)	SVDB Database revision does not support this feature, rerun Calibre.
ERROR(130)	Incorrect MASK SVDB DIRECTORY options for this function.
ERROR(131)	CALIBRE CONNECTIVITY INTERFACE license is unavailable.
ERROR(132)	Delta XY setting is malformed <x> <y>. (RVE only)
ERROR(133)	Must specify DEVICE or ORIGINAL layer.
ERROR(134)	LVS Setup failed.
ERROR(135)	Netlist Read failed.
ERROR(136)	Netlist has not been read.

Table 7-1. Error and Warning Messages (cont.)

Message	Description
ERROR(137)	Invalid device location type specified. (NONE CENTER VERTEX).
ERROR(138)	Invalid hierarchy type specified. (FLAT HCELL ALL AGF).
ERROR(139)	Invalid instance names type specified. (LAYOUT SOURCE).
ERROR(140)	Invalid net names type specified. (LAYOUT SOURCE NONE).
ERROR(141)	Invalid netlist type specified. (SPICE CALIBREVIEW).
ERROR(142)	Text size is negative or malformed: <size>. (RVE only)
ERROR(143)	Layout report requested, but only source netlist was read.
ERROR(144)	Source report requested, but only layout netlist was read.
ERROR(145)	Rules file not compiled.
ERROR(146)	Hcell read failed.
ERROR(147)	Pan/Zoom setting is malformed: <setting> (RVE only)
ERROR(148)	No hcells available to evaluate.
ERROR(149)	Threshold value is malformed or out of range [0-100]: <value>
ERROR(150)	XDB is not available.
ERROR(151)	Configurable hierarchical name is not enabled.
ERROR(152)	Unable to reload PHDB.
ERROR(153)	Failure to set SVDB password: <diagnostic>
ERROR(154)	Configurable hierarchical separator is not enabled.
ERROR(155)	Separator must be a single character from the set <chars>.
ERROR(156)	Password argument is required when input is not a terminal.
ERROR(158)	Unknown device layer name: <layer_name>.
ERROR(159)	Pin location information was separated for PUSHDOWN.
ERROR(160)	Pin location information is not available.
ERROR(161)	Unknown LVS ANNOTATE DEVICES layer name: <layer_name>.

Table 7-1. Error and Warning Messages (cont.)

Message	Description
ERROR(162)	Layer <layer_name> is already present in the ANNOTATED DEVICES map.
ERROR(163)	Layer <layer_name> is not currently present in the ANNOTATED DEVICES map.
ERROR(164)	Bad argument to command: <arg>
ERROR(165)	Tcl command failure: <Tcl error info>
ERROR(166)	Old version of Cross Reference database does not support this feature - rerun using a more recent version of Calibre LVS.
ERROR(167)	<path> is an Invalid Path.
ERROR(168)	Invalid Cell Name: <cell> is not present in the Cross Reference Database.
ERROR(169)	Invalid Port Count: Cross Reference Database contains no Cell <cell> that has <pin_count> ports.
ERROR(170)	Invalid <magnification_factor> <factor>. Must be (LAYOUT PRECISION / PRECISION).
ERROR(171)	Not compatible with MAGNIFY RESULTS setting. The MAGNIFY RESULTS setting must be 1 to use this command.
ERROR(172)	MAGNIFY RESULTS only supports use of LAYOUT MAGNIFY AUTO during extraction.
ERROR(173)	MAGNIFY RESULTS requires use of PRECISION and LAYOUT PRECISION during extraction.
ERROR(174)	MAGNIFY RESULTS requires PRECISION > LAYOUT PRECISION during extraction.
ERROR(175)	<parameter description> is a required argument for command qs::parse_path.
ERROR(176)	<option> is an invalid argument for command qs::parse_path.
ERROR(177)	No rules file has been loaded.
ERROR(184)	Rule file setting LVS CENTER DEVICE LOCATION YES disallows this setting.
ERROR(185)	Layer <layer> cannot be merged.
ERROR(186)	Device information could not be loaded. RESPONSE FILE cannot write compressed output. ¹

Table 7-1. Error and Warning Messages (cont.)

Message	Description
ERROR(187)	ROTATE RESULTS only takes integral multiples of 90 on the interval [-270, 270].
ERROR(188)	hcells::read -rules is called when rules have already been loaded.
ERROR(189)	LVS Push Device SEPARATE PROPERTIES was used. Query must use top level cell context.
ERROR(190)	Cannot process the SOURCE SYSTEM in the rule file loaded by NETLIST READ or hcells::read.
ERROR(191)	WITH XREF keyword specified multiple times.
ERROR(192)	Multiple reason codes specified.
ERROR(193)	Unexpected keyword '<keyword>'.
ERROR(194)	LAYOUT or SOURCE must be specified.
ERROR(195)	NET or INSTANCE must be specified.
ERROR(196)	WITH XREF can only be specified with FLAT.
ERROR(197)	Flattening cannot be produced with ambiguous hcells. See discussion of limitations under “ Reduction Data File Command ” on page 614.
ERROR(198)	One-to-many hcell correspondence presents a flattening ambiguity. Try adding the WITH XREF (or -with_xref in a Tcl shell) option.
ERROR(199)	Unable to resolve flattening ambiguity with XREF data. Even with WITH XREF, the command is unable to flatten the data.
ERROR(202)	The design type (LAYOUT or SOURCE) is specified multiple times.
ERROR(203)	Reduction type (NET or INSTANCE) is specified multiple times.
ERROR(204)	Invalid command syntax.
ERROR(205)	FLAT keyword specified multiple times.
ERROR(206)	Two input netlists are currently required for hcell selection because strict hcell automatching is requested, but only one netlist is available.
ERROR(207)	Two input designs are currently required because strict hcell automatching is requested, but only one design has been read.

Table 7-1. Error and Warning Messages (cont.)

Message	Description
Error 215	The SVDB was created during a -recon -erc run, so it does not contain layer information for softchk commands.
Error 300	The loaded SVDB must have been created by a previous calibre -recon -si run, or a circuit extraction run that used the Mask SVDB Directory SI keyword.
ERROR	Invalid CONNECT/SCONNECT statement is specified in short_db::select_connects -select_by or Invalid CONNECT/SCONNECT statement is specified in short_db::select_connects -select_by A short_db::select_connects command specifies a connectivity set that is unspecified in the rules, or is unavailable due to an LVS Recon Select Connects statement in the rules.
ERROR	0 CONNECT/SCONNECT statements are selected for connectivity extraction by short_db::select_connects. short_db::select_connects -select_by_layer matched no connectivity operation in the rules.
WARNING: <factor> set to non-standard value.	A factor specified in MAGNIFY RESULTS does not match the rule file precision settings. The default of 1 is used.
	WARNING: Port <name>(<x>,<y>) in cell <name> was not reported in DB_CONNECTIVITY mode. A layout port does not interact with a layer specified in an LVS DB Connectivity Layer statement and the PORT TABLE WRITE command is used with the DB_CONNECTIVITY keyword. The port is listed as attached to a Connect or Sconnect layer.

1. The assignment of two messages to this code is due to a historical oversight. To preserve existing customer code that may reference ERROR 186, both messages are supported.

The following is a list of discrepancy messages from the Query Server Tcl shell.

Table 7-2. Tcl Shell Runtime Messages

Code	Meaning
Error 1	The Query Server is not initialized.
Error 2	There is no database loaded. This message applies to DFM or SVDB databases.

Table 7-2. Tcl Shell Runtime Messages (cont.)

Code	Meaning
Error 12	Invalid argument for the command. The message indicates details.
Error 13	Invalid argument type.
Error 18	Missing data for the command.
Error 20	Missing argument for the command.
Error 28	No PHDB information is in the Mask SVDB Directory.
Error 30	Wrong number of arguments for the command.
Error 45	Invalid argument set for the command.
Error 47	Cannot find the cell <name>.
Error 48	Problem with the specified list.
Error 52	Iterator is at its final entry.
Error 197	Requires a Tcl list argument.
Error 250	An internal error has occurred.
Error 264	The option is missing an argument.
Error 275	A CCI license could not be accessed but is required.
Error 276	Mask SVDB Directory was specified without the CCI option, but that option is required for a command to work.
Error 280	There is no cross-reference database (XDB) in the SVDB. Possibly LVS comparison was not run.
Error 300	Mask SVDB Directory SI must have been specified in order for the short isolation command to work.
Error 302	A function is used that requires the SVDB to have been produced using the Mask SVDB Directory SI keyword. This error can occur in Calibre nmLVS Reconnaissance runs.
Error 3401	Argument is not within the allowed range of values.
Error 3402	Argument is not the correct type.
Error 3403	The PHDB failed to open.
Error 3404	The PHDB cannot be restored.
Error 3405	The SVDB is not currently loaded.
Warnings	
Warning 37	LVS Annotate Devices needs to be specified when generating the SVDB in order to get proper output.

Table 7-2. Tcl Shell Runtime Messages (cont.)

Code	Meaning
	<p>WARNING: Port <name>(<x>,<y>) in cell <name> was not reported in DB_CONNECTIVITY mode.</p> <p>A layout port does not interact with a layer specified in an LVS DB Connectivity Layer statement and the qs::port_table command is used with the -db_connectivity keyword. The port is listed as attached to a Connect or Sconnect layer.</p>

Failure Messages

This section describes the failures that may be returned as acknowledgments. They indicate the command (usually a request for design information) was performed but failed to produce the requested response.

Table 7-3. Failure Messages

Message	Description
NOK(1)	There are no unfiltered pins on layout net layout_net_path.
NOK(2)	No source cell corresponds to layout cell cell_name.
NOK(3)	Either layout cell layout_cell has no net layout_net_path or it was removed before comparison.
NOK(4)	Either source cell source_cell has no net source_net_path or it was removed before comparison.
NOK(5)	Layout cell query_cell has no net named layout_net_path.
NOK(6)	No device of the filtered type found within the filter distance.
NOK(7)	Either layout cell query_cell has no device layout_device_path or it was filtered.
NOK(8)	Either source cell source_cell has no device source_dev_path or it was filtered.
NOK(9)	Layout cell query_cell has no device layout_device_path.
NOK(10)	Layout cell query_cell has no ports.
NOK(11)	Layout cell query_cell has no port named port_name.
NOK(12)	There are no ports on layout net layout_net_path.
NOK(13)	No port found on the filter layers.
NOK(14)	No net found on the filter layers.
NOK(15)	Flattened layout cell query_cell has no ports.

Table 7-3. Failure Messages (cont.)

Message	Description
NOK(16)	Layout cell query_cell has no nets.
NOK(17)	Flattened layout cell query_cell has no nets.
NOK(18)	Layout cell query_cell has no devices.
NOK(19)	Flattened layout cell query_cell has no devices.
NOK(20)	Layout cell query_cell has no bad devices.
NOK(21)	Cross-reference commands are disabled due to missing cross-reference.
NOK(22)	Layout cell query_cell has no placements.
NOK(23)	Flattened layout cell query_cell has no placements.
NOK(24)	No placements were selected.
NOK(25)	Layout cell query_cell has no such placement.
NOK(26)	No layout cell corresponds to source cell cell_name.
NOK(27)	No LVS report was specified in the rules.
NOK(28)	No SVDB directory was specified in the rules.
NOK(29)	Layout cell query_cell has no unfiltered shapes on net net_names.
NOK(30)	No source directory was specified in the rules.
NOK(31)	No layout file was specified in the rules.
NOK(32)	Layout cell query_cell has no placement placement_name.
NOK(33)	Interrupted.
NOK(34)	Layout cell query_cell had no unfiltered devices on net net_name.
NOK(35)	Layout cell query_cell had no net net_name extending into placement_name.
NOK(36)	There is no layout cell named target_cell.
NOK(37)	Target net target_net must be a top level net of target_cell.
NOK(38)	Layout cell target_cell is topologically higher than net reference_net.
NOK(39)	No placement of target_cell lies within the current query context.
NOK(40)	Layout database query commands are disabled due to missing PHDB database.
NOK(41)	source_device was not matched to a layout device.
NOK(42)	layout_device was not matched to a source device.

Table 7-3. Failure Messages (cont.)

Message	Description
NOK(43)	File name not available.
NOK(44)	Net source_net_path was not matched to a layout net.
NOK(45)	Net layout_net_path was not matched to a source net.
NOK(46)	Layout cell <cell_name> has no [INVALID] net texts.
NOK(47)	Layout cell <cell_name> has no [INVALID] port texts.
NOK(48)	Invalid placement index <index> specified.
NOK(49)	Unable to find hcells <source_cell> <layout_cell>.
NOK(50)	No evaluated hcells to add.
NOK(51)	No database has been read.
NOK(52)	LVS run was not completed.
NOK(53)	This version of XDB does not have RESULT information.
NOK(54)	Failed to set SVDB password: <i>database_type</i> : detailed error message.
NOK(55)	Pin location information is not available.
NOK(56)	Pin location information is not available when LVS PUSH DEVICES SEPARATE PROPERTIES YES is specified.
NOK(57)	LVS ANNOTATE DEVICES was not specified in original rule file.
NOK(58)	No LVS PUSH DEVICES SEPARATE PROPERTIES YES statement was specified in the rules.
NOK(59)	When LVS PUSH DEVICES SEPARATE PROPERTIES is specified, pin location information requires LAYOUT NETLIST SEPARATED PROPERTIES YES.
NOK(60)	No PDSP layers exist because a SVDB directory is not specified in the rules file.
NOK(61)	Command ignored in FLAT mode.
NOK(62)	Source net <net name> was removed during device reduction.
NOK(63)	Layout net <net name> was removed during device reduction.
NOK(64)	Source net <net name> was removed as a passthrough net.
NOK(65)	Layout net <net name> was removed as a passthrough net.
NOK(66)	Source net <net name> was removed by LVS FILTER UNUSED.
NOK(67)	Layout net <net name> was removed by LVS FILTER UNUSED.

Table 7-3. Failure Messages (cont.)

Message	Description
NOK(68)	LAYOUT SEPARATED PROPERTIES WRITE is executed but no PDSP properties are available.
NOK(69)	Not used.
NOK(70)	Source instance <instance name> was removed by LVS FILTER UNUSED.
NOK(71)	Layout instance <instance name> was removed by LVS FILTER UNUSED.
NOK(73)	AGF MERGE PIN LAYERS cannot find a specified pin.

When the Query Server or RVE restores the PHDB database, the SVRF rule file stored in the PHDB is recompiled. There are circumstances that cause this compilation to fail even when it was successful during the original Calibre run. Query Server and RVE issues a diagnostic message when the rule file compilation fails similar to:

```
RESTORATION OF PHDB FAILED.
PHDB STATUS IS -13
Rules file is invalid or incomplete.
Error ENV1 on line 233 of rules1.4497 - undefined or empty environment
variable: DESIGN_DIR.
```

Note Messages

The following NOTE messages can be issued:

- NOTE: Backward compatibility mode used for LNXF file generation: Old SVDB - Rerun calibre.

This is issued when the SVDB is created with Calibre 2009.1 or earlier, but LNXF file generation requires a later version of the tool to generate the SVDB.

- NOTE: Backward compatibility mode used for LNXF file generation: LAYOUT NETLIST NAMES NONE was set.

This is issued in Calibre 2012.3 or later when LAYOUT NETLIST NAMES NONE is set and an LNXF file is generated. The LNXF format is an older format in this case. To get the later format, use LAYOUT instead of NONE.

- NOTE: LVS DB CONNECTIVITY layers are present, but the DB_CONNECTIVITY option was not used.

The LVS DB Connectivity Layer statement is present in the rule file, but the qs::port_table or PORT TABLE WRITE command does not specify an option to use those layers.

- NOTE: No evaluated hcells to add.

The hcells::select or NETLIST SELECT HCELLS command was unable to find hcells to add to the current hcell list.

- NOTE: Ports are attached to the original CONNECT/SCONNECT layers.

This note is given along with the preceding one to indicate port objects remain attached to Connect or Sconnect layers rather than layers specified on LVS DB Connectivity Layer statements.

- NOTE: <m> short paths from <n> out of <o> shorts were selected for short isolation.

This note is given when the qs::isolate_shorts or short_db::isolate_shorts command is issued.

Index

— Symbols —

[]²⁶

{}²⁷

|²⁷

— A —

Annotated geometry format (AGF)⁵²⁵
Annotated device commands⁵²¹
Annotated geometry format (AGF)
 outputting specific layers⁵²⁷

— B —

Bold words²⁶

— C —

Calibre Connectivity Interface (CCI)⁴³³
 output files generation script⁴⁸⁰
 Tcl shell commands⁴³⁴
Case sensitivity²⁶²
CCI, *see* Calibre Connectivity Interface
Cell
 query²⁵⁵
 viewing²⁵⁵
Cell extents report command⁵⁶⁵
Cell extents reports (CCI)⁵⁶⁵
Cell query commands³⁰¹
Change set¹⁶⁰
Client context²⁵⁵
Command format, standard²⁶¹
Command line
 standard mode²⁴⁸
Command syntax²⁶
Communication and control commands²⁶³
Conflicting layers⁴³⁵
Courier font²⁶
Cross-reference database (XDB)^{22, 249}
Cross-reference files (CCI)^{486, 490}
Cross-reference system commands^{614, 618}
Current client²⁵⁷
Customized layout netlist (CCI)⁵⁶⁸

Customized SPICE netlist format⁴⁹⁷

— D —

Device query commands³¹⁵
Device tables²⁴
Deviceless cells³⁰⁹
Double pipes²⁷

— E —

Environment variables
 CALIBREQSRC³¹
 QS_ISI_TRANSCRIPT³⁰

Error messages⁶³¹

Examples
 AGF generation script⁶⁰³
 creating Hcell reports⁴²⁹
 generating a flat netlist⁵¹⁸
 hcell list generation (Tcl shell)²¹⁴
 hcell report generation²¹⁸
 separated properties file script^{598, 606}
 Writing CCI files⁴⁸⁰
 Writing connectivity and device layers to
 AGF⁵¹⁹
 Writing PDSP properties⁵¹⁵

— F —

Failure messages⁶³⁸
Font conventions²⁶

— H —

Hcell analysis
 current hcell list²¹²
 evaluation report²⁴⁰
 for circuit extraction²¹⁶
 for LVS comparison²¹³
 generating reports²¹⁷
 hierarchy tree report²⁴²
 placement matching²⁴⁶
 setting a threshold for evaluating hcells,
 213
 standard commands (non-Tcl)^{410, 411}

-
- Tcl shell commands, 219
THIC, 213
unbalanced hcells, 246
- Heavy font, 26
Help command, 268
Hierarchy report, 430
- I —
Initialization file (Tcl shell), 31
Invocation
 standard mode, 248
- Italic font, 26
- Iterator, 33
- IXF file format, 490
- L —
Layout netlist commands, 568
Layout netlist names command, 607
Licensing, 21
LNN file, 607
 format, 502
- LNXF file format, 495
- LPH file format, 487
- LVS custom report commands, 37
- LVS Hcell Report statement, 211
- M —
Magnification command, 296, 298, 299, 300
Minimum keyword, 26
Modes of operation, 23
- N —
Net query commands, 330
Netlisting commands (CCI), 568
NXF file format, 490
- O —
Output response file, 272
Outputs
 acknowledgments, 252
 responses, 253
- P —
Parentheses, 27
Passing Tcl scripts, 30
Persistent hierarchical database (PHDB), 19, 22, 249
- Pipes, 27
Placement hierarchy files, 487
Placement query commands, 363
Placementmatch, 246
Port query commands, 294, 380
Port table commands (CCI), 610
Port table file format, 504
Pseudo cells, 309
Pushdown separated properties flow (PDSP), 514, 515
- Q —
Query cell, 255
Query Server
 commands, 261
 described, 248
 error messages, 631
 failure messages, 638
 modes of operation, 23
 rule file compilation failure message, 636
- Quotation marks, 27
- R —
Response file, 254
Response format, 253
Rule file query commands
 standard, 386
 Tcl shell, 38
- S —
Scripts in the Tcl shell, 30
Separated properties, 596, 604
Server status, 291
Set cell context, 278
Setup commands, 276
Short database commands, 204
Short isolation commands, 41, 44
Short repair database commands, 42
Slanted words, 26
SPH file format, 487
SPICE netlist format, 497
Square parentheses, 26
Standard Verification Database (SVDB), 249
 header, 495
 password, 22
- Syntax conventions, 26

— T —

Table of devices, [326](#)
Table of ports, [610](#)
Tcl shell
 command types, [34](#)
 initialization file, [31](#)
Terminate the server, [275](#)
THIC, [240](#)
THIC (total hierarchical instance count), [213](#)
tvf::svrf_var command, [48](#)

— U —

Unbalanced hcell reporting, [246](#)
Underlined words, [26](#)
Usage
 standard mode, [248](#)
Usage syntax, [26](#)

— V —

Viewing cell, [255](#)

— W —

Workflow, [19](#)
Write annotated GDS file, [563](#)
Write layout netlist, [602](#)

— X —

XDB, [22](#), [249](#)

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.

