SIEMENS EDA

Calibre® YieldServer Reference Manual

Software Version 2021.2
Document Revision 14

SIEMENS

Unpublished work. © 2021 Siemens

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

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

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

TRADEMARKS: The trademarks, logos, and service marks ("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' trademarks 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.

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

Revision History

Revision	Changes	Status/ Date
14	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released April 2021
13	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released January 2021
12	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released October 2020
11	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released July 2020

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/.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table of Contents

List of Figures

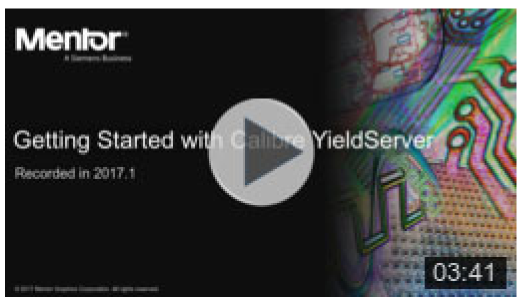
List of Tables

Chapter 1
Introduction to Calibre YieldServer

Calibre® YieldServer processes layout design and rule file data after a calibre -dfm run.YieldServer scripts post process design data in a DFM database. Specifically, YieldServer allows retrieval of layer, polygon, edge, connectivity, property, device, and other information in a DFM database. Annotations and DFM properties facilitate detailed data analysis.

The DFM database used by Calibre YieldServer supports revisions, so multiple versions of a database can be stored and accessed independently. YieldServer also allows modification of this stored data for output to mask or ASCII RDB layers.

Certain commands are supported in the Calibre Query Server Tcl shell. These are of interest in LVS flows.



Requirements

Calibre YieldServer has two general requirements in order to run.

- A DFM Database generated by a Calibre run. Certain commands are recognized in the Calibre Query Server Tcl shell, and a Mask SVDB Directory is used instead of a DFM database in that case.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- A Calibre YieldServer license. Additional licenses may be required depending on the operations performed. Complete licensing information is provided under “Licensing: Physical Verification Products” in the *Calibre Administrator’s Guide*.

Syntax Conventions

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

Table 1-1. Syntax Conventions

<table>
<tr><th>Convention</th><th>Description</th></tr>
<tr><td>Bold</td><td>Bold fonts indicate a required item.</td></tr>
<tr><td>Italic</td><td>Italic fonts indicate a user-supplied argument.</td></tr>
<tr><td><code>Monospace</code></td><td>Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.</td></tr>
<tr><td><u>Underline</u></td><td>Underlining indicates either the default argument or the default value of an argument.</td></tr>
<tr><td>UPPercase</td><td>For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.</td></tr>
<tr><td>[ ]</td><td>Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.</td></tr>
<tr><td>{ }</td><td>Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.</td></tr>
<tr><td>‘ ’</td><td>Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.</td></tr>
<tr><td>| or ||</td><td>Vertical bars indicate a choice between items. Do not include the bars when entering the command.</td></tr>
<tr><td>…</td><td>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.</td></tr>
<tr><td colspan="2">Example:
DEVice {<i>element_name</i> [‘(’<i>model_name</i>‘)’]}
<i>device_layer</i> {pin_layer [‘(’<i>pin_name</i>‘)’] …}
[‘<’<i>auxiliary_layer</i>‘>’ …]
[‘(’<i>swap_list</i>‘)’ …]
[<u>BY NET</u> | BY SHAPE]</td></tr>
</table>

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Work Flow

For the layout design team, the typical design verification work flow is relatively unchanged from customary DRC and LVS practices when using YieldServer. A primary difference is, the YieldServer commands can provide more information than a typical DRC, DFM, or LVS run. For the rule writer and CAD team, the typical flow is expanded to include generating, analyzing, and reporting property values, along with post-processing of data in the DFM database.

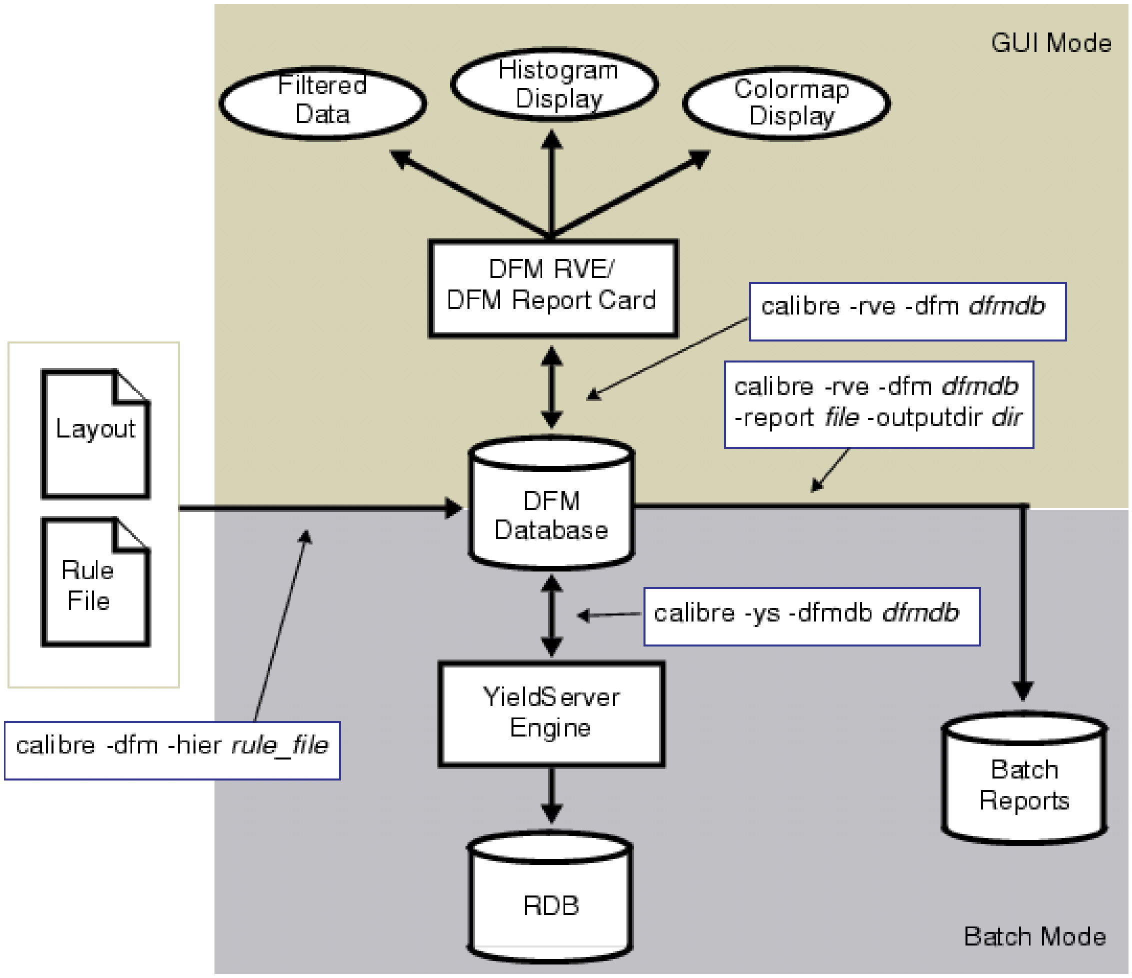
There are a number of components to Calibre YieldServer:

- **DFM Database** — A design database that stores hierarchical layout objects with associated attributes and properties in a proprietary binary format. You create this database from a standard layout database using the calibre -dfm executive. (Certain commands can be used in the Calibre Query Server Tcl shell. These access a Mask SVDB Directory instead of a DFM database.)
- **Calibre YieldServer Engine** — Data processing software that facilitates querying and modifying data in a DFM database. This includes adding annotations, properties, polygons, edges, layers, and so forth. Layout data can be output in mask or ASCII RDB form. Netlists and other LVS data can be output as well.
- **Calibre YieldServer Command Language** — A Tcl-based language for interfacing with the Calibre YieldServer engine. See "Calibre YieldServer Reference" on page 49.
- **DFM Report Card** — Functionality built into the Calibre Results Viewing Environment (RVE) to provide a graphical user interface to the Calibre YieldServer engine. See "Calibre RVE for DFM Report Card" in the *Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual*.
- **DFM Batch Reporting Tool** — Batch interface to the DFM Report Card —see "DFM HTML Reporting" in the *Calibre RVE User's Manual*.

Figure 1-1 shows the typical work flow for GUI and batch modes of operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 1-1. Calibre YieldServer Workflow



The typical Calibre YieldServer workflow involves four steps, as shown in Table 1-2.

Table 1-2. Steps for Using Calibre YieldServer

Step	Description
1	Write or obtain a rule file to calculate and assign DFM properties to objects in the layout.
2	Run calibre -dfm to save the layout plus properties to a DFM database.
3	Load a DFM database into Calibre RVE. or Run Calibre in -ys mode to process the DFM database.
4	Process the YieldServer output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SVRF Statements Supported by -dfm

Running calibre -dfm -hier supports certain SVRF statements that control what appears in the DFM database.

- DFM Select Check, DFM Database, DFM RDB, and DFM Histogram.
- All layer operations that run with the -drc option except for DBClassify and Net Area Ratio Print.
- Any specification statements that run with the -drc option except the following, which are ignored:

Table 1-3. DRC Specification Statements Ignored by calibre -dfm

DRC Cell Name	DRC Cell Text	DRC Check Map
DRC Check Text	DRC Incremental Connect	DRC Incremental Connect Warning
DRC Keep Empty	DRC Magnify Results	DRC Map Text
DRC Map Text Depth	DRC Maximum Cell Name Length	DRC Maximum Results
DRC Maximum Vertex	DRC Print Area	DRC Print Perimeter
DRC Results Database	DRC Results Database Libname	DRC Results Database Precision
DRC Summary Report		

- All statements used for connectivity extraction. These include Connect, Sconnect, Attach, statements from the Text family, and statements from the Port family. Note that all text objects are extracted regardless of Text Depth.
- The Pathchk statement, except for the PRINT NETS and PRINT POLYGONS options, which are ignored.
- The ERC Path Also statement. All other ERC specification statements are ignored.

The following LVS specification statements are used for device recognition:

Table 1-4. LVS Specification Statements Supported by calibre -dfm

LVS Device Type	LVS Downcase Device	LVS Filter Unused Bipolar
LVS Filter Unused Capacitors	LVS Filter Unused Diodes	LVS Filter Unused MOS
LVS Filter Unused Option	LVS Filter Unused Resistors	LVS Ground Name
LVS Power Name	LVS Precise Interaction	LVS Push Devices

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For some SVRF operations, a subset of the options are ignored when running -dfm. These include the following:

Table 1-5. SVRF Operations with Options Ignored by -dfm

Operation	Ignored Options
Density Convolve	PRINT, PRINT ONLY
DFM Analyze	RDB and RDB ONLY
DFM Measure	RDB and RDB ONLY
DFM Transition	RDB and RDB ONLY

In general, any layer operation that uses the RDB, RDB ONLY, PRINT, or PRINT ONLY keyword does not create the specified file when the -dfm option is used on the command line. The RDB ONLY keyword causes the output layer *check_name::<n>* to be empty.

The following operations that print RDBs with the -drc option create DFM database layers instead with the -dfm option: DFM RDB, DFM Analyze, DFM Measure, DFM Spec Fill, Net Area Ratio, and Density.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Modes of Operation

Calibre YieldServer supports a batch mode, an interactive mode, and a GUI mode. The batch mode uses a Tcl script to execute commands. The interactive mode uses a Tcl shell in a terminal. The GUI mode is referred to as a “DFM Report Card” which is implemented in Calibre RVE.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

calibre -ys

Calibre YieldServer command line.

Usage

```
calibre -ys [ -dfmdb dfm_database [ -rev revision_name ] ]
            [-autostart]
            [ { { -turbo [ number_of_processors ] [ -turbo_all ] }
                || { -turbo_litho [ number_of_processors ] }
                || { -turbo [ number_of_processors ] [ -turbo_all ]
                     -turbo_litho [ number_of_processors ] }
              } [ -remote host,host,... || -remotefile filename
                  || -remotecommand filename count ]
             [ -ys_hyper ]   ]
             [ [-cmp] -exec script_filename [arg_list] ]
```

Arguments

- **-ys**

  A required argument to run Calibre YieldServer. Requires a Calibre YieldServer license.

- -dfmdb *dfm_database*

  An optional argument set that loads a DFM database.

- -rev *revision_name*

  An optional argument set specifying the revision of the DFM Database to open. Used with the -dfmdb option. The *revision_name* is either a string or an integer, and is the output of the dfm::get_current_rev or dfm::list_revs command.

- -autostart

  An optional argument enabling the automatic execution of Tcl code within an SVRF rule file. See "Run Calibre YieldServer With calibre -dfm" on page 22 for more information about using this option.

- { -turbo [*number_of_processors*] [-turbo_all] } || { -turbo_litho [*number_of_processors*] } || { -turbo [*number_of_processors*] [-turbo_all] -turbo_litho [*number_of_processors*] } [-remote *host,host,...* || -remotefile *filename* || -remotecommand *filename count*]

  Optional arguments used for running in multithreaded (MT) and distributed processing (MTflex) modes. The -turbo option is generally recommended.

  See "Command Line Options Reference Dictionary" in the *Calibre Administrator's Guide* for details.

- -ys_hyper

  An optional argument enabling hyperscaling in multithreaded operation. Must be specified with -turbo. See "Hyperscaling" in the *Calibre Administrator's Guide* for complete information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -cmp

  An optional argument enabling access to commands in the cmp:: namespace. These commands can appear in a script specified with the -exec option. For more information, refer to the *Calibre CMPAnalyzer User's Manual*.

- -exec *script_filename* [*arg_list*]

  An optional argument set that specifies a Calibre YieldServer command script to execute. Anything that follows the -exec keyword is assumed to be either the name of a script or arguments to the script; hence, this option must appear last in the command line. The *arg_list* specifies arguments to be passed to the script named *script_filename*.

Description

This command invokes Calibre YieldServer. A database must be loaded in order for database manipulation or data retrieval to occur. Loading can be done with the -dfmdb option or the dfm::open_db command. A database revision can be specified with the -rev option or the dfm::open_rev command.

The -autostart and -exec options enable automatic execution of a command script that performs database operations. If an input script does not use the "exit" command, the YieldServer command shell remains open for interactive use after the final script command is executed.

If -autostart or -exec is used and there is a Tcl error in the associated command script, YieldServer terminates.

Calibre YieldServer can execute commands supplied through shell input redirection (sometimes called a "here" document) using either "<" or "<<". See the documentation of your shell for details. Tcl errors that occur using this method do not cause YieldServer to terminate. Alternatively, you can set the CALIBREYSRC environment variable to automatically execute a valid YieldServer command script at start time. The variable's value is the pathname of the script.

Using the YieldServer shell in interactive mode allows for testing of commands to see if they perform the desired functions. You can generate a command script for future runs by using the dfm::write_cmds command.

Examples

Example 1

This shows command invocation with a specified DFM database and a run script:

```
calibre -ys -dfmdb dfmdb -turbo -exec ys.script
```

Example 2

See DFM YS Autostart for how to set up a rule file with a YieldServer script and to use the -autostart option to run the script as part of a calibre -dfm job.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

This shows the use of a “here” document in csh. The puts command returns the names of cells instantiated in the primary cell of the DFM database.

```
% calibre -ys -dfmdb dfmdb << EOF
? puts "[dfm::list_children [dfm::get_top_cell]]"
? EOF
...
Opening database "dfmdb", revision "master"
Loading hierarchy and connectivity
CellA CellB


--- CALIBRE::DFM YIELD SERVER EXECUTION COMPLETED
```

Related Topics

Using a DFM Database in Calibre RVE

Run Calibre YieldServer With calibre -dfm

Run Calibre YieldServer With calibre -dfm

You can combine a calibre -dfm run with a calibre -ys run to automatically invoke Calibre YieldServer as soon as a DFM database has been created. Invoking Calibre YieldServer in this way lets you start with a layout database in any of the supported formats, generate a DFM database, then process that DFM database using a Calibre YieldServer script in a single batch run. It also allows you to update an existing DFM database in a calibre -dfm run and then post-process it with YieldServer.

One method is to use the DFM YS Autostart statement in the rule file. This statement specifies a YieldServer program to run after a DFM Database is generated. To use Calibre YieldServer this way, you must invoke it from the command line with the following syntax:

```
calibre -dfm -hier [dfm_args] -ys [ys_args] -autostart
```

The DFM database path is passed automatically from the DFM executive to the YieldServer executive. The *dfm_args* and *ys_args* argument sets are used *by both* the DFM and YieldServer executives when this command line form is used. The calibre -dfm command line is described under “calibre” in the *Calibre YieldAnalyzer and YieldEnhancer User’s and Reference Manual*. See “calibre -ys” for the complete YieldServer command line usage.

If you do not use DFM YS Autostart in your rule file, then -autostart is not used. A YieldServer script can be passed using the -exec option, and the script will be run in batch mode.

When an existing DFM database is specified on the command line for both the DFM and YieldServer runs, this invocation can be used:

```
calibre -dfm -hier -dfmdb name [dfm_args] rules && calibre -ys -dfmdb name
[ys_args]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In this case, the command lines are executed separately, so the options are distinct for each run. If the -autostart option is used with -ys in this case, the YieldServer script specified in DFM YS Autostart is executed before any other YieldServer script.

YieldServer script development is often best done in an interactive shell invoked as follows:

```
calibre -ys [-dfmdb filename]
```

This shell is similar to "tclsh" and supports Tcl 8.6 commands. When developing scripts, the dfm::help command is useful along with the -help option for individual commands.

Using a DFM Database in Calibre RVE

DFM databases can be opened in Calibre RVE for cross-probing results in a layout viewer. This procedure shows some of the basic features of Calibre RVE for DFM results.

If the DFM database is in the form of DFM Report Card, it is possible to run YieldServer commands in the Calibre RVE interface. See "Running Tcl Scripts in Calibre RVE" and "Calibre RVE for DFM Report Card" in the *Calibre RVE User's Manual* for additional information.

Prerequisites

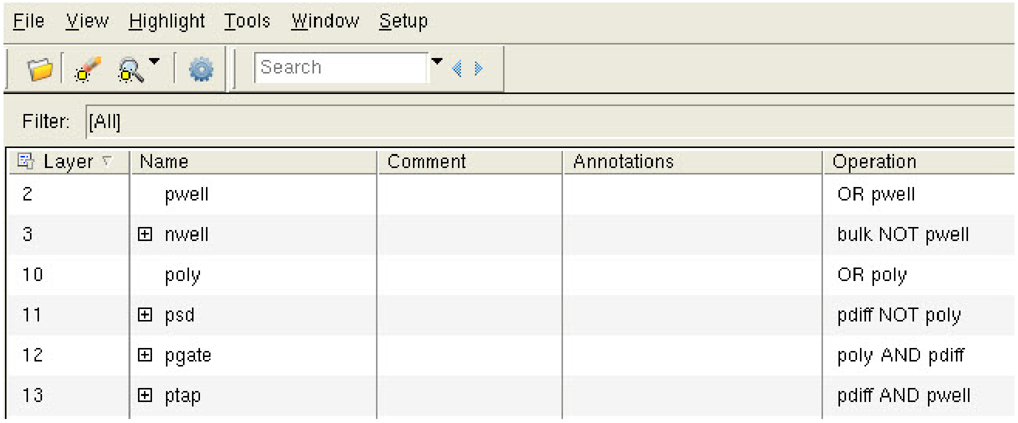
- Layout database.
- DFM database from a calibre -dfm run.

Procedure

1. Start Calibre DESIGNrev (calibredrv from the command line) and load your layout database file.
2. Start Calibre RVE using **Tools** > **Start RVE**.
3. Select **DFM** under Database Type.
4. Using the browse (**...**) button, choose the directory containing your DFM database, select DFM Databases from the **File Type** dropdown menu, and click **Open**. Alternatively, type the path to your DFM database in the **Database** text field and click **Open**.

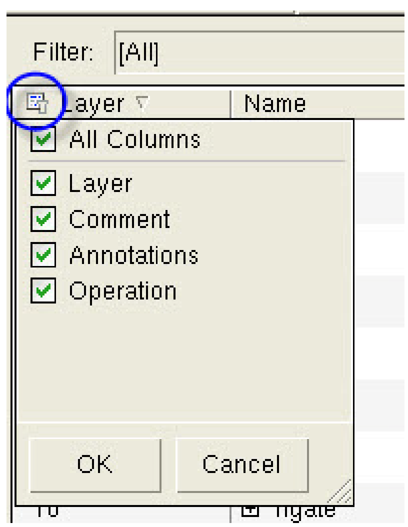
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A table of layers in the database opens, like this:



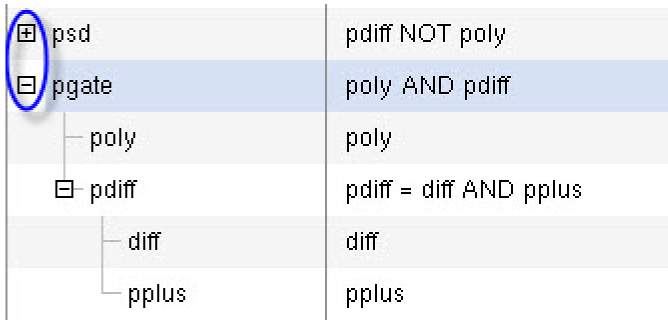
The Layer column has numbers assigned by the DFM module, and they do not necessarily match the rule file. Layer names are taken from the rule file. The Operation column shows the layer operation that generated the corresponding layer in the Name column. (OR *layer* indicates an original layer. Original layers are merged on read-in.) The Comment and Annotations columns are populated if comments and annotations are assigned to the layer at runtime.

5. (Optional.) Click the view column icon to select which columns in the table are viewable.

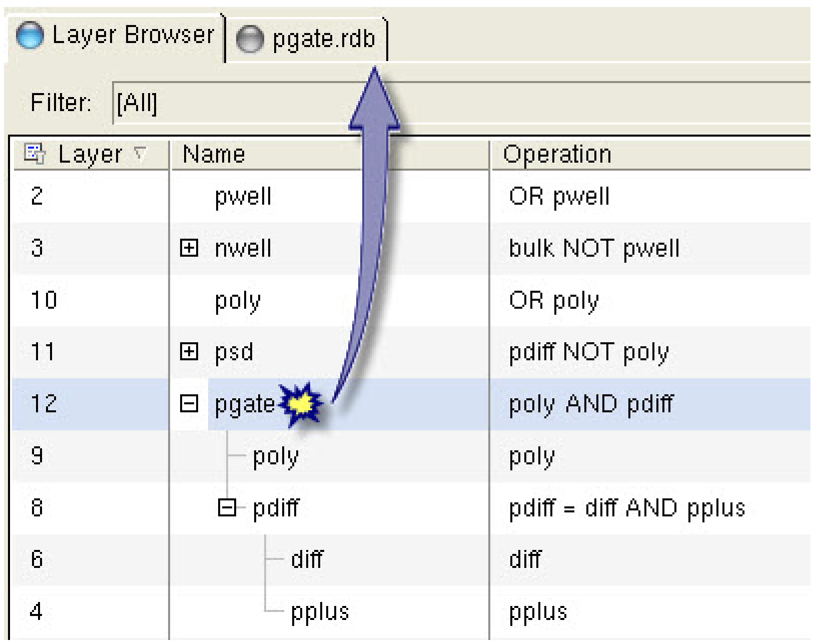


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

6. Expand or collapse layer derivation trees by clicking the +/- icons next to layer names.

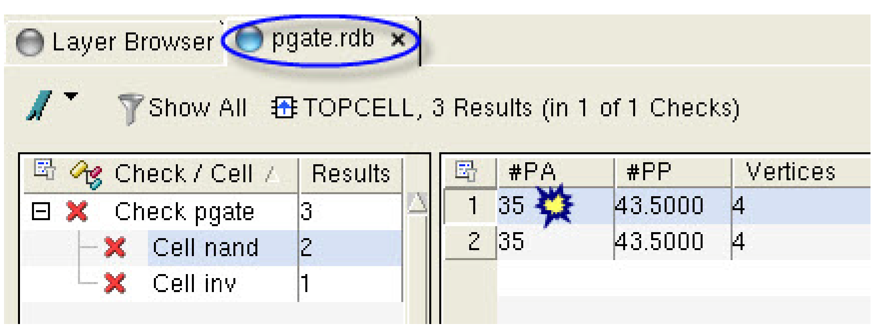


7. Double click a layer name at the head of a tree to open an RDB tab.



Layers that are saved in the DFM database are available for output to an RDB tab. Not all layers in a derivation tree are available for this use.

8. Select the RDB tab that you opened in the previous step. Choose any of the results to highlight in your layout viewer, just as with a DRC results database.



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example Calibre YieldServer Scripts

Calibre YieldServer example scripts. All YieldServer scripts are written in Tcl.

Example 1

This script returns the net names and node numbers associated with layers M1 and M2. Connectivity extraction information must be available in the database, which it is by default if the rule file uses connectivity statements. The script assumes a DFM database is not open at the time of execution. If one is, the dfm::open_db and dfm::close_db statements are not needed.

```
dfm::open_db dfmdb
set iter_M1 [dfm::get_geometries M1]
set iter_M2 [dfm::get_geometries M2]

puts "\nNet names and node numbers associated with M1:"
puts "\n\nnet name\tnode number\n--------\t-----------"
while {$iter_M1 ne ""} {
    set net_name [dfm::get_data $iter_M1 -net_name]
    set node_number [dfm::get_data $iter_M1 -node]
    puts "$net_name\t\t$node_number"
    dfm::inc iter_M1
}
puts "\nNet names and node numbers associated with M2:"
puts "\n\nnet name\tnode number\n--------\t-----------"
while {$iter_M2 ne ""} {
    set net_name [dfm::get_data $iter_M2 -net_name]
    set node_number [dfm::get_data $iter_M2 -node]
    puts "$net_name\t\t$node_number"
    dfm::inc iter_M2
}
puts "\n"
dfm::close_db
exit -force
```

Example 2

This script writes the contents of a DFM database to an OASIS® [1] file. Error and edge layers are converted to polygon layers before being written out with properties.

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.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Load the DFM database
dfm::open_db dfmdb

# set up list variables
set type3_layer [list]
set type2_layer [list]

# Write the layers to an RDB file.
set layers [dfm::get_layers]
while {$layers ne "" } {
  set name [dfm::get_data $layers -layer_name]
  puts "\nwriting layer $name"

  # get list of derived error layers to convert to polygon
  # or write out if polygons
  set type  [dfm::get_data $layers -layer_type]
  puts "type is $type"
  if {$type == 3 } {
    lappend type3_layer $name
  } elseif {$type == 2 } {
    lappend type2_layer $name
  } else {
    dfm::write_oas -layer_info [list $name 300 0 ] -write_properties \
      -file ${name}.oas
  }
  dfm::inc layers
}

# write out derived error layers as polygons
set typ3_iter 1
foreach type3 $type3_layer {
  puts "Writing out type 3 $type3"
  dfm::new_layer -svrf "reg2$typ3_iter = DFM COPY \"$type3\" REGION"
  dfm::write_oas -layer_info "reg2$typ3_iter 300 0"  -write_properties \
    -file ${type3}.oas
  incr typ3_iter
}
# write out derived edge layers as polygons using EXPAND EDGE
set typ2_iter 1
foreach type2 $type2_layer {
  puts "Writing out type 2 $type2"
  dfm::new_layer -svrf "regt2$typ2_iter = EXPAND EDGE \"$type2\" \
    OUTSIDE BY 0.004 CORNER FILL"
  dfm::write_oas -layer_info "regt2$typ2_iter 300 0"  -write_properties \
    -file ${type2}.oas
  incr typ2_iter
}
dfm::close_db
exit -force
```

See “Calibre YieldServer Example Scripts” on page 307 for additional examples.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 2
Working with DFM Databases

A DFM database is created during a calibre -dfm run per the DFM Database statement in the rule file. This is the most frequently used database by Calibre YieldServer. A small number of YieldServer commands support a Mask SVDB Directory database as input. These are not covered in this chapter.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DFM Database Generation

You generate a DFM database by running calibre -dfm -hier.

Invoking the Calibre hierarchical engine with the -dfm option causes it to write its results in the form of a DFM database rather than a DRC results database or an RDB. For more details about invocation parameters, see "Calibre DFM Command Line Invocation" in the *Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual*.

The rule file used as input to a -dfm run is similar to a rule file used with the -drc or -lvs options, with the following conditions:

- The rule file must contain a DFM Database specification statement.
- Device recognition is supported. Note that there are some constraints on this as discussed under "Connectivity Extraction and Device Recognition Using -dfm" on page 46.
- Generating a separate DFM RDB database is unsupported. All database output is directed to the DFM database as described in "Database Output Using calibre -dfm" on page 33.

DFM Database Generation from a DRC Rule File

DRC output can be directed to a DFM database by running the DRC rules with the -dfm option.

At minimum, you must add the following line to your DRC rule file:

```
DFM DATABASE path_to_dfm_database
```

The configuration of this statement affects which layers get written to the database. By default, derived and connect layers that are required for the run are written to the database. Statement options such as OVERWRITE are frequently useful.

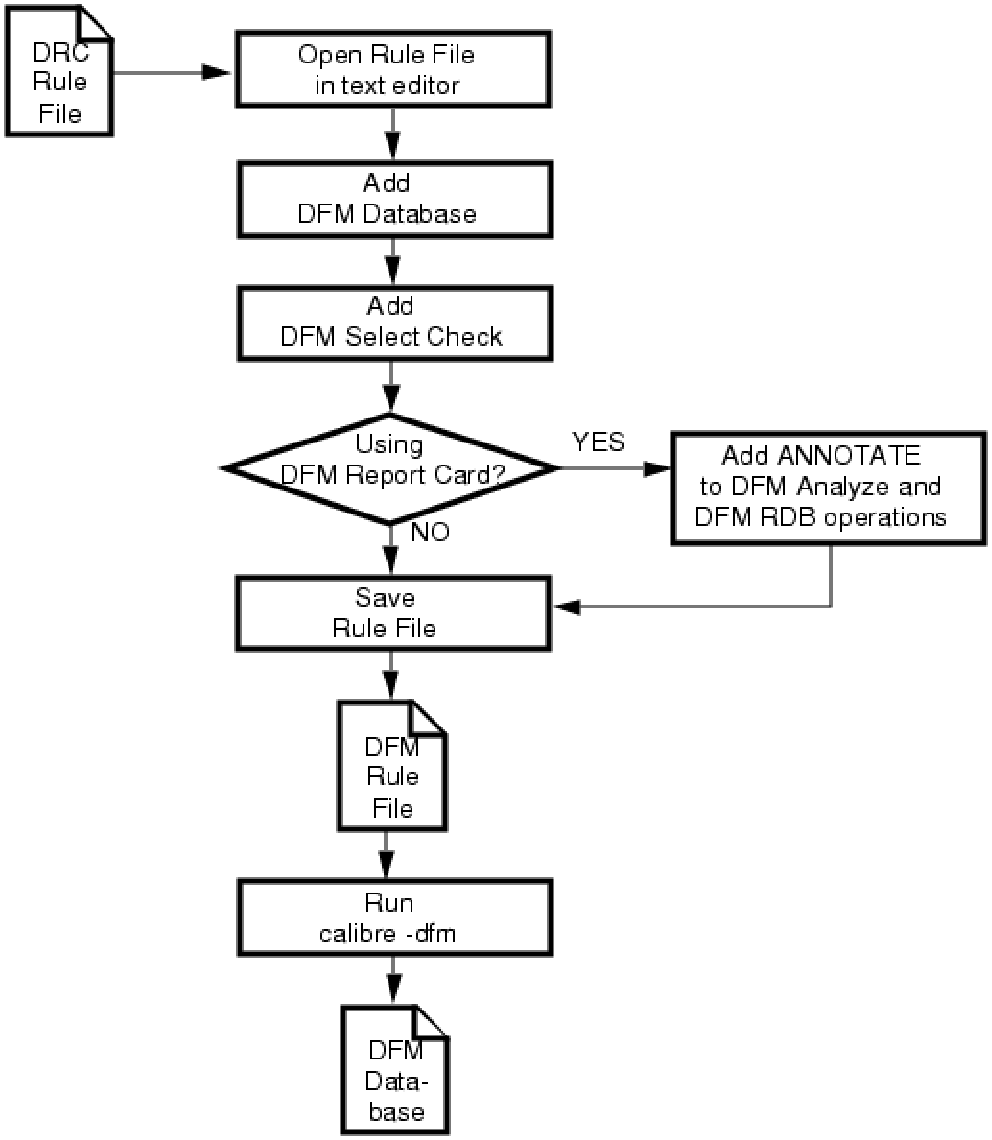
You can explicitly specify checks to execute using DFM Select Check:

```
GROUP group_a check1 check2
DFM SELECT CHECK group_a     //Runs all checks in group_a.
```

Otherwise the usual DRC rule check selection statements are used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-1. Generating a DFM Database From an nmDRC Rule Deck



Density Output to a DFM Database

Density input layers are saved in a -dfm run by using the DFM Database DENSITY keyword.

The outputs generated by the RDB [ONLY] and PRINT options of the Density operation are saved as additional layers in the DFM database. This feature applies to both calibre -dfm and to the dfm::new_layer command, which can be used to create Density layers in Calibre YieldServer. These additional layers can be accessed in YieldServer and output to an ASCII RDB file of the same format as the one produced by the Density operation in Calibre nmDRC. The dfm::write_rdb command can write an ASCII RDB from a YieldServer script.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The additional DFM database layers produced are as follows, where *output_layer* is the name of the layer the operation created:

- A layer named *output_layer_detail* is always produced.
- A layer named *output_layer_detail_print* is also produced whenever the Density statement includes the COMBINE option.

In Calibre YieldServer, the RDB output is written by invoking dfm::write_rdb as follows, where *filename* is the name of the RDB created:

```
dfm::write_rdb -layer output_layer_detail -file filename
```

For PRINT output, the -print option is added. Note that the layer used is still the *output_layer*_detail layer, not the *output_layer*_detail_print layer:

```
dfm::write_rdb -print -layer output_layer_detail -file filename
```

The RDB and PRINT output is written using the options (WINDOW, ABSOLUTE, COMBINE, and so on) that were specified in the Density statement that produced the layer. However, both RDB and PRINT output can be written by the dfm::write_rdb command, as long as either option was specified for the Density statement in the calibre -dfm run.

Net Area Ratio Output to a DFM Database

When a Net Area Ratio operation is specified with the RDB [ONLY] option in a rule file and calibre -dfm is used, an additional layer named *output_layer*_detail is saved to the DFM database (where *output_layer* is the name of the layer the operation produces). This _detail layer contains the information necessary both to perform various types of analysis and to write an RDB that is similar to the one produced with calibre -drc when the RDB [ONLY] option is specified.

In Calibre YieldServer, you can write an RDB containing the results of the Net Area Ratio operation by using the dfm::write_rdb command as follows, where *filename* is the name of the RDB produced:

```
dfm::write_rdb -layer output_layer_detail -file filename
```

The following are important notes regarding this functionality:

- None of the *RDB_options* valid with dfm::write_rdb (-nodal, -nopseudo, -noempty, and so on) are valid when writing a Net Area Ratio _detail layer to an RDB. The -append option is valid.
- The order of polygons written to the RDB using dfm::write_rdb may differ from those in the RDB produced with a calibre -drc run. All property names and values are identical.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- When incremental connectivity is enabled in a calibre -dfm run, new connect zones (and corresponding revisions) can be created. The _detail layers present in a connect zone are not carried forward into subsequent connect zones.
- The dfm::write_rdb command does not write empty Net Area Ratio _detail layers to the output RDB.

Database Output Using calibre -dfm

Many elements of a layout design are saved to a DFM database in a calibre -dfm run. The layers that get saved are governed by DFM Database keywords.

- Cell hierarchy.
- Connectivity layers when connectivity extraction is performed. Node IDs and net names are preserved.
- Layer attributes, such as name, type, configuration, derivation layers, and generating operation type.
- Device layers and information, when the DEVICES keyword is used.
- Device pin locations, when the PINLOC keyword is used.
- Original layers used in the run, when the ORIGINAL keyword.
- All input Density layers, when the DENSITY keyword is used.
- All input DFM Measure layers, when the MEASURE keyword is used.
- All input DFM Analyze layers, when the ANALYZE keyword is used.
- All input DFM Property layers, when the PROPERTY keyword is used.
- Copy of the rules.

Output operations in a rule check typically write their results as a DFM database layer having a system-generated name with the following format:

check_name::<*n*>

where:

- *check_name* is the name of the check containing the output operation.
- *n* is the ordinal value of the output operation within the check.

There are exceptions to the preceding behavior, however.

DFM RDB operations save their input layers to the DFM database without using the *check_name*::<*n*> format. (The TRANSITION keyword is not supported in -dfm mode, however.)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Density RDB and PRINT outputs to the database are discussed under "Density Output to a DFM Database" on page 31.

Net Area Ratio RDB outputs to the database are discussed under "Net Area Ratio Output to a DFM Database" on page 32.

DFM Analyze operations output two layers to the database:

- A polygon layer containing all the capture areas evaluated by the check. These are either rectangular capture windows or the bounding boxes of the cells that were evaluated. Note that if any capture areas abut or overlap, they are merged within this layer.

  The output layer name follows the *check_name*::<*n*> convention mentioned previously.

  This layer is created regardless of whether the DFM Analyze operation includes the RDB or RDB ONLY keywords. However, if RDB ONLY is specified, this layer is empty.

- A polygon layer containing those polygons and properties that would have been written to the RDB when running with the -drc option. There is one polygon per capture area. Note that these are not merged, even if they abut or overlap.

  This layer is saved to the DFM database with a system generated name created using the following format:

  *output_layer*_detail

  where *output_layer* is the name of the layer the operation created.

The Copy operation creates a copy of an original or derived layer without using the *check_name*::<*n*> format; however, DFM Copy does use that format.

Multiple Layer Data Save Operations

When a rule file contains operations that would save the same layer to the DFM database multiple times, only the first operation writes the layer to the DFM database.

Assume your rule file contains the following code:

```
DFM DATABASE dfmdb OVERWRITE

new_layer = diff AND pwell
new_layer {COPY new_layer}
new_layer2 = COPY new_layer
new_layer2 {COPY new_layer2}
```

When this rule file is processed using the -dfm option, the new_layer check's Copy operation causes new_layer to be written to the DFM database. The new_layer2 check's Copy operation does not cause new_layer2 to be written to the DFM database because new_layer2 is the same

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

as new_layer. If you open the *dfmdb* results in Calibre RVE, only new_layer appears in the database:

Layer	Name
20	⊟ new_layer
6	diff
1	pwell
21	⊟ new_layer2
20	⊟ new_layer
6	diff
1	pwell

The behavior is the same for the DFM RDB operation, except when the ANNOTATE keyword is used. DFM RDB ANNOTATE operations that reference the same layer name that is output in some other rule check cause a separate layer to be added to the DFM database because an annotated layer is different from a non-annotated layer, even if they share the same name in the rule file. See "Annotations in DFM Databases" for more information.

DFM Database Metadata

Both DFM and RDB databases allow users to attach metadata to some types of objects. Metadata can be stored as DFM annotations, DFM properties, check comments, or check names.

These are the usages:

- **Types of Data** — DFM properties can be strings or vectors. Other types of metadata are limited to string values only.
- **Types of Objects** — DFM properties are attached to individual geometric objects. Check names and comments can only be attached to layers. Annotations can be attached to geometric objects, layers, or databases.
- **Database Support** — RDBs support properties, check names, and comments. DFM databases support all metadata types.
- **Annotations** — A layer or database can have an unlimited number of annotations, each stored as a name and value pair. The annotation names need not be unique so it is possible for a layer to have multiple attributes with the same name but different values. Further details about annotations are given under "Annotations in DFM Databases" on page 37.
- **Properties**— A geometric object can have an unlimited number of properties, each stored as a name and value pair. Each property name must be unique.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 2-1. Summary of Metadata Stored In Databases

Metadata	Databases	Target Objects	Data Type	Stored as...
DFM Annotation	DFM database	Layer Database	String	Name and value pair. Allows multiple values for a single name.
DFM Property	DFM database DFM RDB	Geometric object	Numeric netID String Vector	Name and value pair. Only one value allowed for each name.
Check name	DFM database DFM RDB	Layer	String	0 or more values.
Check comment	DFM database DFM RDB	Layer	String	0 or more values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Annotations in DFM Databases

Annotations are name and value pairs associated with data in a DFM database. Annotations are associated with either individual layers or a database revision.

You add annotations to DFM Database output by using the ANNOTATE keyword in commands that support it. This keyword is used only in calibre -dfm -hier runs.

Table 2-2. DFM Operations that Support Annotations

SVRF Command	What is Annotated
DFM Analyze	Layers containing the actual check results (the *detail* layer).
DFM RDB	Input layers
DFM Database	DFM database

The ANNOTATE syntax is this:

ANNOTATE '[' ***name*** = “***value***” ']'

where:

- Brackets [ ] are required.
- The ***name*** can be a quoted or unquoted string.
- The ***value*** must be a quoted string.

Annotations are processed in YieldServer using “Annotation Commands” on page 52.

ANNOTATE and DFM Analyze

By default, when you run DFM Analyze in calibre -dfm, the operation produces two layers in the DFM database. One contains the merged capture areas, and the other contains the unmerged capture areas with the analysis results attached as properties.

If the ANNOTATE keyword is present, the layer containing the unmerged capture areas also contains the specified annotations.

Table 2-3. Information from DFM Analyze

Mode	Capture Area Geometries	Analysis Results (values)	Annotations
-drc	Written to the DRC Results Database.	Written to the RDB as properties attached to the individual check result (capture area) polygons.	Not saved.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 2-3. Information from DFM Analyze (cont.)

Mode	Capture Area Geometries	Analysis Results (values)	Annotations
-dfm	Written to the DFM database as an unannotated, merged layer.	Written to the DFM database as *properties* on *individual geometries* on the layer containing the unmerged capture areas.	Written to the DFM database as *annotations* on the *layer* containing the unmerged capture areas.

This example calculates the sum of the GaSpc and GaExt geometries for each capture window using the COUNT measurement function:

```
Analyze_sample {
   DFM ANALYZE GaSpc GaExt >= 0 WINDOW 5
      [COUNT(GaSpc) + COUNT(GaExt)]
      RDB ONLY "./output/yield.rdb"
      ANNOTATE ["DFM_LEVEL" = "WINDOW"]
      ANNOTATE ["DFM_BIN" = "b3"]
      ANNOTATE ["DFM_METRIC" = "DFM_Score"]
      ANNOTATE ["DFM_RULE" = "Chip_Total"]
      ANNOTATE ["DFM_TYPE" = "rra"]
      ANNOTATE ["DFM_GROUP" = "ALL"]
      ANNOTATE ["DFM_PRIORITY" = "2"]
}
```

When processed by -drc, this code creates a derived layer named Analyze_sample in memory, and the layer contains all the capture windows merged into one. While this layer can be written to the DRC Results Database, it is not generally useful to do so. The actual analysis results are written to the RDB, with values saved as properties (DV, DC GaSpc, and DC GaExt) for the individual capture windows.

When processed by -dfm, the preceding code generates a layer called “Analyze_sample::<1>” containing the merged capture windows and a second layer called “Analyze_sample::<1>_detail” containing one polygon per capture area, each with the properties DV, DC GaSpc, and DC GaExt.

The layer “Analyze_sample::<1>_detail” is annotated with the seven annotations specified in the check:

```
"DFM_LEVEL" = "WINDOW"
"DFM_BIN" = "b3"
"DFM_METRIC" = "DFM_Score"
"DFM_RULE" = "Chip_Total"
"DFM_TYPE" = "rra"
"DFM_GROUP" = "ALL"
"DFM_PRIORITY" = "2"
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ANNOTATE and DFM RDB

Using the DFM RDB operation, annotations are added to a set of geometric layers in the DFM database.

There are two ways to use a DFM RDB with annotations:

- Add annotations when a layer is first saved to the DFM database by supplying the ANNOTATE keyword in the DFM RDB operation.
- Add annotations to a layer that has already been saved to the DFM database by supplying additional DFM RDB operations that reference that layer. Specifically, when using -dfm, the DFM RDB... ANNOTATE operation can be executed multiple times for the same input layers. Note that if the layer already exists in the DFM database, only the annotation portion of the command is evaluated; the annotations are added to the layer in the DFM database as if they had appeared in the original statement.

Results generated using -dfm can be different from the results generated using -drc with the same rule file. When using -drc, executing DFM RDB... ANNOTATE command multiple times for the same input layers results in multiple instances of the input layers in the RDB. None of these would be annotated because annotations are supported only by -dfm. You can override the default -drc behavior by defining the RDB filename as “NULL”, in which case, the operation is not executed by the DRC module.

Example 1

When processed by -drc, the layers GaSpc and GaExt are written to the RDB *my_file.rdb*.

```
Rdb_sample {
       DFM RDB GaSpc my_file.rdb GaExt
          ANNOTATE ["DFM_LEVEL" = "WINDOW"]
          ANNOTATE ["DFM_BIN" = "b3"]
          ANNOTATE ["DFM_METRIC" = "DFM_Score"]
          ANNOTATE ["DFM_RULE" = "Chip_Total"]
          ANNOTATE ["DFM_TYPE" = "rra"]
          ANNOTATE ["DFM_GROUP" = "ALL"]
          ANNOTATE ["DFM_PRIORITY" = "2"]

       DFM RDB GaSpc NULL ANNOTATE [ DFM_CATEGORY = "SPACING" ]

       DFM RDB GaExt NULL ANNOTATE [ Ext = "" ]
    }
```

When processed by -dfm, the layers GaSpc and GaExt are saved to the DFM database, with all of the annotations present in the first DFM RDB statement. The annotation “DFM_CATEGORY” = “SPACING” is also added to the layer GaSpc. The annotation “Ext” with an empty string as the value is added to the layer GaExt.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

Assume you want the annotated layers saved to the DFM database when running with -dfm, but you do not want these layers written to any RDB when running with -drc.

To do this, set the file name for the first DFM RDB to NULL.

```
-----------------------------------
Rdb_sample {
       DFM RDB GaSpc NULL GaExt
          ANNOTATE ["DFM_LEVEL" = "WINDOW"]
          ANNOTATE ["DFM_BIN" = "b3"]
          ANNOTATE ["DFM_METRIC" = "DFM_Score"]
          ANNOTATE ["DFM_RULE" = "Chip_Total"]
          ANNOTATE ["DFM_TYPE" = "rra"]
          ANNOTATE ["DFM_GROUP" = "ALL"]
          ANNOTATE ["DFM_PRIORITY" = "2"]

       DFM RDB GaSpc NULL ANNOTATE [ DFM_CATEGORY = "SPACING" ]

       DFM RDB GaExt NULL ANNOTATE [ Ext = "" ]
}
```

DFM Report Card Annotation Usage

When you open a DFM database in Calibre RVE, that application uses annotations on layer data to control the contents and behavior of the DFM Report Card.

Refer to "DFM Report Card Controls" in the *Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual* for a list of the annotations recognized by the DFM Report Card.

Sample CAA Rule File for calibre -dfm

The following rule file finds regions in a layout that are at risk for shorts on the poly layer due to manufacturing defects. This is used in critical area analysis (CAA) and incorporates annotations. The generated results can be opened in Calibre RVE and Calibre YieldServer.

This example demonstrates the DFM expression PROPERTY function.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
LAYOUT SYSTEM  GDSII
LAYOUT PATH    "./my_file.gds"
LAYOUT PRIMARY "*"
LAYOUT DEPTH   ALL

DRC CELL NAME YES XFORM ALL
DRC MAXIMUM RESULTS  ALL

/////////////////////////////////////////////////////////////////////////
// ORIGINAL LAYERS
/////////////////////////////////////////////////////////////////////////

LAYER NWELL      3   //  N-Well
LAYER OD         1   //  thin oxide
LAYER POLY1     41   //  Poly Si
LAYER PIMP      11   //  P+ S/D Implantation
LAYER NIMP      12   //  N+ S/D Implantation
LAYER CONT      39   //  Contact
LAYER M1        46   //  Metal-1
LAYER M1TXT    101   //  metal1 text
LAYER EMPTY    999   //  empty for meta

/////////////////////////////////////////////////////////////////////////
// DERIVED LAYERS
/////////////////////////////////////////////////////////////////////////

field_poly = POLY1 NOT OD
gate       = POLY1 AND OD
sd         = OD NOT POLY1
end_cap    = (field_poly NOT INTERACT CONT) INTERACT gate == 1

CONNECT M1 POLY1 sd BY CONT
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//*******************************************************************
// POLY SHORTS Critical Area Analysis
//*******************************************************************

poly_short = DFM CRITICAL AREA POLY1 SHORT S 0.09 OUTPUT
poly_short_DFM_Score = DFM PROPERTY poly_short
                        [DFM_SCORE = (AREA(poly_short)/0.05> 1) ? 1 :
                         AREA(poly_short)/0.05 ]

Poly_Shorts_DFM_Score#caa# {

@ Full chip analysis

DFM ANALYZE poly_short_DFM_Score >= 0

[PROPERTY(poly_short_DFM_Score,DFM_SCORE)/COUNT(poly_short_DFM_Score)]
        RDB ONLY DV COORD  "designerDFM.rdb"
        ANNOTATE [DFM_RULE = "Poly_Shorts_DFM_Score" ]
        ANNOTATE [DFM_GROUP = " " ]
        ANNOTATE [DFM_METRIC = "Dfm_Score" ]
        ANNOTATE [DFM_BIN = " " ]
        ANNOTATE [DFM_LEVEL = "chip" ]
        ANNOTATE [DFM_PRIORITY = "1" ]
        ANNOTATE [DFM_TYPE = "caa" ]
        ANNOTATE [DFM_SORT = "LO_TO_HI" ]
}

Poly_Shorts_DFM_Score#caa#_c {

@ By Cell analysis

DFM ANALYZE poly_short_DFM_Score >= 0 BY CELL NOPSEUDO
[(COUNT(poly_short_DFM_Score) > 0) ?

PROPERTY(poly_short_DFM_Score,DFM_SCORE)/COUNT(poly_short_DFM_Score) : 0]
        RDB ONLY DV COORD  "designerDFM.rdb"
        ANNOTATE [DFM_RULE = "Poly_Shorts_DFM_Score" ]
        ANNOTATE [DFM_GROUP = " " ]
        ANNOTATE [DFM_METRIC = "Dfm_Score" ]
        ANNOTATE [DFM_BIN = " " ]
        ANNOTATE [DFM_LEVEL = "cell" ]
        ANNOTATE [DFM_PRIORITY = "1" ]
        ANNOTATE [DFM_TYPE = "caa" ]
        ANNOTATE [DFM_SORT = "LO_TO_HI" ]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Poly_Shorts_DFM_Score#caa#_w {

@ By Window analysis

DFM ANALYZE poly_short_DFM_Score >= 0 WINDOW 1
[(COUNT(poly_short_DFM_Score) > 0) ?

PROPERTY(poly_short_DFM_Score,DFM_SCORE)/COUNT(poly_short_DFM_Score) : 0]
        RDB ONLY DV COORD  "designerDFM.rdb"
        ANNOTATE [DFM_RULE = "Poly_Shorts_DFM_Score" ]
        ANNOTATE [DFM_GROUP = " " ]
        ANNOTATE [DFM_METRIC = "Dfm_Score" ]
        ANNOTATE [DFM_BIN = " " ]
        ANNOTATE [DFM_LEVEL = "window" ]
        ANNOTATE [DFM_PRIORITY = "1" ]
        ANNOTATE [DFM_TYPE = "caa" ]
        ANNOTATE [DFM_SORT = "LO_TO_HI" ]
}

Poly_Shorts_DFM_Score#err# {

@ Error layer for drill down

DFM RDB poly_short_DFM_Score "designerDFM.rdb" ALL CELLS
        CHECKNAME Poly_Shorts_DFM_Score#err#
        ANNOTATE [DFM_RULE = "Poly_Shorts_DFM_Score" ]
        ANNOTATE [DFM_GROUP = " " ]
        ANNOTATE [DFM_METRIC = "Dfm_Score" ]
        ANNOTATE [DFM_BIN = " " ]
        ANNOTATE [DFM_LEVEL = "error" ]
        ANNOTATE [DFM_PRIORITY = "1" ]
        ANNOTATE [DFM_TYPE = "caa" ]
        ANNOTATE [DFM_SORT = "LO_TO_HI" ]
}

GROUP DFM_CHECKS ? // Group all the checks and name the group
                   // DFM_CHECKS.
DFM SELECT CHECK DFM_CHECKS  // Select DFM_CHECKS for execution
DFM DATABASE dfmdb OVERWRITE [ANALYZE ORIGINAL]
```

DFM Database Revisions

To modify and save DFM databases, Calibre YieldServer supports database revisions.

The following table discusses database state definitions. These states govern whether a database can be revised.

Table 2-4. DFM Database Revision States

State	Description	Valid Commands
Frozen	A finalized revision. Once a revision is frozen, it can no longer be modified, but can be used as a parent for new revisions.	dfm::open_rev, dfm::close_db, dfm::create_rev

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

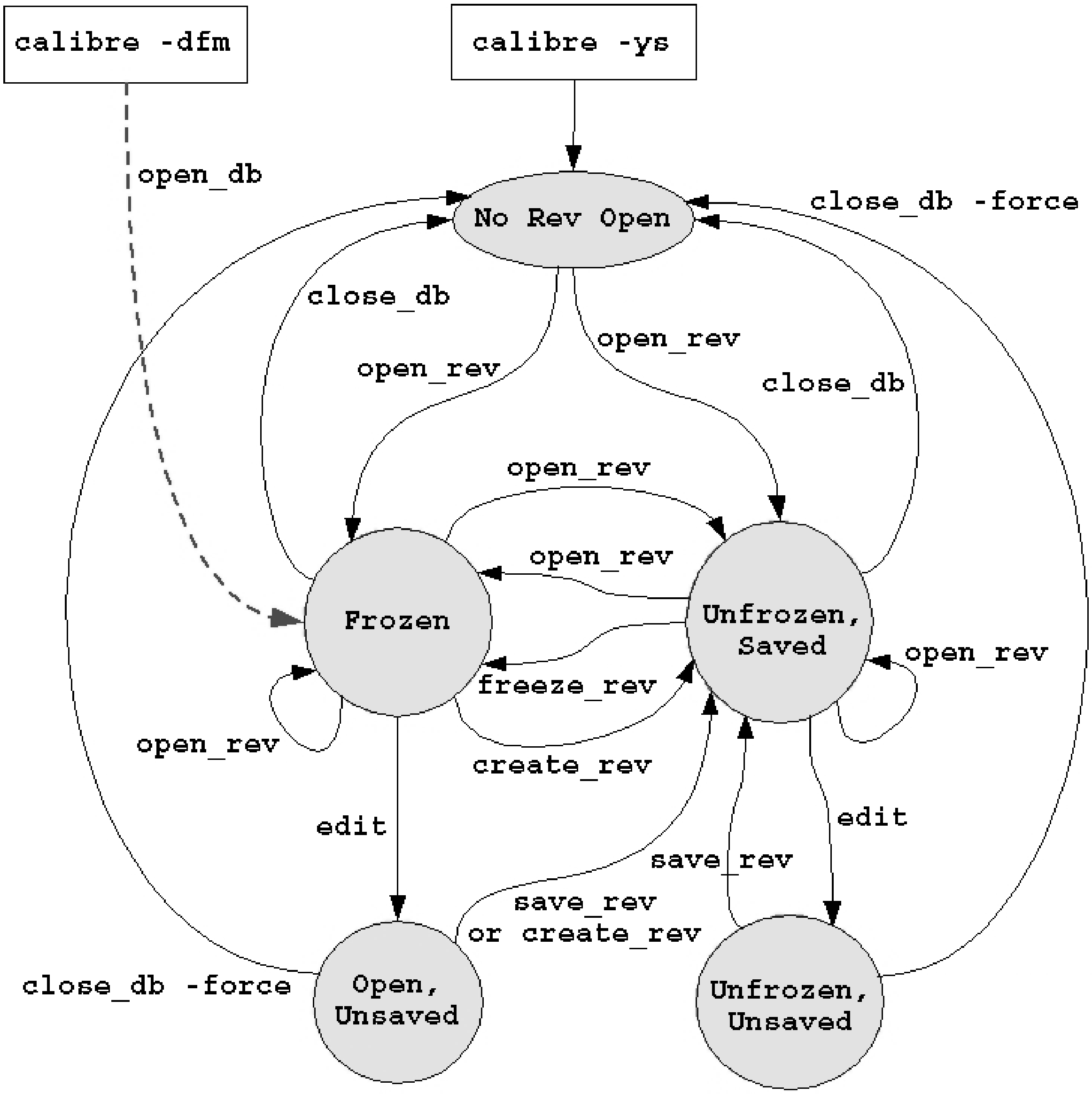
Table 2-4. DFM Database Revision States (cont.)

State	Description	Valid Commands
Unfrozen, Saved	A revision that has no unsaved changes but has not been frozen.	dfm::open_rev, dfm::freeze_rev, dfm::close_db
Open, Unsaved	A revision created from a frozen revision that has unsaved changes.	dfm::create_rev, dfm::save_rev, dfm::close_db -force
Unfrozen, Unsaved	A revision that has unsaved changes and has not been frozen.	dfm::save_rev, dfm::close_db -force

Database revision states are shown in Figure 2-2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-2. DFM Database Revision State Diagram



See “NO TITLE” for summary information.

If a database does not have sufficient write permissions for a user, it is opened in read-only mode. This mode permits many YieldServer functions to work except any that modify the database.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connectivity Extraction and Device Recognition Using -dfm

Connectivity extraction is performed in a calibre -dfm run if nodal information is needed by any checks. Connectivity information is then stored in the DFM database by default. The calibre -dfm executive module converts connectivity and device layers with exclusive flat instantiations to exclusive hierarchical instantiations before performing connectivity extraction and device recognition, respectively.

The DFM Database DEVICES keyword causes device layers to be stored in the DFM database. This also occurs when an operation that requires devices (Pathchk and Device Layer) is used in a rule check. Information about device layers, pins, properties, and Device statements is accessible from the database.

If a DFM database has no connectivity model, it can be added during the YieldServer session by using the dfm::new_layer -svrf option with appropriate connectivity statements.

Note

LVS Push Devices SEPARATE PROPERTIES YES and LVS Preserve Box Cells YES are not supported in circuit extraction in calibre -dfm or -ys.

DFM Database Incremental Connectivity

DFM databases support incremental connectivity in the same way as in DRC.

The specification statements DRC Incremental Connect and DRC Incremental Connect Warning can be used to control incremental connectivity for DFM databases. Layers saved to a DFM database have the same connectivity[1] as they do in an equivalent DRC run when accessed in the appropriate database revision.

For background information on incremental connectivity, refer to “Incremental Connectivity and Antenna Checks” in the *Calibre Verification User’s Manual*.

Incremental connectivity for DFM databases follows a number of rules:

1. For each connect zone in which node numbers have been saved, a DFM database revision is created. This revision is a child of the last such revision (that is, no branching of revisions is involved), and contains the following:
    - All layers that were saved in the previous revisions (see Rule 3).
    - All layers that have been saved since the previous revision.

1. Note that node numbers may be different between running calibre in -drc mode versus -dfm mode due to differences in hierarchy optimization and the order in which operations are executed. Net names, however, are the same between -drc and -dfm runs. Alias layers may not be saved to the database if they are not really needed. This is for efficiency.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The connectivity model for the most recent connect zone.

Layers saved in the revision corresponding to a connect zone are said to be saved in the connect zone.

2. Node numbers are considered to be saved in a connect zone only if a nodal layer[2] or a layer with nodal properties[3] has been saved since connectivity was extracted for the connect zone but before connectivity is extracted for the subsequent connect zone.
3. Once saved, layers are present in all subsequent revisions. Only the existence or values of node numbers and nodal properties on the layers may change.

Note

Layers that have not been modified in a new revision are hard-linked into the revision rather than re-saved. When a layer is changed from nodal in one revision to non-nodal in another revision, the layer file can also be hard-linked. The DFM database ignores the node numbers and nodal properties when the layer is loaded into revisions in which the node numbers do not correspond to the connectivity model.

4. Node numbers and nodal properties on a layer are not accessible when the layer is loaded into a revision that has a different connectivity model than the one with which the node numbers or properties were saved.

   When a nodal layer is saved with node numbers, it is changed to a non-nodal layer in the revision for the next connect zone, unless it is connected and saved in the latter connect zone, in which case it is re-saved with new node numbers. Nodal properties are similarly changed, but cannot be re-created in subsequent connect zones.
5. Nodal properties created in a connect zone are removed from all layers before connectivity is extracted for the next connect zone.
6. Connect layers[4] may be saved in the connect zone in which they are produced or a later zone, but before the connect zone in which they are first connected. In the latter case, they are saved with or without node numbers, depending on how they are produced. Note that under incremental connectivity, a layer may be produced with net numbers in one connect zone and then become a Connect layer in a later connect zone, replacing the original net numbers.

   Connect layers may be saved to the DFM database explicitly or implicitly. Explicit saves are done with the DFM RDB statement or an unassigned Copy operation. Implicit saves are done when a layer is an original layer, a Connect layer, part of an unassigned operation other than Copy, or an input to a DFM Property or DFM Analyze operation

2. A nodal layer refers to any layer that is configured with node numbers; it can be a Connect layer or a layer where its connectivity is inherited from another layer.
3. Nodal properties refer to net properties (created with DFM Property BY NET), or properties of type netID or vnetID.
4. Connect layers refer to layers that are arguments in a Connect statement.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

when DFM Database PROPERTY or DFM Database ANALYZE is specified, respectively.

7. When the -hyper command line option is used, connect zones are still executed in the order in which they appear in the rule file, and connectivity-dependent operations execute in the same connect zone, though the order of operations within a connect zone may differ. However, operations that do not depend on connectivity can be executed during any connect zone. This can result in layers being saved in different revisions when hyperscaling is enabled—either earlier or later revisions. This can in turn cause a layer to be saved a different number of times with hyperscaling enabled versus hyperscaling disabled (and a different number of times between separate hyperscaling runs). Node numbers can even be saved when they would not have been saved in a run without hyperscaling, and vice-versa. Because of these variations, runs with hyperscaling can create different numbers of revisions compared to runs without. Additionally, different numbers of revisions can be created between separate hyperscaling runs.

8. If DFM Database ORIGINAL is specified, non-nodal original layers are saved implicitly in the connect zone in which they are produced. Original layers that are Connect layers are saved according to the rules for Connect layers.

9. The calibre -dfm run mode divides the rule file into connect zone sets. Connect zone sets begin at the start of the rule file and at the first Connect statement following a Disconnect statement. (Disconnect statements prior to the first Connect statement are ignored.) If DFM Database CONNECT (or ALL) is specified, all Connect layers from the previous connect zone set are saved implicitly in the last connect zone in a connect zone set. Note that unlike calibre -dfm runs with non-incremental connectivity, Connect layers are not saved implicitly following each set of Connect statements.

10. When a nodal layer is saved in the connect zone in which it acquired node numbers, it is saved with node numbers even if the operation causing it to be saved does not require the layer to have node numbers.

11. Between a Disconnect statement and the next Connect statement in the rule file, any nodal layers that are saved are saved with node numbers, even though the compiler does not allow access to connectivity in this region of the rule file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 3
Calibre YieldServer Reference

Calibre YieldServer provides a large set of commands for accessing and modifying a DFM database. Certain commands in the dfm:: scope can also be used in the Calibre Query Server Tcl shell. In that case, a Mask SVDB Directory is used instead of a DFM database.

Iterator Concepts

Certain Calibre YieldServer commands create or process iterators. An *iterator* is a Tcl construct that provides access to data in a DFM database. Not all data is accessed through iterators, but iterators are the most powerful data retrieval feature in Calibre YieldServer.

An iterator references one or more database objects. Each iterator gives access to a database object, and only within the defined context. Iterators are returned by YieldServer commands having the "dfm::get_" prefix, although not all commands with this prefix return an iterator.

An iterator is opaque; that is, you cannot see its contents without accessing them from some command that takes an iterator as input. A single-element iterator references a single object. Single-element iterators are frequently used to access information about an object with a known name. A multi-element iterator is an ordered list of elements, each providing access to a different database object. Multi-element iterators can be manipulated in a loop to access the objects referenced by the iterator elements.

The sequence of data in an iterator is not generally predictable nor meaningful. Using multi-element iterators is similar to "stepping" through an ordered list of pointers or handles that reference data (single-element iterators cannot be stepped forward). For example, the following code steps through each of the cells in the database, printing each cell name to the transcript:

```
set cells [dfm::get_cells]
while { $cells ne ""} {
    puts [dfm::get_data $cells -cell_name]
    dfm::inc cells
}
```

The iterator here is referenced by the variable name "cells". An empty string ("") signifies the end of iterator. The dfm::inc command steps through the iterator elements. The general form of this loop example is typical of iterator usage in a script.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Iterators can be empty when an iterator creation command does not find an object of interest. It is also possible an iterator can be stepped forward to the end of its sequence of values, where the returned value is empty, such as in the preceding example. If either of these cases is possible, you should test the iterator return value against the empty string ("").

Calibre YieldServer iterators can only be stepped through in a single direction (forward.) They must be re-generated after you have stepped through the entire list.

Note that you cannot create a copy of an iterator using the **set** command, so this does not work:

```
set cells [dfm::get_cells]
set copy_of_cells $cells; # DOES NOT WORK
```

Handling an iterator in a Tcl string context should be avoided as this creates performance problems. For example, do not do this:

```
set len [string length $iterator]; # bad
```

The most common Tcl commands that cause problems in this context are string, concat, and eval. Iterator values used as array keys should also be avoided.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Reference Dictionary

Calibre YieldServer commands can be grouped into families that perform closely related functions.

You can issue the command dfm::help [-gui] in the shell to see a list of all the commands.

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

Every command has a -help option which can be used to get syntax details.

Reference descriptions appear in the remainder of the chapter.

Note

Unless stated otherwise, the commands used in Calibre YieldServer require an open DFM database (use dfm::open_db from within the server or -dfmdb on the command line).

Here are the command classifications found in the following tables:

Annotation Commands

Connectivity Commands

Database Administration Commands

DFM Property Management Commands

Edge Collection Commands

Hierarchy Traversal Commands

Iterator Processing Commands

Layer Cluster Commands

Layer Management Commands

Layout Data Query Commands

Layout Database Output Commands

LVS and CCI Commands

Netlist Commands

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Server Administration Commands

Source Design Query Commands

Timer Commands

Annotation commands process annotations, which are informational objects associated with layers, databases, or database revisions. These are different from DFM properties but can serve a related purpose of classifying objects.

Table 3-1. Annotation Commands

Command	Description
dfm::add_annotation	Adds an annotation to a layer or database revision.
dfm::delete_annotation	Deletes annotations from a layer or database revision.
dfm::list_annotated_layers	Returns a list of layers having annotations that match the specified name and value pairs.
dfm::list_annotation_names	Returns a list of names of all annotations found on the specified layer or database.
dfm::list_annotation_values	Returns a list of the name and value pairs for annotations on the specified layer or database.
dfm::list_annotation_values_for_layers	Returns a list of annotation names and values for specified lists of layers and annotation names.
dfm::list_annotation_values_for_name	Returns a list of values associated with the specified annotation name.

Connectivity commands manage aspects of the electrical connectivity model. These commands are related to LVS and CCI Commands and Netlist Commands.

Table 3-2. Connectivity Commands

Command	Description
dfm::ascend_net	Returns an iterator referencing a net in the parent cell of the current placement.
dfm::create_filter	Creates a filter object.
dfm::descend_net	Returns an iterator referencing nets connected to lower-level placements in a cell placement.
dfm::disconnect	Deletes the current connectivity model.
dfm::get_connect_warnings	Returns a circuit extraction warning iterator.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Connectivity Commands (cont.)

Command	Description
dfm::get_data -net, -net_is_epin, -net_is_original, -net_name, -net_property, -netlist_net_name, -node, -port_dir, -port_info, -port_name	Returns the specified type of connectivity data for the current database object.
dfm::get_device_pins	Returns a device pin iterator.
dfm::get_lay_vs_netlist_net_name	Returns a list of layout cell net names that differ from the extracted netlist.
dfm::get_net_name	Returns the name of the net (if any) for a particular node number and cell.
dfm::get_net_shapes	Returns a geometry iterator referencing geometries on a specified net.
dfm::get_nets	Returns a net iterator.
dfm::get_pins	Returns a pin iterator referencing pins of a cell placement.
dfm::get_port_data	Returns information about ports.
dfm::get_ports	Returns a port iterator for a given cell.
dfm::net_is_epin	Returns 1 if a cell's net is connected farther up in the hierarchy and 0 otherwise.
The following command is documented in the *Calibre Query Server Manual*.	
qs::port_table	Returns a listing of information about layout cell ports.

Database administration commands manage various aspects making databases active, controlling versions and access, querying database attributes, and closing of databases.

Table 3-3. Database Administration Commands

Command	Description
dfm::chmod	Changes the permissions of a DFM database.
dfm::close_db	Closes the current database.
dfm::copy_db	Copies the source DFM database to a destination DFM database.
dfm::copy_svdb_to_dfmdb	Converts a Mask SVDB Directory to a DFM database.
dfm::create_rev	Creates a new revision of the current database.
dfm::delete_rev	Deletes the specified database revision.
dfm::freeze_rev	Causes a database revision to be unalterable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-3. Database Administration Commands (cont.)

Command	Description
dfm::get_current_rev	Returns the revision number or name of the current database revision.
dfm::get_db_creation_info	Returns information about the calibre -dfm run that created the DFM database.
dfm::get_db_name	Returns the name of the current database.
dfm::get_db_precision	Returns the current database precision.
dfm::get_default_rev	Gets the default revision for the currently open database.
dfm::get_revision_info	Returns revision information for a DFM database.
dfm::is_rev_frozen	Returns 1 if the current database revision is frozen and 0 otherwise.
dfm::list_revs	Returns a list of revisions for a DFM database.
dfm::open_db	Opens a DFM database.
dfm::open_rev	Opens a specified revision of the current database.
dfm::save_rev	Saves the current database revision.
dfm::set_default_rev	Defines the revision of the database to be opened by default.
dfm::update_rev_format	Creates a new revision by re-saving the current open (frozen) revision.
exit -force	Quits Calibre YieldServer. An open database with unsaved changes is closed without saving the changes.

DFM Property management commands process DFM properties.

Table 3-4. DFM Property Management Commands

Command	Description
dfm::add_geometry_property	Adds a property to a DFM Analyze _detail layer geometry identified by polygon number.
dfm::add_property	Adds a property to geometries on a layer or referenced by an iterator.
dfm::delete_property	Deletes properties from geometries on a layer.
dfm::get_data -cell_property, -geometry_property, -has_property, -is_vector_property, -net_property	Returns the specified type of property data for the current database object.
dfm::get_geometry_property	Returns a specified property value from a DFM Analyze _detail layer geometry.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-4. DFM Property Management Commands (cont.)

Command	Description
dfm::list_properties	Lists all properties defined on a layer or geometry.
dfm::v_minmax	Returns minimum and maximum values of a vector property.
dfm::v_sumprod	Returns sum and product values of a vector property.

Edge collections are useful in creating layers.

Table 3-5. Edge Collection Commands

Command	Description
dfm::add_geometry	Adds a single shape to an edge collection.
dfm::create_ec	Returns an edge collection Tcl object. Edge collections are cell-insensitive.
dfm::create_layer	Creates a layer from an edge collection.

Hierarchy traversal commands enable a program to process net and placement path relationships in the hierarchy.

Table 3-6. Hierarchy Traversal Commands

Command	Description
dfm::ascend_hierarchy	Returns an iterator referencing placements in the parent cell of the current placement.
dfm::ascend_net	Returns an iterator referencing a net in the parent cell.
dfm::ascend_path_context	Moves a layout path context up from a placement to the parent cell.
dfm::descend_hierarchy	Returns an iterator referencing placements in a cell placement.
dfm::descend_net	Returns an iterator referencing nets connected to lower-level placements in a cell placement.
dfm::descend_path_context	Moves a layout path context down into a placement.
dfm::get_data -cell_name, -context_cell_name, -depth, -leaf_cell_name, -original_cell_name, -path_context, -path_name, -placement_name, -xform	Returns the specified type of hierarchy data for the current database object.
dfm::get_path_context	Returns a hierarchical layout path context navigation object.

Iterator processing commands create or access iterators.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-7. Iterator Processing Commands

Command	Description
dfm::apply_transform	Applies a transformation to an iterator.
dfm::area	Returns the area of the current polygon or cell.
dfm::ascend_net	Returns an iterator referencing a net in the parent cell.
dfm::count	Returns the number of polygons in a geometry iterator or the number of placements of the current cell in a cell iterator.
dfm::descend_hierarchy	Returns an iterator referencing placements in a cell placement.
dfm::descend_net	Returns an iterator referencing nets connected to lower-level placements in a cell placement.
dfm::ec	Returns the edge projection length for an error cluster edge pair.
dfm::ew	Returns the shortest distance between edges in an error cluster edge pair.
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_device_data	Returns details from a device-related iterator.
dfm::get_device_geometries	Returns a device seed shape iterator.
dfm::get_device_instances	Returns a device instance iterator.
dfm::get_device_pins	Returns a device pin iterator.
dfm::get_devices	Returns an iterator referencing Device statements in the rule file for devices existing in the DFM database.
dfm::get_flat_geometries	Returns an iterator referencing geometries in a cell's context, including sub-hierarchy. The geometries are referenced as though all sub-cells within a cell are expanded to the level of the specified cell.
dfm::get_flat_placements	Returns an iterator referencing placements in a cell's context, including sub-hierarchy. The placements are referenced as though all sub-cells within a cell are expanded to the level of the specified cell.
dfm::get_geometries	Returns a geometry iterator referencing shapes on a layer.
dfm::get_geometry_count	Returns the total number of geometries on a layer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-7. Iterator Processing Commands (cont.)

Command	Description
dfm::get_lay_vs_netlist_net_name	Returns a list of layout cell net names that differ from the extracted netlist.
dfm::get_layers	Returns a layer iterator.
dfm::get_net_shapes	Returns a geometry iterator referencing geometries on a specified net.
dfm::get_nets	Returns a net iterator.
dfm::get_pins	Returns a pin iterator referencing pins of a cell placement.
dfm::get_placements	Returns a placement iterator referencing placements in a cell.
dfm::get_port_data	Returns information about ports.
dfm::get_ports	Returns a port iterator for a given cell.
dfm::get_xref_cell_data	Returns information from a cell cross-reference iterator generated by qs::get_xref_cells.
dfm::get_xref_cells	Returns an iterator that references cell names in an LVS comparison cross-reference database (XDB).
dfm::inc	Increments the specified iterator or SVRF layer trace object.
dfm::length	Returns the length of the current edge in user units.
dfm::perimeter	Returns the perimeter of the current polygon or cell.
dfm::reset_transform	Resets the transformation on an iterator.
The following commands are documented in the *Calibre PERC User's Manual*.	
dfm::get_ldl_data	Retrieves LDL run data from a dfm::get_ldl_results iterator or from the DFM database directly. This command is used only for databases created by Calibre PERC LDL runs.
dfm::get_ldl_results	Returns an iterator of Calibre PERC LDL CD or P2P results from a DFM database. This command is used only for databases created by Calibre PERC LDL runs.
The following commands are documented in the *Calibre Query Server Manual*.	
qs::device_valid	Indicates whether a layout or source device path is valid. Accepts a cell iterator input.
qs::net_valid	Indicates whether a layout or source net path is valid. Accepts a cell iterator input.
qs::placement_valid	Indicates whether a layout or source placement path is valid. Accepts a cell iterator input.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Layer clustering is an important consideration in certain DFM contexts where multiple layer interactions are of interest.

Table 3-8. Layer Cluster Commands

Command	Description
dfm::create_cluster_initializer	Returns an edge cluster initializer Tcl object.
dfm::get_clusters	Returns an iterator referencing clustered geometries of all layers in a cluster object and in a specified cell context.

Layer management commands manipulate layers and access information about them.

Table 3-9. Layer Management Commands

Command	Description
dfm::add_layer_info	Adds checkname and comment attributes to a layer.
dfm::clear_layer	Deletes all geometries from a layer and makes it an empty layer.
dfm::copy_layer	Copies an input layer to an output layer.
dfm::create_filter	Creates a layer filter object.
dfm::create_layer	Creates a layer from an edge collection.
dfm::delete_layer	Deletes a layer generated by dfm::create_layer.
dfm::get_data -is_detail_layer, -is_nodal_layer, -is_unmerged_layer, -layer_comments, -layer_configuration, -layer_name, -layer_type, -object_type	Returns the specified form of data for the specified option.
dfm::get_layers	Returns a layer iterator.
dfm::list_layers	Lists all the layers in the database and internally generated layer IDs.
dfm::list_original_layers	Returns a list of original layers in the database.
dfm::move_layer	Copies the contents of a layer to an existing target layer, then deletes the copied layer.
dfm::new_layer	Creates a new layer as the result of processing a rule script.
dfm::print_layers	Prints the internally-generated numbers and names of all layers in the database to STDOUT.
dfm::set_new_layer_error_severity	Specifies an error exception severity for dfm::new_layer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-9. Layer Management Commands (cont.)

Command	Description
dfm::split_unmerged	Splits an un-merged DFM Analyze detail layer into two new layers.
dfm::unload_layer	Unloads the specified layer from memory.

Layout data query commands return various details about the layout. Commands returning iterators are listed under Table 3-7.

Table 3-10. Layout Data Query Commands

Command	Description
dfm::area	Returns the area of the current polygon or cell.
dfm::count	Returns the number of polygons in a geometry iterator or the number of placements of the current cell in a cell iterator.
dfm::ec	Returns the edge projection in an error cluster edge pair.
dfm::ew	Returns the shortest distance between edges in an error cluster edge pair.
dfm::get_check_geometry_count	Returns the number of geometries in a check or check and cell combination.
dfm::get_data	Returns the specified type of data for the current database object.
dfm::get_db_extent	Returns the database extent as two vertex coordinate sets of a rectangle.
dfm::get_device_data	Returns details from a device-related iterator.
dfm::get_geometry_count	Returns the total number of geometries on a layer.
dfm::get_gds_file_info	Returns the database precision for the specified GDS file.
dfm::get_lay_vs_netlist_net_name	Returns a list of layout cell net names that differ from the extracted netlist.
dfm::get_layout_name	Returns corresponding layout information for source design objects in an XDB.
dfm::get_placement_count	Returns the total number of placements of a cell.
dfm::get_port_data	Returns information about ports.
dfm::get_top_cell	Returns the name of the top-level cell.
dfm::get_unit_length	Returns the Unit Length statement value from the rule file. By default, the value is 1e-06.
dfm::get_xform_data	Retrieves data from a specified layout transformation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-10. Layout Data Query Commands (cont.)

Command	Description
dfm::get_xref_cell_data	Returns information from a cell cross-reference iterator generated by qs::get_xref_cells.
dfm::length	Returns the length of the current edge in user units.
dfm::list_children	Returns a Tcl list of names of cells placed in the context cell.
dfm::list_layers	Lists the layers and layer IDs in the database. Other information can be included. IDs can be excluded.
dfm::list_original_layers	Returns a list of original layers in the database.
dfm::perimeter	Returns the perimeter of the current polygon.
dfm::print_layers	Prints the internally-generated numbers and names of all layers in the database to STDOUT.
dfm::transform_vertices	Applies a placement transformation to a list of vertices.
The following commands are documented in the *Calibre Query Server Manual*.	
qs::device_table	Prints a table containing detailed information about all layout devices.
qs::device_valid	Indicates whether a layout or source device path is valid.
qs::net_valid	Indicates whether a layout or source net path is valid.
qs::placement_valid	Indicates whether a layout or source placement path is valid.
qs::port_table	Prints a list or table containing detailed information about layout ports.
qs::write_cell_extents	Writes a cell extents file.

Physical databases can be output in GDSII, OASIS, and ASCII formats.

Table 3-11. Layout Database Output Commands

Command	Description
dfm::write_gds	Writes specified layers to a GDSII database file. The various options configure the structure of the database.
dfm::write_oas	Writes specified layers to an OASIS database file. The various options configure the structure of the database.
dfm::write_rdb	Writes specified layers to an ASCII RDB file.

LVS commands enable running LVS on a DFM database design and are related to Connectivity Commands and Netlist Commands. Calibre Connectivity Interface (CCI) commands are used in the context of third-party timing analysis tools.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. LVS and CCI Commands

Command	Description
dfm::get_layout_name	Returns corresponding layout information for source design objects in an XDB.
dfm::get_source_name	Returns corresponding source information for layout design objects in an XDB.
dfm::get_xref_cell_data	Returns information from a cell cross-reference iterator generated by qs::get_xref_cells.
dfm::get_xref_cells	Returns an iterator that references cell names in a cross-reference database (XDB).
dfm::run_compare	Executes an LVS comparison between a source netlist file and the layout represented by the DFM database.
dfm::set_layout_netlist_options	Customizes behavior of the dfm::write_spice_netlist command for writing a layout netlist.
dfm::write_cmp_report	Writes an LVS comparison report to the specified file.
dfm::write_ixf	Writes an instance cross-reference (IXF) file.
dfm::write_lph	Writes a layout placement hierarchy (LPH) file.
dfm::write_nxf	Writes a net cross-reference (NXF) file.
dfm::write_reduction_data	Writes a file containing information about transformation reductions occurring in LVS comparison.
dfm::write_sph	Write a source placement hierarchy (SPH) file.
dfm::write_spice_netlist	Writes a SPICE netlist to the specified file.
dfm::xref_xname	Configures the subcircuit call format of cross-reference file commands.
The following commands are documented in the *Calibre Query Server Manual*.	
qs::device_table	Writes a table containing detailed information about all layout devices.
qs::device_valid	Indicates whether a layout or source device path is valid. Accepts a cell iterator input.
qs::agf_map	Prints the layer map scheme for AGF output.
qs::net_valid	Indicates whether a layout or source net path is valid. Accepts a cell iterator input.
qs::placement_valid	Indicates whether a layout or source placement path is valid. Accepts a cell iterator input.
qs::port_table	Writes a list or table containing detailed information about layout ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. LVS and CCI Commands (cont.)

Command	Description
qs::set_agf_options	Configures AGF output settings for the qs::write_agf command.
qs::write_agf	Writes an annotated geometry file.
qs::write_cell_extents	Writes a cell extents file.

Netlist commands manage SPICE netlist configuration, extraction, and output. They are related to Connectivity Commands and LVS and CCI Commands.

Table 3-13. Netlist Commands

Command	Description
dfm::close_netlist	Closes a netlist object created by dfm::read_netlist.
dfm::get_lay_vs_netlist_net_name	Returns a list of layout net names by cell that differ from the extracted netlist.
dfm::list_layout_netlist_options	Lists options set using dfm::set_layout_netlist_options.
dfm::read_netlist	Returns a SPICE netlist object.
dfm::set_layout_netlist_options	Customizes behavior of the dfm::write_spice_netlist command for writing a layout netlist.
dfm::set_netlist_options	Customizes behavior of the dfm::write_spice_netlist command regarding the output of comment-coded string properties.
dfm::write_spice_netlist	Writes a SPICE netlist into the specified file.

Rule file query commands access rule file settings.

Table 3-14. Rule File Query Commands

Command	Description
dfm::create_svrf_analyzer	Returns a rule file analyzer Tcl object.
dfm::get_data -check_name, -check_text, -layer_name, -operation	Returns the specified type of rule file data for the current database object.
dfm::eval_dfm_func	Evaluates the specified DFM function.
dfm::get_check_text	Returns the check text comments for a rule check.
dfm::get_drc_result_db_magnify	Returns the DRC Magnify Results statement value.
dfm::get_drc_result_db_precision	Returns the DRC Results Database Precision statement value.
dfm::get_layout_magnify	Returns the value of the Layout Magnify specification statement in the rule file that generated the input database.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-14. Rule File Query Commands (cont.)

Command	Description
dfm::get_layout_path	Returns the value of the Layout Path specification statement in the rule file that generated the input database.
dfm::get_layout_path2	Returns the value of the Layout Path2 specification statement in the rule file that generated the input database.
dfm::get_layout_system	Returns the value of the Layout System specification statement in the rule file that generated the input database.
dfm::get_layout_system2	Returns the value of the Layout System2 specification statement in the rule file that generated the input database.
dfm::get_source_path	Returns the value of the Source Path specification statement in the rule file that generated the XDB database.
dfm::get_source_system	Returns the value of the Source System specification statement from the rule file that generated the XDB database.
dfm::get_svrf_data	Retrieves information from an SVRF analyzer or SVRF layer trace object.
dfm::list_checks	Returns a list of rule checks that have layers in the DFM database.
dfm::static_analyze_tvf	Retrieves information about TVF Function block code.

Server administration commands assist in using server commands and ending a session.

Table 3-15. Server Administration Commands

Command	Description
dfm::get_calibre_version	Returns information about the current Calibre version.
dfm::help	Lists all available commands.
dfm::write_cmds	Writes out a history of all the commands that were successfully executed during the run.
exit -force	Quits Calibre YieldServer. An open database with unsaved changes is closed without saving the changes.

This command accesses information in the source design.

Table 3-16. Source Design Query Commands

Command	Description
dfm::get_source_name	Returns corresponding source information for layout design objects in an XDB.
dfm::get_source_path	Returns the value of the Source Path specification statement in the rule file that generated the XDB database.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-16. Source Design Query Commands (cont.)

Command	Description
dfm::get_source_system	Returns the value of the Source System specification statement from the rule file that generated the XDB database.

Timer commands process time data.

Table 3-17. Timer Commands

Command	Description
dfm::create_timer	Returns a timer object.
dfm::get_timer_data	Returns time and memory use information from a timer object.
dfm::reset_timer	Restarts a timer object.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::add_annotation

Adds an annotation to a layer or database revision.

Usage

dfm::add_annotation {**-layer** ***layer_name*** | **-db_revision**} **-annotation** ***annotation_name*** **-value** ***annotation_value***

Arguments

- **-layer *layer_name***

  Argument set that specifies the layer to receive the annotation. May not be specified with **-db_revision**.

- **-db_revision**

  Argument that specifies the annotation is added to the currently open database revision. May not be specified with **-layer**.

- **-annotation *annotation_name***

  A required argument that specifies the name of the annotation to be added. The ***annotation_name*** parameter must be enclosed in double quotes or braces ({}).

- **-value *annotation_value***

  A required argument that specifies the value of the annotation to be added. The ***value*** parameter must be enclosed in double quotes or braces.

Return Values

None.

Description

Adds an annotation to the specified layer or to the current database revision.

An annotation is an identifier that acts like a key-value pair, such as is used in associative arrays. Annotations can be retrieved using various commands in the dfm::list_annotation family for classifying layers or database revisions.

Annotations cannot be added to frozen database revisions.

Examples

This command adds an annotation to the METAL1 layer.

```
# layer METAL1 receives an annotation named LAYER_TYPE with value ROUTE
dfm::add_annotation -layer METAL1 -annotation "LAYER_TYPE" -value "ROUTE"
```

The annotation is saved to the DFM database and can be queried at a later time using one of the annotation listing commands such as dfm::list_annotated_names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Annotations in DFM Databases

Annotation Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::add_geometry

Adds a single shape to an edge collection.

Usage

dfm::add_geometry ***edge_collection*** **-vertices** "***vertices_list***" [-property {{*name value*} …}] [-ordered_property "*values_list*"]

Arguments

- ***edge_collection***

  A required argument that specifies a Tcl object to an edge collection created with dfm::create_ec.

- **-vertices** "***vertices_list***"

  A required argument that specifies the vertices of the shape as a Tcl list. Vertices are specified as sequential X and Y coordinate pairs. The values are in user units.

- -property {{*name value*} …}

  An optional argument that specifies property *name* and *value* pairs for annotating the generated shape. The name-value pairs are in a Tcl list of lists.

- -ordered_property "*values_list*"

  An optional argument set that specifies a Tcl list of property values for annotating the generated shape. The values are assigned to properties in the edge collection object in the order the properties were defined.

Return Values

None.

Description

Adds a single shape specified with a list of vertices to an edge collection and optionally annotates the shape with DFM properties.

Properties can be defined in the edge collection object when it is created. Property values may be assigned by name using the -property option. Property values may be assigned by the order in which the properties were originally defined in the dfm::create_ec -property specification by using -ordered_property.

Examples

This script demonstrates the -property and -ordered_property options. The edge_layer_prop and edge_layer_ordered_prop layers can be examined in Calibre RVE or using appropriate YieldServer commands to see how the properties are stored.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# create two identical edge collection objects with three properties
set ec_1 [dfm::create_ec -type edge -property {{d_prop double} \
    {l_prop long} {s_prop string}}]
set ec_2 [dfm::create_ec -type edge -property {{d_prop double} \
    {l_prop long} {s_prop string}}]

# create a new layer of m1 edges spaced <= 0.4 um apart
dfm::new_layer -svrf {
   m1_edge = INT [m1] <= 0.4
}

# geometry iterator for m1_edge edges
set g_iter [dfm::get_geometries m1_edge]

# property values of types double precision, long integer, and string
set d 1.8
set l 10
set s "string prop"

# iterate over the m1_edge edges. add the edges to the respective edge
# collections with properties using the -property and -ordered_property
# formats. increment the l property value for each iteration.
while {$g_iter ne ""} {
   set v [dfm::get_data $g_iter -vertices]
   dfm::add_geometry $ec_1 -vertices $v -property [list [list d_prop $d]\
        [list l_prop $l] [list s_prop $s]]
# add property values in the order d_prop, l_prop, and s_prop were
# defined originally
   dfm::add_geometry $ec_2 -vertices $v -ordered_property [list $d $l $s]
   dfm::inc g_iter
   incr l
}

# create layers from the edge collections.
dfm::create_layer $ec_1 -name edge_layer_prop
dfm::create_layer $ec_2 -name edge_layer_ordered_prop

# save the database revision as the default and close the db
dfm::save_rev
dfm::set_default_rev [dfm::get_current_rev]
dfm::close_db
```

Related Topics

Edge Collection Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::add_geometry_property

Adds a property to a DFM Analyze _detail layer geometry identified by polygon number.

Usage

dfm::add_geometry_property -layer *layer_name* -polygon *polygon_number*
-property *property_name* {{**-double** | **-long**} ***property_value***} [-cell *cell_name*]

Arguments

- **-layer *layer_name***

  A required argument that specifies a DFM Analyze _detail layer.

- **-polygon *polygon_number***

  A required argument that specifies the polygon number for receiving the property. Polygon numbers are assigned internally by the tool and can be accessed through the dfm::get_data -curr_geometry_id option.

- **-property *property_name***

  A required argument that specifies the name of the property.

- **-double**

  An argument that specifies ***property_value*** is assigned as a double-precision floating-point number. May not be specified with **-long**.

- **-long**

  An argument that specifies ***property_value*** is assigned as a long format integer. May not be specified with **-double**.

- ***property_value***

  A required argument that specifies a numeric property value.

- -cell *cell_name*

  An optional argument that specifies the context cell name containing the geometry that receives the property. The context is the cell itself, not any sub-hierarchy. If this argument is not specified, the property is assigned to a polygon in the top cell.

Return Values

None.

Examples

```
# adds the property M1_COUNT with a value of 4.0 to geometry 1 on
# layer m1_count detail layer
dfm::add_geometry_property -layer m1_count::<1>_detail -polygon 1 \
-property M1_COUNT -double 4.0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

dfm::get_data

DFM Property Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::add_layer_info

Adds checkname and comment attributes to a layer.

Usage

dfm::add_layer_info -layer *layer_name* {{**-check *check_name***} | {**-comments** "***comments***"} | {**-check *check_name* -comments** "***comments***"}}

Arguments

- **-layer *layer_name***

  A required argument that specifies the layer to receive attributes.

- **-check *check_name***

  An argument set that specifies a check name to associate with the ***layer_name***. If **-check** is not specified, then **-comments** must be.

- **-comments** "***comments***"

  An argument set that specifies one or more comment lines to associate with the ***layer_name***. The comments value must be enclosed in double quotes (" "). If **-comments** is not specified, then **-check** must be.

Return Values

None.

Description

Adds check name and comment attributes to a layer. The specified ***layer_name*** must be present in the DFM database. If both a ***check_name*** and ***comments*** are provided, the comments are associated with the combination of the layer and the check. The -layer_comments option for dfm::get_data is used to retrieve comment and check name attributes.

Examples

```
# adds check name and comments to m1_edge
dfm::add_layer_info -layer m1_edge -check check_m1e -comments "m1 edges"
```

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::add_property

Adds a property to geometries on a layer or referenced by an iterator.

Usage

dfm::add_property {***layer_name*** | ***geometry_iterator***} ***“property_name” property_value type*** [-force_configure]

Arguments

- ***layer_name***

  Argument that specifies a layer name. Properties are added to geometries on the layer. May not be specified with ***geometry_iterator***.

- ***geometry_iterator***

  Argument that specifies the geometry iterator to which the property is added. May not be specified with ***layer_name***.

- ***“property_name”***

  A required argument specifying a property name. This string must be enclosed in quotes.

- ***property_value***

  A required argument that specifies the property value. This can be any Tcl expression. The expression is evaluated once for each geometry in the layer. Inside the expression, the current geometry can be referenced by a special substitution variable called “geometry”, which behaves like a geometry iterator that references one geometry.

  The expression can contain any Calibre YieldServer measurement commands, such as dfm::area and dfm::length.

- ***type***

  A required argument defining the property type. One of these options must be used:

  -string — Store the value as a string. Can be used for numeric and list values.

  -long — Store the value as a long integer.

  -double — Store the value as a double-precision floating-point number.

  In Tcl, all values are strings. In the database, numeric values, list values, and string values are stored differently.

- -force_configure

  An optional parameter that fully merges a ***layer_name*** layer and configures it for adding the property. This option is not used with a ***geometry_iterator***.

Return Values

None.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Description

When ***layer_name*** is used, for each geometry on a layer, the command adds a property to the geometry with the specified name, value, and type. Note that if you change property values on a saved layer, you cannot re-save the layer. You must first make a copy of the saved layer, then change property values on the copied layer.

Properties can only be added to an unmerged original layer by using the -force_configure option.

When a ***geometry_iterator*** used, for each geometry in the iterator the command adds a property to the geometry with the specified name, value, and type.

Properties can be deleted using dfm::delete_property.

Examples

Example 1

This example shows the use of the special “geometry” argument that acts as an iterator. An AREA property is assigned to all polygons on layer metal1c. The -force_configure option is needed because the input layer is unmerged.

```
dfm::copy_layer -input_layer metal1 -output_layer metal1c
dfm::add_property metal1c "AREA" "dfm::area geometry" -double \
  -force_configure
...
puts "=== PROPS metal1c: [dfm::list_properties metal1c -min_max]"
```

Example 2

This example uses a geometry iterator as the input for adding properties. The output is to an ASCII results database.

```
# copy database layer so properties can be attached
  dfm::copy -input_layer metal1 -output_layer metal1_prop
# create a new string property on the entire layer
  dfm::add_property metal1_prop {new_prop} {new_string} -string \
    -force_configure
# create a geometry iterator
  set geo_iter [dfm::get_geometries metal1_prop]
# add the value returned by dfm::area for each geometry as a property
  while {$geo_iter ne ""} {
    dfm::add_property $geo_iter "area_prop" [dfm::area $geo_iter] -double
    dfm::inc geo_iter
  }
 dfm::write_rdb -layer metal1_prop -file metal1_prop.rdb
```

For each cell in the DFM database, a rule check of the name metal1_prop_*N*, where *N* is a positive integer, is instantiated in *metal1_prop.rdb*. Output to the results database comes from cells having metal1 at their primary levels. For cells that only have metal1 in their sub-hierarchies, the generated rule checks are empty. Results are presented at the top level when using Calibre RVE and a layout viewer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

DFM Property Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::apply_transform

Applies a transformation to an iterator.

Usage

dfm::apply_transform *iterator* -xform *xform*

Arguments

- ***iterator***

  A required argument supplying the geometry or placement iterator.

- **-xform *xform***

  A required argument that specifies a transformation to apply. The ***xform*** is an object generated by dfm::get_data -xform.

Return Values

None.

Description

Applies a transformation to an iterator. Subsequent calls to obtain vertices or extents return the coordinates in the transformed context.

This command is useful for accessing geometries of a cell in particular placement's context.

Examples

```
# initialize the geometry iterator to M1 geometries in MYCELL
set g_iter [dfm::get_geometries M1 -cell MYCELL]

# get placements of MYCELL in TOPCELL
set placement_iter [dfm::get_placements "TOPCELL" -of_cell "MYCELL"]

# apply transform of first MYCELL placement in TOPCELL to the geometry
# iterator
set placement_xform [dfm::get_data $placement_iter -xform]
dfm::apply_transform $g_iter -xform $placement_xform

# report transformed shape coordinates
while {$g_iter ne ""} {
    puts "---Shape ID: [dfm::get_data $g_iter -curr_geometry_id]"
    puts "---Transform Coordinates: [dfm::get_data $g_iter -vertices]"
    dfm::inc g_iter
}
```

Related Topics

dfm::reset_transform

dfm::transform_vertices

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::area

Returns the area of the current polygon or current cell's extent.

Usage

dfm::area *iterator*

Arguments

- ***iterator***

  A required argument specifying a polygon iterator, such as from dfm::get_device_geometries, dfm::get_flat_geometries, dfm::get_geometries, or dfm::get_net_shapes, or a cell iterator from dfm::get_cells. This command gives an error when passed an iterator other than for polygons (type 1 layer) or cells.

Return Values

Floating-point number.

Examples

See Example 1 under dfm::get_geometries.

Related Topics

Layout Data Query Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::ascend_hierarchy

Also used in the Query Server Tcl shell.

Returns an iterator referencing placements in the parent cell of the current placement.

Usage

dfm::ascend_hierarchy *placement_iterator*

Arguments

- ***placement_iterator***

  A required argument specifying a placement iterator created with dfm::descend_hierarchy.

Return Values

Iterator.

Examples

Given this code:

```
# Get placements in top level
set topItr [dfm::get_placements [dfm::get_top_cell]]
while {$topItr ne ""} {
# Get placement names
  set cellName [dfm::get_data $topItr -cell_name];
# If placement corresponds to CellA, descend the hierarchy
  if {$cellName eq "CellA"} {
    set descendItr [dfm::descend_hierarchy $topItr];
    puts "Context cell: $cellName";
# Report descendants and placement names in CellA
    while {$descendItr ne ""} {
      puts "Descendant:   [dfm::get_data $descendItr -cell_name] \
        ([dfm::get_data $descendItr -path_name])";
# Ascend to CellA's containing context
      set ascendItr [dfm::ascend_hierarchy $descendItr];
# Report cells and placements in the containing context of CellA
      while {$ascendItr ne ""} {
        puts "Ascendant:    [dfm::get_data $ascendItr -cell_name] \
          ([dfm::get_data $ascendItr -path_name])";
        dfm::inc ascendItr;
      }
      dfm::inc descendItr;
    }
  }
  dfm::inc topItr;
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

and this design hierarchy:

```
TOPCELL
  CellA (X0)
  CellA (X1)
  CellB (X2)

CellA
  nand (X0)
```

the script outputs this:

```
Context cell: CellA
Descendant:   nand  (X0/X0)
Ascendant:    CellA  (X0)
Ascendant:    CellA  (X1)
Ascendant:    CellB  (X2)
Context cell: CellA
Descendant:   nand  (X1/X0)
Ascendant:    CellA  (X1)
Ascendant:    CellB  (X2)
```

CellA is placed twice in the top-level cell, so it is visited twice as the Context cell. Each time the descendant hierarchy of CellA is visited, the nand cell is reported as placement X0. The second time CellA is visited as the Context cell, it is in the context of placement X1 at the top level rather than placement X0. So CellA is reported only once in the ascended context in the second iteration.

Related Topics

Hierarchy Traversal Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::ascend_net

Returns an iterator referencing a net in the parent cell of the current placement.

Usage

dfm::ascend_net ***placement_iterator*** **-net** ***node_ID*** [-original]

Arguments

- ***placement_iterator***

  A required argument specifying a placement iterator created with dfm::get_placements.

- **-net** ***node_ID***

  Required argument set that specifies a valid node ID in the placement cell. The ***node_ID*** is a non-negative integer. Node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.

- -original

  Optional keyword that instructs the tool to ascend into the parent cell only if the connected net exists in the original data.

Return Values

Iterator.

Description

Creates an iterator referencing the net in the parent cell that the specified ***node_ID*** is connected to. If ***node_ID*** is not connected to anything in the super-hierarchy, the iterator is empty. This command is useful for browsing connectivity.

Examples

```
# get placements in CELLA
set placement_iter [dfm::get_placements CELLA]
# get net in parent of current placement connected to net 392 of
# current placement
set net_iter [dfm::ascend_net $placement_iter -net 392]
# get the name, if any, of the parent's net
set net_name [dfm::get_data $net_iter -net_name]
```

Related Topics

Hierarchy Traversal Commands

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::ascend_path_context

Also used in the Query Server Tcl shell.

Moves a layout path context up from a placement to the parent cell.

Usage

dfm::ascend_path_context ***path_context*** [-to_top]

Arguments

- ***path_context***

  A required argument that specifies a layout path context object from dfm::get_path_context.

- -to_top

  An optional argument that when specified, ascends the layout path context object to the highest level possible.

Return Values

None.

Examples

See the "Examples" section under dfm::get_path_context.

Related Topics

Hierarchy Traversal Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::chmod

Changes the permissions of a DFM database.

Usage

dfm::chmod ***permissions dfmdb_path***

Arguments

- ***permissions***

  Required argument that specifies the permissions mode. This argument accepts either numeric (octal) or alphabetic permissions values, similar to the Unix chmod command.

- ***dfmdb_path***

  Required argument that specifies the path to the DFM database to which permissions are applied.

Return Values

None.

Examples

```
# change permissions using octal values
dfm::chmod 777 dfmdb

# change permissions using letters
dfm::chmod u+wr dfmdb
```

Related Topics

Database Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::clear_layer

Deletes all geometries from a layer and makes it an empty layer.

Usage

dfm::clear_layer ***layer_name***

Arguments

- ***layer_name***

  Required argument that specifies the layer to be emptied.

Return Values

None.

Examples

```
# makes layer1 empty
dfm::clear_layer layer1
```

Related Topics

dfm::delete_layer

dfm::unload_layer

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::close_db

Closes the current database.

Usage

dfm::close_db [-force]

Arguments

- -force

  Optional keyword that closes the database and discards any unsaved changes. If this option is not specified, dfm::close_db returns an error if there are any unsaved changes in your database.

Return Values

None.

Description

Closes the current database without exiting Calibre YieldServer. If you do not specify the -force option, this command returns an error if there are any unsaved changes in the database. If you specify -force, the database is closed and all unsaved changes are discarded.

Examples

This shows a typical command sequence for closing a database revision.

```
> dfm::open_db dfmdb
Opening database "dfmdb", revision "master"
> dfm::create_rev new

Creating revision new

Saving revision new
...
# perform some changes
...
> dfm::save_rev
> dfm::set_default_rev [dfm::get_current_rev]
> dfm::close_db
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::close_netlist

Also used in the Query Server Tcl shell.

Closes a netlist object created by dfm::read_netlist. This command should be used after a netlist object is no longer needed and before a new one is created.

Usage

dfm::close_netlist

Arguments

None.

Return Values

None.

Examples

```
# read the source netlist from the rule file; use YS default xforms
set source_object [dfm::read_netlist -source -rules rules]
# write the transformed netlist
dfm::write_spice_netlist src_xform.sp -netlist_handle $source_object
# close the netlist object
dfm::close_netlist
```

Related Topics

Netlist Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::copy_db

Copies the source DFM database to a destination DFM database.

Usage

dfm::copy_db ***source_database destination_database***

Arguments

- ***source_database***

  Path to the source DFM database directory.

- ***destination_database***

  Path to the destination DFM database directory.

Return Values

None.

Description

Copies the specified ***source_database*** to the specified ***destination_database***. The ***source_database*** must be closed before running this command. You cannot open the source or destination database until the copy is completed.

This command produces an error if the ***destination_database*** already exists, or a trailing slash is present in the ***destination_database*** pathname.

Examples

```
# make copy of database
dfm::copy_db dfmdb1 dfmdb2
```

Related Topics

Database Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::copy_layer

Copies an input layer to an output layer.

Usage

dfm::copy_layer -input_layer *layer1* -output_layer *layer2* [-annotations]

Arguments

- **-input_layer *layer1***

  Required argument that specifies the input layer.

- **-output_layer *layer2***

  Required argument that specifies the output (or destination) layer.

- -annotations

  Optional argument that copies all annotations (as generated with dfm::add_annotation) present on the input layer to the output layer.

Return Values

None.

Description

Copies the contents of ***layer1*** to ***layer2***, including any node numbers and properties. This command causes an error if ***layer2*** already exists in the database.

The. dfm::move_layer command is related to dfm::copy_layer and moves the contents of one layer to another existing layer. The layer whose contents are moved is then deleted.

Examples

```
# copy via1 layer
dfm::copy_layer -input_layer via1 -output_layer via1c
```

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::copy_svdb_to_dfmdb

Converts a Mask SVDB Directory database to a DFM database.

Usage

dfm::copy_svdb_to_dfmdb -svdb *svdb* -dfmdb *dfmdb*

Arguments

- **-svdb *svdb***

  A required argument set that specifies a pathname of a source Mask SVDB Directory.

- **-dfmdb *dfmdb***

  A required argument set that specifies a pathname of an output DFM database.

Return Values

None.

Description

Transforms an SVDB into a DFMDB. The target ***dfmdb*** can then be loaded in the current session.

This command can be useful in certain LVS or Calibre PERC flows where post-processing of PHDB layer data is desired. Requires a Calibre PERC Advanced license.

Examples

```
# convert svdb and open it
> dfm::copy_svdb_to_dfmdb -svdb svdb -dfmdb dfmdb
> dfm::open_db dfmdb
...
```

Related Topics

Database Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::count

Returns the number of polygons in a geometry iterator or the number of placements of the current cell in a cell iterator.

Usage

dfm::count *iterator*

Arguments

- ***iterator***

  A required argument specifying a geometry iterator, such as from dfm::get_device_geometries, dfm::get_flat_geometries, dfm::get_geometries, or dfm::get_net_shapes, or a cell iterator from dfm::get_cells. This command gives an error when passed an iterator other than for polygons or cells.

Return Values

Integer.

Examples

```
# get the count of M1 shapes
set m1_iter [dfm::get_geometries M1]
set m1_count [dfm::count $m1_iter]
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::create_cluster_initializer

Returns an edge cluster initializer Tcl object.

Usage

dfm::create_cluster_initializer -layer *list* [-same_cell] [-emulate_trans]

Arguments

- **-layer *list***

  A required argument that specifies a Tcl list of layers for clustering. The list should have at least two layers. The first layer in the list is the primary layer, which determines the promotion and clustering of geometries. All other layers are secondary layers whose polygons are clustered with primary layer polygons. All layers in the list must be polygon layers.

- -same_cell

  An optional argument that specifies to iterate over geometries and perform clustering on a per-cell basis (no clustering across cell boundaries). When this argument is specified, the command returns any geometries that remain in the cell after promotion. Geometries from pseudo cells (ICV_ cells) are promoted.

- -emulate_trans

  An optional argument that specifies clustering for DFM Transition outputs. You must specify at least three layers in the layer ***list*** if you use this argument.

Return Values

Tcl object.

Description

Returns a Tcl object that enables processing of interacting shapes across layers in the ***list***. Overlapping polygons from the specified layers are added to layer clusters. The object handles all of the promotion necessary during cluster iteration.

The dfm::get_clusters command creates layer cluster iterators that are used for processing layer clusters after initialization is performed using dfm::create_cluster_initializer.

You should always unset the cluster initializer after using it to reduce memory usage.

Examples

See dfm::get_clusters.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::create_ec

Returns an edge collection Tcl object. Edge collections are insensitive to design hierarchy.

Usage

dfm::create_ec [**-type** {**polygon** | **edge** | **error_cluster**}] [-property {{*name type*} …}]

Arguments

- **-type** {**polygon** | **edge** | **error_cluster**}

  Optional argument that specifies the layer type. One of these arguments can be specified:

  polygon — Specifies the edges are from a polygon layer. This is the default.

  edge — Specifies the edges are from an edge layer.

  error_cluster — Specifies the edges are from an error layer.

- -property {{*name type*} …}

  Optional argument that specifies one or more properties as *name type* pairs. The properties are associated with the edge collection object and permit assignment of property values to the geometries generated from the edge collection object. Valid *type* values are these:

  string — character string format.

  long — long integer format.

  double — double-precision floating-point numeric format.

  For example:

  ```
  {{prop1 long} {prop2 double}}
  ```

Return Values

Tcl object.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

```
// create an HDB edge collection of type edge with two properties
// d_prop and l_prop
set EC [dfm::create_ec -type polygon \
        -property {{d_prop double} {l_prop long}}]

# property values of type double precision and long integer
set d 1.8
set l 10

# add geometries to the collection
dfm::add_geometry $EC -vertices [list 219000 0 143000 108251] \
  -property [list [list d_prop $d] [list l_prop $l]]
dfm::add_geometry $EC -vertices [list 219000 0 143000 108251] \
  -property [list [list d_prop $d] [list l_prop $l]]
dfm::add_geometry $EC -vertices [list 219000 0 157250 21249] \
  -property [list [list d_prop $d] [list l_prop $l]]

# create a layer from the edge collection
dfm::create_layer $EC -name new_layer
```

Related Topics

Edge Collection Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::create_filter

Used only in the Query Server Tcl shell.

Creates a filter object. If no argument is specified, the filter object does nothing to constrain a command that references the object. Otherwise, the filter object acts to constrain a referencing command to select only elements from the filter object. This command is not qualified for use in Calibre YieldServer.

Usage

dfm::create_filter [-box_layers *layer_list*] [-device_ids *id_list* | -device_names *name_list*] [-layers *layer_list*] [-magnify {*factor* | -auto}] [-reflect_x] [-rotate *angle*] [-translate *x_offset* *y_offset* [-db_units]] [-maximum_vertex_count]

Arguments

- -box_layers *layer_list*

  An optional argument set that specifies a Tcl list of layer names to be written to LVS Box cells. This option has a similar intent as the AGF FILTER BOX LAYERS command in the Calibre Connectivity Interface. If this option is not used, box cells are written with the same layers that other cells have.

  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 AGF output.

- -device_ids *id_list*

  An optional argument set that specifies a Tcl list of device IDs. The list values correspond to the integer indices of Device statements in the rule file, which are given by $D properties in the extracted netlist. The indices begin at 0, which corresponds to the first device statement in the rules. May not be specified with -device_names. This option has a similar intent as the FILTER DEVICES command in the Query Server.

- -device_names *name_list*

  An optional argument set that specifies a Tcl list of Device element names. May not be specified with -device_ids. This option has a similar intent as the FILTER DEVICENAMES command in the Query Server.

- -layers *layer_list*

  An optional argument set that specifies a Tcl list of layer names. This option has a similar intent as the FILTER LAYERS command in the Query Server. However, unlike FILTER LAYERS, dfm::create_filter does not include all layers by default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -magnify {*factor* | -auto}

  An optional argument set that specifies to multiply output coordinates by a value, as follows:

  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.

  -auto — Option 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. Layout Magnify AUTO must be specified in the rules when this option is used.

  This argument set is analogous to the MAGNIFY RESULTS command in the standard Query Server.

- -reflect_x

  An optional argument that specifies to reflect output y-coordinate values across the x axis.

  This argument set is analogous to the REFLECTX RESULTS command in the standard Query Server.

- -rotate *angle*

  An optional argument set that specifies to rotate output coordinates about the database origin by a specified angle. The positive direction is counter-clockwise. The allowed *angle* values in degrees are from the set {-270, -180, -90, 0, 90, 180, 270}. The default is 0.

  This argument set is analogous to the ROTATE RESULTS command in the standard Query Server.

- -translate *x_offset y_offset*

  An optional argument set that specifies to displace output coordinates by the specified offset values. The *x_offset* is added to x coordinates, and the *y_offset* is added to y coordinates. Both are user units by default. The defaults are 0.

  This argument set is analogous to the TRANSLATE RESULTS command in the standard Query Server.

- -db_units

  An optional argument used with -translate that specifies database units are used. When this option is present, the *x_offset* and *y_offset* must be integers.

- -maximum_vertex_count *number*

  An optional argument set that specifies the maximum number of vertices an output polygon can have before partitioning occurs. The *number* is an integer greater than or equal to 4. The default is 4096.

  This argument is analogous to the MAXIMUM VERTEX COUNT command in the standard Query Server.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

Filter object.

Description

Filter objects created by this command are referenced in -filter option specifications of commands that support them. Filter objects constrain the outputs of the commands referencing them. Each filter object acts independently of any other filter objects.

If -device_ids or -device_names is used, then the returned filter affects the qs::get_devices_at_location, qs::status -filter, and qs::write_agf commands.

If -maximum_vertex_count is used, then the returned filter affects the dfm::get_data, qs::status -filter, and qs::write_agf commands.

The transformation options are applied to output coordinates in this order: reflection, rotation, magnification, translation. Transformation filters affect these commands:

dfm::get_db_extent	dfm::get_port_data
dfm::get_data	dfm::get_ports
dfm::get_device_data	
qs::get_devices_at_location	qs::write_agf
qs::get_nets_at_location	qs::write_cell_extents
qs::port_table	qs::write_separated_properties
qs::status	qs::get_placements_at_location

Examples

This example creates an iterator of references to ports in cell BLOCK on layers m8 and m9.

```
set port_itr [dfm::get_ports BLOCK -filter \
  [dfm::create_filter -layers {m8 m9}]]
```

Related Topics

Connectivity Commands

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::create_layer

Creates a layer from an edge collection.

Usage

dfm::create_layer ***edge_collection*** **-name** ***layer_name*** [-overwritable] [-is_merged]

Arguments

- ***edge_collection***

  A required argument that specifies an edge collection Tcl object created with dfm::create_ec.

- **-name** ***layer_name***

  A required argument that specifies the name of the layer to create. If the layer is already present in the database and is not writable, the command produces an error.

- -overwritable

  An optional keyword that sets the ***layer_name*** to be a layer that can be overwritten.

- -is_merged

  An optional keyword that sets the ***layer_name*** as a merged layer. (Certain functions and SVRF operations can only use merged input layers.)

Return Values

None.

Description

Creates a layer from an edge collection. The output layer is added to the top cell only. Generated layers can be deleted using dfm::delete_layer.

An unmerged layer created using this command can be used only in functions and SVRF operations that accept unmerged input layers. If you encounter errors regarding unmerged layer inputs, specify the -is_merged option.

Examples

This example generates layer data and saves it to new layers.

```
# create edge clusters with edges that can have certain properties
set ec_1 [dfm::create_ec -type edge \
  -property {{d_prop double} {l_prop long} {s_prop string}}]
set ec_2 [dfm::create_ec -type edge \
  -property {{d_prop double} {l_prop long} {s_prop string}}]

# get M1 edges where width is <= 0.4
dfm::new_layer -svrf {
   m1_edge = INT [m1] <= 0.4
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# get the edges on m1_edge
set g_iter [dfm::get_geometries m1_edge]

# set some property values
set d 1.8
set l 10
set s "string prop"

# for the geometries in g_iter, get their vertices, add edges on ec_1 and
# ec_2 with the property values
while {$g_iter ne ""} {
   set v [dfm::get_data $g_iter -vertices]
   dfm::add_geometry $ec_1 -vertices $v \
     -property [list [list d_prop $d] [list l_prop $l] [list s_prop $s]]
   dfm::add_geometry $ec_2 -vertices $v -ordered_property [list $d $l $s]
   dfm::inc g_iter
   incr l
}

dfm::create_layer $ec_1 -name edge_layer_prop
dfm::create_layer $ec_2 -name edge_layer_ordered_prop
```

Related Topics

Edge Collection Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::create_rev

Also used in the Query Server Tcl shell.

Creates a new revision of the current database.

Usage

dfm::create_rev [*revision_name*]

Arguments

- *revision_name*

  An optional argument that specifies the name of the revision to create. The *revision_name* must be unique, and it must not be simply numeric.

Return Values

TCL_OK or TCL_ERROR.

Description

This command creates a new database revision. The new revision is created as a branch of the currently open revision (the parent). Multiple revisions can have the same parent; such "siblings" have independent existence. New revisions start with contents identical to their parent. This is accomplished by hard links so that identical files are not duplicated on disk. Any modifications made in memory before the call to dfm::create_rev are also saved. (To avoid this, close the current database first.)

If there are multiple revisions derived directly from any revision, the number assigned to the child of a revision is not necessarily "parent + 1". For example:

```
    0
   / \
  1   2
 /
3
```

You cannot use dfm::create_rev when the current revision is an unfrozen revision.

Examples

This example shows a typical sequence of database administrative commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> dfm::open_db dfmdb
Opening database "dfmdb", revision "master"
dfm::create_rev new

Creating revision new

Saving revision new
...
# perform some changes
...
> dfm::save_rev
> dfm::set_default_rev [dfm::get_current_rev]
> dfm::close_db
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::create_svrf_analyzer

Returns a rule file analyzer object. The object is used in a subsequent dfm::get_svrf_data command.

Usage

dfm::create_svrf_analyzer [*rule_file*]

Arguments

- *rule_file*

  An optional argument that specifies the name of the input rule file. If this argument is not present, the rule file that created the DFM database is used.

Return Values

Tcl object.

Examples

This shows how to write the layers in a DFM database to the transcript.

```
# store the analyzer object for further analysis
set svrf_obj [dfm::create_svrf_analyzer rules]
# write a Tcl list of design layers to the transcript
puts "LAYERS: [dfm::get_svrf_data $svrf_obj -list_layers]"
```

Also see the dfm::run_compare Examples section.

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::create_timer

Also used in the Query Server Tcl shell.

Returns a timer object.

Usage

dfm::create_timer

Arguments

None.

Return Values

Tcl object.

Description

Creates a timer object to track time and memory usage, which you return using the dfm::get_timer_data command.

The timer objects can be used as a performance monitor for any Tcl application written for Calibre YieldServer.

Examples

Once a timer object is created, it can be referenced as many times as desired to output time or memory data.

```
# initialize timer object
set timer [dfm::create_timer]
...
# timer report to transcript
puts "LOG DATA: [dfm::get_timer_data $timer -print]"
...
puts "LOG DATA: [dfm::get_timer_data $timer -print]"
```

Related Topics

Timer Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::delete_annotation

Deletes annotations from a layer or database revision.

Usage

dfm::delete_annotation {**-layer** ***layer_name*** | **-db_revision**} **-annotation** ***annotation_name***
[-value *annotation_value*]

Arguments

- **-layer** ***layer_name***

  Argument that specifies the annotation layer to process. May not be specified with **-db_revision**.

- **-db_revision**

  Argument that specifies a currently open database revision to process. May not be specified with **-layer**.

- **-annotation** ***annotation_name***

  A required argument that specifies the name of the annotation to be removed. The ***annotation_name*** parameter must be enclosed in quotes or braces ({}).

- -value *annotation_value*

  An optional argument that specifies the value of the annotation to be removed. When specified, only annotations that match the ***annotation_name*** and *annotation_value* pair are removed. The *annotation_value* parameter must be enclosed in quotes or braces.

Return Values

None.

Description

Deletes an annotation from the specified layer or the currently open database revision. Annotations cannot be deleted from a frozen database revision. Annotations are added with dfm::add_annotation.

Examples

```
# delete all LAYER_TYPE annotations from METAL1
dfm::delete_annotation -layer METAL1 -annotation "LAYER_TYPE"
```

Related Topics

Annotations in DFM Databases

Annotation Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::delete_layer

Deletes a layer generated by dfm::create_layer.

Usage

dfm::delete_layer *layer_name*

Arguments

- ***layer_name***

  A required argument specifying the name of the layer to delete. This must be a layer generated by dfm::create_layer. You cannot delete a layer that was saved to a DFM database through a calibre -dfm run.

  > **Caution**
  > Deleting a layer invalidates any layer or geometry iterator that contains references to that layer.

Return Values

None.

Description

Deletes a layer. You cannot delete a layer that is already saved on disk unless it is part of an unfrozen revision.

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::delete_property

Deletes properties from geometries on a layer.

Usage

dfm::delete_property {***layer_name*** | ***iterator***} {***property_name*** | **-all**}

Arguments

- ***layer_name***

  An argument specifying a layer name. May not be specified with ***iterator***.

- ***iterator***

  An argument specifying a layer iterator. May not be specified with ***layer_name***.

- ***property_name***

  An argument specifying the name of a property to delete. May not be specified with **-all**.

- **-all**

  An argument specifying to delete all properties. May not be specified with ***property_name***.

Return Values

None.

Description

Deletes the specified property from all objects on the ***layer_name*** or the current layer of an ***iterator***. Note that if you delete property values on a saved layer, you cannot re-save the layer. You must make a copy of the saved layer first, then delete the property values on the copied layer.

Properties can be added to shapes in the database using dfm::add_geometry_property and dfm::add_property.

Examples

```
# copy database layer metal1_prop to manipulate properties on the copy
dfm::copy_layer -input_layer metal1_prop -output_layer metal1_prop_c
# delete property "AREA" and output layer to results database
dfm::delete_property metal1_prop_c "AREA"
dfm::write_rdb -layer metal1_prop_c -file metal1_prop.rdb
```

Related Topics

DFM Property Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::delete_rev

Deletes the specified database revision. This command cannot delete a revision if it is the default or current revision.

Usage

dfm::delete_rev {***revision_name*** | ***revision_number***}

Arguments

- ***revision_name***

  At argument that specifies a revision name. Not specified with ***revision_number***.

- ***revision_number***

  At argument that specifies a revision number. Not specified with ***revision_name***.

Return Values

None.

Examples

```
# deletes database revision 2
dfm::delete_rev 2
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::descend_hierarchy

Also used in the Query Server Tcl shell.

Returns an iterator referencing placements in a cell placement.

Usage

dfm::descend_hierarchy *placement_iterator* [-of_cell *cell_name*]

Arguments

- ***placement_iterator***

  A required argument specifying a placement iterator created with dfm::get_placements.

- -of_cell *cell_name*

  An optional argument set that restricts the generated iterator to referencing placements in the specified *cell_name*.

Return Values

Iterator.

Examples

```
# Iterate over all placements in TOP cell and descend into the
# placement if the placed cell is "oa21"
set top_cell [dfm::get_top_cell]
set piter [dfm::get_placements $top_cell]
# Returns iterator to placements in top cell
while {$piter ne ""} {
  set cell_name [dfm::get_data $piter -cell_name]
  if {$cell_name eq "oa21"} {
# Descend into the placement of oa21 and look at all
# placements of via cells within that hierarchy
    set descended_placement_iter [dfm::descend_hierarchy $piter \
      -of_cell "via"]
# do some processing
    ...
  }
  dfm::inc piter
}
```

Related Topics

Hierarchy Traversal Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::descend_net

Returns an iterator referencing nets connected to lower-level placements in a cell placement.

Usage

dfm::descend_net {***placement_iterator*** | ***pin_iterator***}

Arguments

- ***placement_iterator***

  A placement iterator created with a context net using the -net option of the dfm::get_placements command. May not be specified with ***pin_iterator***.

- ***pin_iterator***

  A cell instance pin iterator created with dfm::get_pins. May not be specified with ***placement_iterator***.

Return Values

Iterator.

Examples

```
# Keep descending into placements while tracing the net 24 from TOP cell.
set top_cell [dfm::get_top_cell]
set placements_connected_to_net [dfm::get_placements $top_cell -net 24]
set net_iter [dfm::descend_net $placements_connected_to_net]
set no_stop 0
if {$net_iter ne ""} {
  set no_stop 1
}

while {$no_stop} {
  set current_cell [dfm::get_data $net_iter -cell_name]
  set current_net_number [dfm::get_data $net_iter -node]
  set placements_connected_to_net [dfm::get_placements $current_cell \
        -net $current_net_number]

  if {$placements_connected_to_net ne ""} {
    set net_iter [dfm::descend_net $placements_connected_to_net]
    if {$net_iter ne ""} {
      set descended_net_number [dfm::get_data $net_iter -node]
      set descended_cell_name [dfm::get_data $net_iter -cell_name]
      puts "Net $current_net_number in cell $current_cell is \
       connected to net $descended_net_number in cell \
       $descended_cell_name"
      } else {
        set no_stop 0
      }
   } else {
     set no_stop 0
   }
 }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Hierarchy Traversal Commands

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::descend_path_context

Also used in the Query Server Tcl shell.

Moves a layout path context down into a placement given by a placement iterator.

Usage

dfm::descend_path_context ***path_context placement_iterator***

Arguments

- ***path_context***

  A required argument that specifies a layout path context object generated by dfm::get_path_context.

- ***placement_iterator***

  A required argument that specifies a placement iterator created with dfm::get_placements. The context cell of the placement iterator should match the leaf cell of the layout path context.

Return Values

None.

Examples

See Example 2 under dfm::get_path_context.

Related Topics

Hierarchy Traversal Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::disconnect

Deletes the current connectivity model.

Usage

dfm::disconnect

Arguments

None.

Return Values

None.

Description

Removes the current connectivity model in the database, if the model exists. The effects of using this command are the following:

- The hierarchical database connectivity model is deleted.
- All layers that have existing connectivity are downgraded to a configuration (CFG layer statistic in the transcript) of 1 or 0 as appropriate.
- All outstanding iterators for connectivity layers are invalidated.
- Any net properties are discarded from layers.

Examples

```
# disconnects the current connectivity model
dfm::disconnect

# initialize connectivity on M1 and M2
dfm::new_layer -svrf {
    CONNECT M1 M2 BY VIA
}
```

Related Topics

Connectivity Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::ec

Returns the edge projection in an error cluster edge pair.

Usage

dfm::ec ***iterator***

Arguments

- ***iterator***

  A required argument specifying an edge cluster iterator created by dfm::get_geometries with an error layer argument.

  This command results in an error when passed an iterator for objects other than edge clusters (type 3 layer).

Return Values

Floating-point number in user units.

Description

Returns the *edge concurrency* (EC) for an edge cluster pair, which is the length of the projection of one edge onto the other. Projection is as defined by the PROJECTING keyword for the ENClosure, EXTernal, or INTernal dimensional check operations.

Examples

```
# error_layer contains error edge clusters
set geo_iter [dfm::get_geometries error_layer]
# get the EC value for each cluster and process
while { $geo_iter != ""} {
  set EC [dfm::ec $geo_iter]
# test EC
   ...
  dfm::inc geo_iter
}
```

Related Topics

Layout Data Query Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::eval_dfm_func

Evaluates the specified DFM function.

Usage

dfm::eval_dfm_func *function_name* [-double] ***args_list***

Arguments

- ***function_name***

  A required argument supplying the name of the DFM Function to be evaluated. The function must be a TABLE or MATRIX function.

- -double

  An optional argument that specifies numeric output values are double-precision floating-point. This is the default.

- ***args_list***

  A required argument that specifies a Tcl list of arguments for the DFM Function. The order of the list should match the order specified in the function definition.

Return Values

String.

Description

Evaluates a DFM Function and returns the result (if any).

Examples

Assume this DFM Function:

```
DFM FUNCTION [ fib (NUMBER x)
TABLE {1 1 2 1 3 2 4 3 5 5 6 8 7 13 8 21 9 34 10 55} ]
```

then this proc:

```
proc function {} {
  set val 7.5
  puts "fib [format "%2.1f" $val]: [dfm::eval_dfm_func fib "$val"]"
}
```

writes this to the transcript:

```
fib 7.5: 17.0
```

The value 17.0 is the linearly interpolated value between 13 and 21.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::ew

Returns the shortest distance between edges in an error cluster edge pair.

Usage

dfm::ew ***iterator***

Arguments

- ***iterator***

  A required argument specifying an edge cluster iterator created by dfm::get_geometries with an error layer argument.

  This command results in an error when passed an iterator for objects other than edge clusters (type 3 layer).

Return Values

Floating-point number in user units.

Description

Returns the *edge separation* (EW) for an edge cluster pair.

Examples

```
# error_layer contains edge clusters
set geo_iter [dfm::get_geometries error_layer]
# get EW for each cluster
while { $geo_iter != ""} {
    set EW [dfm::ew $geo_iter]
# test EW
    dfm::inc geo_iter
}
```

Related Topics

Layout Data Query Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::freeze_rev

Causes a database revision to be unalterable.

Usage

dfm::freeze_rev

Arguments

None.

Return Values

None.

Description

Finalizes the current database revision and saves it. Once dfm::freeze_rev is called, the revision can no longer be modified, but it can be used as the parent for new revisions.

Examples

```
# freezes (finalizes) the current revision and saves it.
dfm::freeze_rev
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_calibre_version

Returns information about the current Calibre version.

Usage

dfm::get_calibre_version {**-rls_date** | **-version**}

Arguments

- **-rls_date**

  Argument that specifies to return the time stamp of the release.

- **-version**

  Argument that specifies to return the version ID of the release.

Return Values

String.

Examples

This shows commands executed in the interactive mode:

```
> dfm::get_calibre_version -version
v2015.4_4.0004
> dfm::get_calibre_version -rls_date
Tue Oct 6 18:46:50 PDT 2015
```

Related Topics

Server Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_cells

Also used in the Query Server Tcl shell.

Returns a cell iterator.

Usage

dfm::get_cells [*cell_iterator* | *cell_name* | *netlist_object*] [-reverse]

Arguments

The following three arguments are mutually exclusive:

- *cell_iterator*

  An optional argument specifying a cell iterator created with dfm::get_cells. The current cell referenced by the iterator is used to generate the returned iterator.

- *cell_name*

  An optional argument specifying a cell name used to generate the returned iterator.

- *netlist_object*

  An optional argument specifying a *netlist_object* generated by dfm::read_netlist. The netlist object is used to generate the returned iterator.

- -reverse

  Causes cells to be traversed from bottom of the hierarchy to the top. By default, this is performed top to bottom.

Return Values

Iterator.

Description

Returns a cell iterator. When none of the optional arguments is used, all cells in the database from the top level down are processed when generating the returned iterator. The -reverse option changes this order from bottom to top. When any of the other three arguments is used, the returned iterator references cells based on the specified arguments, including any sub-hierarchy.

Examples

Example 1

This creates a cell iterator from data in a DFM database. The iterator is stepped through using a while block to output design cell names and extents in the run transcript.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set cells [dfm::get_cells]
set i 1
while {$cells ne ""} {
  set name [dfm::get_data $cells -cell_name]
  set extent [dfm::get_data $cells -extent]
  puts "${i}. $name ($extent)"
  dfm::inc cells
  incr i
}
```

Example 2

This creates a subcircuit definition iterator from a SPICE netlist object.

```
set nl [dfm::read_netlist -perc svdb -top_cell_name PLL]
set subckt_iterator [dfm::get_cells $nl]
```

Related Topics

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_check_geometry_count

Returns the number of geometries in a check or check and cell combination.

Usage

dfm::get_check_geometry_count *check_name* [-cell *cell_name* [-nopseudo]]

Arguments

- ***check_name***

  A required argument that specifies the name of a rule check.

- -cell *cell_name*

  An optional argument that specifies a cell name in which to perform the count. The context is the cell itself, not any sub-hierarchy. By default, the count occurs in all cells.

- -nopseudo

  An optional argument that returns the number of geometries after flattening pseudo cells. This argument cannot be used if *cell_name* is a pseudo cell.

Return Values

Integer.

Examples

```
# store the number of geometries for check1
set check1_count [dfm::get_check_geometry_count check1]
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_check_text

Returns the check text comments for a rule check.

Usage

dfm::get_check_text -check *check_name*

Arguments

- ***check_name***

  A required argument that specifies the name of a rule check.

Return Values

String.

Description

Returns the check text for the specified ***check_name***. Check text is preceded by an @ symbol in a typical rule check.

DFM RDB can specify its own check name that differs from the usual rule check name, and the DFM RDB name can be used for ***check_name***. DFM RDB can also specify its own check text comment using the COMMENT keyword, which is different from the usual check text comment.

Examples

Example 1

```
checkname {
@ svrf check text
DFM RDB M1 m1.rdb
}

# returns "svrf check text" in the YS shell
> dfm::get_check_text -check checkname
```

Example 2

```
checkname {
@ svrf check text
DFM RDB M1 m1.rdb CHECKNAME "rdbcheck" COMMENT "rdb check text"
}

# does not return anything because DFM RDB defines its own check name
dfm::get_check_text -check checkname

# returns "rdb check_text"
dfm::get_check_text -check rdbcheck
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_clusters

Returns an iterator referencing clustered geometries of all layers in a cluster object and in a specified cell context.

Usage

dfm::get_clusters -cluster_handle *object* -cell *cell_name*

Arguments

- **-cluster_handle *object***

  A required argument set that specifies the name of a object generated by a dfm::create_cluster_initializer command.

- **-cell *cell_name***

  A required argument set that specifies the cell context for returned geometries. The context is the cell itself, not any sub-hierarchy. You can determine the cell context of an iterator by using the following command:

  ```
  dfm::get_data $iterator -cell_name
  ```

Return Values

Iterator.

Examples

This example shows how to iterate over DFM Transition output.

```
# Get the cluster object
set cluster_init [dfm::create_cluster_initializer \
  -layer {M1new M2new VIAnew} -emulate_trans]

# Initialize the cell iterator
set cell_iter [dfm::get_cells]

# Loop over all cells from TOP to BOTTOM
while {$cell_iter ne ""} {
  set cell [dfm::get_data $cell_iter -cell_name]
# Initialize the cluster iterator for each cell
  set cluster_iter [dfm::get_clusters -cell $cell \
    -cluster_handle $cluster_init]
    while {$cluster_iter ne ""} {
      set vertices_list [dfm::get_data $cluster_iter -vertices]
# process vertices
            ...
      dfm::inc cluster_iter
    }
    dfm::inc cell_iter
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_connect_warnings

Returns a circuit extraction warning iterator.

Usage

dfm::get_connect_warnings [-cell {*cell_name* | *cell_iterator*}]
[-open | -short | -unattached | -virtual_connect]

Arguments

- -cell {*cell_name* | *cell_iterator*}

  A optional argument set that specifies the cell to query. When this option is not used, the command iterates over all cells. When -cell is specified, the context is the primary level of the cell, not any sub-hierarchy. One of these arguments must be used:

  cell_name — Name of a cell.

  cell_iterator — An iterator created by dfm::get_cells. The current cell referenced by the iterator is used.

- -open

  An option that specifies open circuit warnings are referenced. The associated type when accessed is OPEN. This option is used by default.

- -short

  An option that specifies short circuit warnings are referenced. The associated type when accessed is SHORT. This option is used by default.

- -unattached

  An option that specifies unattached label warnings are referenced. The associated type when accessed in UNATTACHED. This option is used by default.

- -virtual_connect

  An option that specifies virtual connection warnings are referenced. The associated type when accessed in VIRTUAL CONNECT. This option is used by default.

Return Values

Circuit extraction warning iterator.

Description

Returns an iterator that references circuit extraction warnings. Circuit extraction must be run before this command is used. The relevant dfm::get_data options for retrieving data from the iterator are these:

-cell_name	Returns the names of cells having warnings.
-warning_info	Returns data specific to the type of warning.
-warning_type	Returns just the warning types.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

This example reports connectivity warning types by cell.

```
# get all circuit extraction warnings
set w_itr [dfm::get_connect_warnings]
# transcript any bad cells and the type of problem
while {$w_itr ne ""} {
  puts "BAD CONNECTIVITY IN CELL: [dfm::get_data $w_itr -cell_name]"
  puts "             DISCREPANCY: [dfm::get_data $w_itr -warning_type]"
  dfm::inc w_itr
}
```

Related Topics

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_current_rev

Returns the revision number or name of the current database revision.

Usage

dfm::get_current_rev [-name]

Arguments

- -name

  An optional argument that returns the name of the revision (instead of the number).

Return Values

Integer or string.

Examples

This example shows a typical sequence of database administration commands.

```
> dfm::open_db dfmdb
Opening database "dfmdb", revision "master"
> dfm::create_rev new

Creating revision new

Saving revision new
...
# perform some changes
...
> dfm::save_rev
> dfm::set_default_rev [dfm::get_current_rev]
> dfm::close_db
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_data

Also used in the Query Server Tcl shell.

Returns the specified type of data for the current database object. Data from port iterators is also accessible from dfm::get_port_data, which has a more comprehensive set of options for ports.

Usage

dfm::get_data *object* [-filter *filter_object*] ***data_type***

Arguments

- ***object***

  A required argument that specifies an iterator or a layout path context object. The command reports the specified data type attribute for the current object referenced by the iterator or the path context object.

- -filter *filter_object*

  An optional argument set specifying a dfm::create_filter object containing transformation information. When used, this argument set must precede the ***data_type*** argument. The ***data_type*** arguments that observe the transformations in the filter object include -extent, -port_info, and -vertices.

- ***data_type***

  A required argument specifying the data to return. Refer to the following table for a complete list of allowed option names and their associated attributes. If the type of data requested is not relevant for the current referenced object, the command results in an error.

Table 3-18. Options and Data Attributes

Option	Applies to	Return Values
-cell_name	geometry iterators, cell iterators, net iterators, placement iterators, pin iterators, port iterators, circuit extraction warning iterators	For a geometry, placement, net, and pin iterator, returns the name of the containing cell. For a cell or circuit extraction warning iterator, returns the name of the cell.
-cell_property *property_name* *layer_name*	geometry iterators, cell iterators, placement iterators	Cell property value, such as those created by DFM Property BY CELL. For cell and placement iterators, you must specify *layer_name*. See -object_type for a relevant query option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-18. Options and Data Attributes (cont.)

Option	Applies to	Return Values
-check_name	geometry iterators, layer iterators	List of the checks the layer or geometry belongs to.
-check_text	geometry iterators, layer iterators	Check text comment for the check that the layer or geometry belongs to.
-context_cell_name	layout path context object[1]	Context cell name of the current layout path context.
-curr_geometry_id	geometry iterators	Index number of the current geometry.
-depth	placement iterators, layout path context object[1]	Path depth from the context cell.
-extent	cell iterators, placement iterators, layout path context object[1]	Bounding box coordinates as a list of vertices: x_0, y_0, x_1, y_1 For layout path context objects, this option returns the extent of the leaf placement. All extents are reported with respect to the context cell, with one exception: if the path context is ascended to the context cell, this option returns the extent of the context cell in the global coordinate space. Cell or placement extent values are adjusted by any geometric transformations in a *filter_object*, if specified.
-geometry_property *property_name*	geometry iterators	Geometry property value, such as those created by DFM Property without the BY CELL or BY NET keywords. See -object_type for a relevant query option.
-has_property *property_name*	geometry iterators	Returns 1 if the object has the named property; 0 otherwise.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-18. Options and Data Attributes (cont.)

Option	Applies to	Return Values
-is_box	cell iterators	Returns the type of LVS Box cell. LVS_BOX — regular box cell. LVS_BOX_BLACK — black box cell. LVS_BOX_GRAY — gray box cell. empty string — non-box cell.
-is_detail_layer	layer iterators	Returns 1 if the layer is a _detail layer created by DFM Analyze; 0 otherwise.
-is_frozen	layer iterators, geometry iterators	Returns 1 if the layer belongs to a frozen revision; 0 otherwise.
-is_nodal_layer	layer iterators	Returns 1 if the layer has connectivity; 0 otherwise.
-is_pseudo_cell	geometry iterators, cell iterators, or cell names	Returns 1 if current cell is a pseudo cell; 0 otherwise.
-is_unmerged_layer	layer iterators	Returns 1 if the layer contains un-merged shapes; 0 otherwise.
-is_vector_property *property_name*	geometry iterators	Returns 1 if the property is a vector property; 0 otherwise.
-layer_comments [-check *checkname*]	layer iterators	Returns layer comments; if -check is specified, returns the layer comments that are specific to a particular check.
-layer_configuration	geometry iterators, layer iterators	Returns the layer configuration (CFG) type: 0 — neither polygon nor nodal 1 — polygon 2 — nodal 3 — polygon and nodal
-layer_name	geometry iterators, layer iterators	For a geometry iterator, returns the name of the containing layer. For a layer iterator, returns the name of the layer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-18. Options and Data Attributes (cont.)

Option	Applies to	Return Values
-layer_type	geometry iterators, layer iterators	Returns the following integers based upon the layer type: 1 — polygon 2 — edge 3 — error cluster
-leaf_cell_name	layout path context object[1]	Cell name of the leaf placement.
-net	net iterators, layout path context object for a net[1], port iterators	Net name as a path.
-net_is_epin	layout path context object for a net[1]	Returns 1 if the specified net object connects to a higher level in the hierarchy; 0 otherwise.
-net_is_original	layout path context object for a net[1]	Returns 1 if the specified net is identified in the cell during connectivity extraction and not created as part of promotion; 0 otherwise.
-net_name	geometry iterators, net iterators, port iterators	For a geometry iterator, returns the name of the net that the current geometry belongs to (if it belongs to a net, otherwise ""). For a net iterator, returns the name of the net.
-net_property *property_name*	geometry iterators	Net property value, such as those created by DFM Property BY NET. See -object_type for a relevant query option.
-netlist_net_name	net iterators	Names of nets as reported in the SPICE netlist written for the layout from the dfm::write_spice_netlist command.
-node	geometry iterators, net iterators, layout path context object for a net[1], pin iterators, port iterators	Node number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-18. Options and Data Attributes (cont.)

Option	Applies to	Return Values
-object_type	any iterator type or path context object	Type of the current object, that is, CELL, GEOMETRY, LAYER, NET, and so on.
-operation	layer iterators	The operation type that generated the layer.
-original_cell_name	cell iterators	The original cell name of the current cell. The original cell name is different from the Calibre cell name only for faux bin cells.
-path_context	hierarchical placement iterators	Layout path context object.[1] The object references the hierarchical placement path of the input iterator. The context cell for the object is the same as was used to create the placement iterator. This option is an adjunct to dfm::get_path_context.
-path_name	placement iterators, layout path context object[1]	Path name to the current layout path context.
-placement_name	placement iterators, pin iterators	Layout name for the placement.
-polygon_number	geometry iterators	Positive integer corresponding to a polygon in an edge collection. If this does not apply, the value is 0.
-port_dir	port iterators	Port's direction of current flow such as in, out, in/out, or unknown.
-port_info	port iterators	Port name, net ID, net name, coordinates in cell context, and layer name to which the port is attached. Coordinate values are adjusted by any geometric transformations in a *filter_object*, if specified.
-port_name	port iterators	Port name.
-revision_of_origin	layer iterators	Revision in which the layer was created. Returns an error for unsaved layers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-18. Options and Data Attributes (cont.)

Option	Applies to	Return Values
-use_value	port iterators	Value of the "use" construct in a LEF/DEF database.
-vertices [*top_cell_name*]	geometry iterators	List of vertices in cell coordinates[2]. If *top_cell_name* is specified, the list is returned in top-cell coordinates. The values are adjusted by any geometric transformations in a *filter_object*, if specified.
-warning_info	circuit extraction warning iterator created by dfm::get_connect_warnings	List of lists containing information specific to the type of warning referenced by the iterator. See Table 3-19.
-warning_type	circuit extraction warning iterator created by dfm::get_connect_warnings	String corresponding to the type of warning the iterator references. Possibly strings include: OPEN, SHORT, UNATTACHED, or VIRTUAL CONNECT.
-xform	cell iterators, geometry iterators, placement iterators, layout path context object[1]	Transformation object for the placement or cell pointed to by the iterator or path context object. The dfm::get_xform_data command returns transformation details from the object.

1. Layout path context objects are created with the dfm::get_path_context and dfm::get_data -path_context commands.
2. The list contains pairs of x,y values for each vertex, which are listed in sequence going around the polygon.

Return Values

Varies according to type of data returned. When objects appear in more than one location, more than one value can be returned in a list. Data is returned in database units when appropriate.

For -warning_info, these are the returned lists:

Table 3-19. -warning_info Return Values

Warning type	Returned list
OPEN	{*cell retained_node* {*retained_net* {*x y layer*}} {*rejected_node* {*x y layer*}} …}
SHORT	{*cell retained_node* {*retained_net_name* {*x y layer*}} {*rejected_net_name* {*x y layer*} …} …}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-19. -warning_info Return Values (cont.)

Warning type	Returned list
UNATTACHED	{*cell* {*label* {*x y layer*}}
VIRTUAL CONNECT	{*cell net_name node* {*x y layer*} {*x y layer*}}

Examples

Example 1

```
# layout path context object
set path [dfm::get_path_context [dfm::get_top_cell] -path X1/X2]
# returns "context: TOPCELL"
puts "context: [dfm::get_data $path -context_cell_name]"
# returns instance path
puts "[dfm::get_data $path -path_name]"
# returns the cell name of the leaf placement X2
puts "[dfm::get_data $path -leaf_cell_name]"
# returns rotation and reflection with respect to context cell
set xobj "[dfm::get_data $path -xform]"
puts "rotation:   [dfm::get_xform_data $xobj -rotation]"
puts "reflection: [dfm::get_xform_data $xobj -reflection]"
# returns placement extent in coordinate space of context cell
puts "extent: [dfm::get_data $path -extent]"
# returns depth: 2
puts "depth: [dfm::get_data $path -depth]"
```

Example 2

```
# net path context object
set pc [dfm::get_path_context [dfm::get_top_cell] -net X18/1]
set c [dfm::get_data $pc -leaf_cell_name]
# both statements return a true if the net in the cell is identified
# during connectivity extraction
puts "Net 1 in cell $c is original: [dfm::get_data $pc -net_is_original]"
# both statements return a true if the specified net is connected
# to a higher level in the design hierarchy
puts "Net 1 in cell $c is epin: [dfm::get_data $pc -net_is_epin]"
# returns the full net name including the path
puts "[dfm::get_data $pc -net]"
```

Example 3

```
# loop over nets and get netlisted names
set net_iter [dfm::get_nets]
while {$net_iter ne ""} {
  set netlist_net_name [dfm::get_data $net_iter -netlist_net_name]
  ...
  dfm::inc net_itr
}
```

Related Topics

Layout Data Query Commands

dfm::get_port_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_db_creation_info

Returns information about the calibre -dfm run that created the DFM database.

Usage

dfm::get_db_creation_info ***info_type***

Arguments

- ***info_type***

  A required argument that specifies the type of information to return from the run transcript when the database was generated. All times are reported to the nearest second. This argument can only be specified once from among these choices:

  -cpu_time — Returns the CPU TIME entry. This is the CPU clock time required to create the database.

  -remote_cpu_time — Returns the remote CPU TIME entry. This is the aggregate CPU clock time of remote processors required to create the database.

  -wall_time — Returns the REAL TIME entry. This is the amount of “stopwatch” time taken to create the database.

  -elapsed_time — Returns the ELAPSED TIME entry. This is the amount of “stopwatch” time elapsed since the “Starting time” in the header of the transcript.

  -lvheap — Returns the LVHEAP usage.

  -malloc — Returns the MALLOC usage.

  -hostname — Returns the host name.

  -calibre_build — Returns the build of Calibre that was used to create the database.

  -summary_header — Returns the header from the DFM Summary Report.

  The “Executive Process” section of the *Calibre Verification User’s Manual* discusses these statistics in detail.

Return Values

String.

Examples

```
> dfm::get_db_creation_info -wall_time
REAL TIME = 0
> dfm::get_db_creation_info -elapsed_time
ELAPSED TIME = 1
```

Related Topics

Database Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_db_extent

Also used in the Query Server Tcl shell. Corresponding Query Server command: EXTENT.

Returns the database extent as a Tcl list of lower-left and upper-right vertices as X and Y coordinates in database units.

Usage

dfm::get_db_extent [-filter *filter_object*]

Arguments

- -filter *filter_object*

  An optional argument set that specifies a dfm::create_filter transformation filter object. Geometric transformation settings of the filter are applied to the output.

Return Values

Tcl list.

Examples

```
> dfm::get_db_extent
Loading hierarchy and connectivity
-23000 2750 45500 79750
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_db_name

Returns the name of the current database.

Usage

dfm::get_db_name

Arguments

None.

Return Values

String.

Examples

```
> dfm::get_db_name
dfmdb
```

Related Topics

Database Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_db_precision

Returns the current database precision as the ratio of database units to user units.

Usage

dfm::get_db_precision

Arguments

None.

Return Values

Double-precision floating-point number.

Examples

```
> dfm::get_db_precision
1000.0
```

Related Topics

Database Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_default_rev

Gets the default revision for the currently open database.

Usage

dfm::get_default_rev

Arguments

None.

Return Values

Integer.

Examples

```
> dfm::get_default_rev
0
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_device_data

Also used in the Query Server Tcl shell. Corresponding Query Server command: DEVICE INFO.

Returns details from a device-related iterator. Device recognition must occur before this command is used.

Usage

dfm::get_device_data {***device_iterator*** | ***device_template_id*** | ***device_instance_iterator*** | ***seed_shape_iterator*** | ***device_pin_iterator*** | ***path_context***} [-filter *filter_object*] ***data_type***

Arguments

One of the following must be specified followed by a ***data_type*** option:

- ***device_iterator***

  An argument that specifies an iterator created by dfm::get_devices.

- ***device_template_id***

  An argument that specifies the ID number of a Device statement in the rule file. Active Device statements are internally assigned ID numbers indexed from 0 in the order they appear in the rule file. In an extracted SPICE netlist, the $D property gives the index of the Device template that was used to extract the device. The ***device_template_id*** value can be obtained by a dfm::get_device_data command using the -device_id option.

- ***device_instance_iterator***

  An argument that specifies an iterator created by dfm::get_device_instances.

- ***seed_shape_iterator***

  An argument that specifies an iterator created by dfm::get_device_geometries.

- ***device_pin_iterator***

  An argument that specifies an iterator created by dfm::get_device_pins.

- ***path_context***

  An argument that specifies a path context object. The ***path_context*** must be created for a device instance. For example:

  ```
  set pc [dfm::get_path_context TOP -device X1/M1]
  ```

- -filter *filter_object*

  An optional argument set specifying a dfm::create_filter object containing transformation information. When used, this argument set must precede the ***data_type*** argument. The -location ***data_type*** argument observes the transformations in the filter object.

- ***data_type***

  A required argument that specifies the type of data to return. Refer to the following table for a complete list of allowed types.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-20. Allowed Options for dfm::get_device_data

Option	Description
-aux_layers	Returns the auxiliary layers for the current device or instance.
-cell_name	Returns the name of current context cell. Does not apply to ***device_iterator*** or ***device_template_id***.
-device_id	Returns the unique device template id from the current device or instance. This corresponds to the $D properties in extracted SPICE netlists.
-device_info	Returns the device ID number, a list of nets attached to pins, a list of property values, and the seed shape layer name. Does not apply to ***device_iterator*** or ***device_template_id***. Corresponding Query Server command: DEVICE INFO.
-device_inst_id	Returns the unique device instance id. Applies only to ***device_instance_iterator*** or ***seed_shape_iterator***.
-device_name	Returns the Device element name. Corresponding Query Server command: DEVICE NAMES.
-device_type	Returns the classification corresponding to the current device or instance. This is not the SPICE element name or Device statement type, but an internal classification.
-instance_name	Returns the device instance name. Does not apply to ***device_iterator*** or ***device_template_id***.
-list_properties	Returns a Tcl list of device property names for the current device or instance.
-location	Returns the location of the device. Returns the location of a pin if ***device_pin_iterator*** is used and the DFM Database or Mask SVDB Directory PINLOC option is also specified. Does not apply to ***device_iterator*** or ***device_template_id***. The coordinates are adjusted by any geometric transformations in a *filter_object*, if specified.
-model_name	Returns the device subtype (or model name) corresponding to the current device or instance.
-net	Returns the node number of the net that is connected to the pin. Applies only to ***device_pins_iterator***.
-netlist_element	Returns the NETLIST ELEMENT argument name from the Device statement.
-netlist_model	Returns the NETLIST MODEL argument name from the Device statement.
-object_type	Returns the type of iterator.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-20. Allowed Options for dfm::get_device_data (cont.)

Option	Description
-original_cell_name	Returns the original name of current context cell. Does not apply to ***device_iterator*** or ***device_template_id***.
-pins	Returns the device or instance pin names. Does not apply to ***device_pin_iterator***.
-pin_info	Returns the device pin info (pin name, pin layer, pin group, node number). The format of the output is a Tcl list of all pins as {*pin1 pin2 pin3* …}. Each *pin* is itself a Tcl list that is formatted as follows: {PIN_NAME *name_of_pin*} {PIN_LAYER *name_of_pin_layer*} {PIN_GROUP *pin_group_number*} {NET *node_number*} where PIN_NAME, PIN_LAYER, PIN_GROUP, and NET are tags. Corresponding Query Server command: DEVICE PINS.
-pin_layers	Returns the device pin layers for the current device or instance.
-pin_name	Returns the name of the current pin. Applies only to ***device_pin_iterator***. Corresponding Query Server command: DEVICE PINS.
-property *property_name*	Returns the device property value for the named property. Does not apply to ***device_iterator*** or ***device_template_id***.
-property_info	Returns the device property name and value as a Tcl list, with the following result format: {{*property_name value*} {*property_name value*} …} For a device with array properties, the result includes the array as follows: {*property_name* {*value1 value2 value3* …}} If there are no properties, an empty string is returned. Does not apply to ***device_iterator*** or ***device_template_id***.
-seed_layer	Returns the name of the seed layer corresponding to the current device or instance.
-seed_shape_layer	Returns the YieldServer system name of the layer containing the extracted seed shapes. Returns an empty string if no device seed shapes are extracted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

Device-specific information.

Examples

Example 1

This example shows how to use dfm::get_device_data to identify a device.

```
set my_device_name "MN"
set my_seed_layer_name "pgate"
set my_pin_list [list [list "g" "POLY"] [list "s" "pdiff"] \
  [list "d" "pdiff"] [list "b" "NWELL"]]

set dev_temp_iter [dfm::get_devices]

while { $dev_temp_iter ne "" } {
  set pin_info [dfm::get_device_data $dev_temp_iter -pin_info]
  set seed_layer_name [dfm::get_device_data $dev_temp_iter -seed_layer]
  set device_name [dfm::get_device_data $dev_temp_iter -device_name]
  if { $device_name eq $my_device_name && $seed_layer_name eq \
    $my_seed_layer_name } {
    foreach pin $pin_info {
      set pin_name [lindex [lindex $pin 0] 1]
      set pin_layer [lindex [lindex $pin 1] 1]
      lappend pin_list [list $pin_name $pin_layer]
    }
    if { $pin_list eq $my_pin_list } {
      puts "Device template found."
      break
    }
  }
  dfm::inc dev_temp_iter
}
if { $dev_temp_iter eq "" } {
  puts "Unable to find device template."
}
```

Example 2

This example shows how you can create a path context to a specific device, then query the device for various information.

```
set path_context [dfm::get_path_context TOP -device X837/X38/M4]

set dname [dfm::get_device_data $path_context -device_name]
set dtype [dfm::get_device_data $path_context -device_type]
set dsubtype [dfm::get_device_data $path_context -model_name]
set dstats [dfm::get_device_data $path_context -device_info]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

This example shows how to get device instance properties:

```
set di [dfm::get_device_instances -cell somecell]
while { $di != "" } {
  puts [ dfm::get_device_data $di -property_info ]
  dfm::inc di
}
```

The output might appear as follows:

```
{w 1.5e-05} {l 1.25e-06}
{w 1.5e-05} {l 1.25e-06}
{w 1.5e-05} {l 1.25e-06}
{w 2e-05} {l 1.75e-06}
{w 2e-05} {l 1.75e-06}
{w 2e-05} {l 1.75e-06}
```

Related Topics

Iterator Processing Commands

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_device_geometries

Returns a device seed shape iterator. Device recognition must occur before this command is used.

Usage

dfm::get_device_geometries {***device_instance_iterator*** | ***device_iterator*** | ***device_template_id*** | ***layer_iterator*** | ***layer_name***}

Arguments

One of the following must be specified:

- ***device_instance_iterator***

  An argument that specifies an iterator created by dfm::get_device_instances.

- ***device_iterator***

  An argument that specifies an iterator created by dfm::get_devices.

- ***device_template_id***

  An argument that specifies the ID number of a Device statement in the rule file. Active Device statements are internally assigned ID numbers indexed from 0 in the order they appear in the rule file. In an extracted SPICE netlist, the $D property gives the index of the Device template that was used to extract the device. The ***device_template_id*** value can be obtained by a dfm::get_device_data command using the -device_id option.

- ***layer_iterator***

  An argument that specifies an iterator created by dfm::get_layers.

- ***layer_name***

  An argument that specifies a layer name.

Return Values

Iterator.

Examples

Iterate over all seed shapes of a given device template:

```
set dti [dfm::get_devices]
while {$dti ne ""} {
  set shape_iter [dfm::get_device_geometries $dti]
  while {$shape_itr ne ""} {
# process seed shapes
    ...
    dfm::inc shape_iter
  }
  dfm::inc dti
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_device_instances

Also used in the Query Server Tcl shell. Corresponding Query Server command: DEVICE NAMES.

Returns a device instance iterator. Device recognition must occur before this command is used.

Usage

dfm::get_device_instances [-cell {*cell_iterator* | *cell_name*}] [-flat]
{[-device *device_template_iterator*] | {[-of_name *device_template_name*]
[-of_type *device_type_name*] [-of_model *model_name*]}}

Arguments

- -cell {*cell_iterator* | *cell_name*}

  An optional argument set that specifies a cell context for returning device instances. The context is the cell itself, not any sub-hierarchy. By default, the command returns all instances in the design.

  cell_iterator — A name of a cell iterator created with dfm::get_cells. The command applies to the current cell pointed to by the iterator. May not be specified with *cell_name*.

  cell_name — A name of a cell. May not be specified with *cell_iterator*.

- -flat

  An optional argument that specifies to return a flat instance iterator rather than a hierarchical one. If used with -cell, the iterator is flat with respect to the context cell, not the primary cell.

- -device *device_template_iterator*

  An optional argument that specifies an iterator created with dfm::get_devices.

- -of_name *device_template_name*

  An optional argument that specifies a device template (SPICE) name. This name is returned by the dfm::get_device_data -device_name option.

- -of_type *device_type_name*

  An optional argument that specifies a device type. This type is returned by the dfm::get_device_data -device_type option. This is not the SPICE element name or Device statement type, but an internal classification.

- -of_model *model_name*

  An optional argument that specifies a device subtype (or model name). This name is returned by the dfm::get_device_data -model_name option.

Return Values

Iterator.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

Get all device instances with device template name “MP” and subtype “pmos”:

```
set pDev [dfm::get_device_instances -of_name MP -of_model pmos]
```

Example 2

This code returns flat instance paths with respect to cell AAA.

```
set dev_itr [dfm::get_device_instances -flat -cell AAA]
while {$dev_itr ne ""} {
   puts [dfm::get_device_data $dev_itr -instance_name]
   dfm::inc dev_itr
}
```

Output could look like this:

```
R3
R3
X0/M0
X0/M1
X0/X2/R0
```

Related Topics

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_device_pins

Also used in the Query Server Tcl shell. Corresponding Query Server command: DEVICE PINS.

Returns a device pin iterator. Device recognition must occur before this command is used.

Usage

dfm::get_device_pins {***device_instance_iterator*** | ***path_context***}
[-net {*node_ID* | *net_iterator*}] [-pin_name *pin_name*] [-pin_layer *pin_layer*]

Arguments

- ***device_instance_iterator***

  An argument that specifies the name of a device instance iterator created by dfm::get_device_instances. Either this option or ***path_context*** must be specified.

- ***path_context***

  An argument that specifies a path context navigation object created by dfm::get_path_context.

- -net {*node_ID* | *net_iterator*}

  An optional argument that causes the command to reference only pins connected to the specified net. One of these arguments is used:

  node_ID — A non-negative integer corresponding to the net ID assigned by the Calibre circuit extractor. Node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.

  net_iterator — A net iterator created by dfm::get_nets. The current net from the iterator is used.

- -pin_name *pin_name*

  An optional argument that limits the returned pin to the one having the *pin_name*.

- -pin_layer *pin_layer*

  An optional argument that limits the returned pins to those on the *pin_layer*.

Return Values

Iterator.

Examples

This example reports pin names, node IDs, and locations for device instances in cellA.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# iterate over device instance pins in cellA and report pin info
set inst_iter [dfm::get_device_instances -cell cellA]
set pins [dfm::get_device_pins $inst_iter]

while {$pins ne ""} {
  set p_name [dfm::get_device_data $pins -pin_name]
  set node [dfm::get_device_data $pins -net]
  set loc [dfm::get_device_data $pins -location]
  set dev_inst_name [dfm::get_device_data $pins -instance_name]
  puts "--- Pin $p_name of device $dev_inst_name is connected to \
    node $node at location $loc"
  dfm::inc pins
}
```

Related Topics

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_devices

Also used in the Query Server Tcl shell.

Returns an iterator referencing Device statements in the rule file for devices existing in the DFM database.

Usage

dfm::get_devices [-bad]

Arguments

- -bad

  An optional argument that specifies to return only bad devices. This option is equivalent to the standard Query Server DEVICE BAD command.

  Information about bad devices can be returned from the dfm::get_device_data -location and -seed_layer options.

None.

Return Values

Iterator.

Examples

This example gets all device templates in the database and stores the device names, pin names, and pin layers for further processing.

```
set dev_templates [dfm::get_devices]
while { $dev_templates ne "" } {
  set dev_name [dfm::get_device_data $dev_templates -device_name]
  set pins [dfm::get_device_data $dev_templates -pins]
  set pin_layers [dfm::get_device_data $dev_templates -pin_layers]
# process information
  ...
  dfm::inc dev_templates
}
```

Related Topics

Iterator Processing Commands

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_drc_result_db_magnify

Returns the DRC Magnify Results statement value.

Usage

dfm::get_drc_result_db_magnify

Arguments

None.

Return Values

Floating-point number.

Description

Returns the magnification value specified by the DRC Magnify Results statement. If DRC Magnify Results is not specified in your rule file, this command returns 0.0.

Examples

```
# returns the value of DRC Magnify Results from the rule file
puts "Layout Magnification: [dfm::get_drc_result_db_magnify]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_drc_result_db_precision

Returns the DRC Results Database Precision statement value.

Usage

dfm::get_drc_result_db_precision

Arguments

None.

Return Values

Double-precision floating-point number.

Description

Returns the value specified by the DRC Results Database Precision statement in the rule file. If DRC Results Database Precision is not specified, the Precision statement value is returned.

Examples

```
# returns the value of DRC Results Database Precision
# from the rule file
puts "Layout Magnification: [dfm::get_drc_result_db_precision]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_flat_geometries

Returns an iterator referencing geometries in a cell's context, including sub-hierarchy. The geometries are referenced as though all sub-cells within a cell are expanded to the level of the specified cell. Net names are unique throughout the design hierarchy; however, node IDs are not unique, and they are not flattened by this command. If dfm::get_data queries node ID information from the dfm::get_flat_geometries iterator, shapes from any cell with that ID can return data.

Usage

dfm::get_flat_geometries {***cell_iterator*** | ***cell_name***} **-layers** {***layer_iterator*** | ***layer_list***}
[-window *x1 y1 x2 y2*] [-hierarchy_level *number*] [-net *net_name* | -node *node_ID*]

Arguments

- ***cell_iterator***

  An argument that specifies a cell iterator created with 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.

- **-layers** {***layer_iterator*** | ***layer_list***}

  A required argument set that specifies the layers to retrieve geometries from. Either ***layer_iterator*** or ***layer_list*** must be specified.

  layer_iterator — A layer iterator created with dfm::get_layers.

  layer_list — A Tcl list of layer names.

- -window *x1 y1 x2 y2*

  An optional argument set that specifies a rectangular region. When this argument is specified, the returned iterator references geometries of the specified cell and its hierarchy interacting with the specified region. The window coordinates *x1 y1 x2 y2* are integers specified in database units.

- -hierarchy_level *number*

  An optional argument set that limits the hierarchical levels, in the context of the specified cell, from which to reference geometries. By default, all geometries throughout the cell's hierarchy are referenced. The *number* is a non-negative integer, with 0 being the cell's primary level.

- -net *net_name*

  An optional argument set that specifies a name of a net. Only geometries belonging to the net, and that meet other criteria of the command, are referenced in the output. A net instance path with the following format is permitted: X<*placement-number*>/X<*placement-*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

number>/…/<*net*>. May not be specified with -node. Connectivity extraction must have occurred in order for this argument set to be used.

- -node *node_ID*

  An optional argument set that specifies a numeric net ID. Only geometries belonging to the node ID are referenced in the output. Node IDs are not necessarily unique per cell, so geometries are returned from any cell containing the *node_ID*, and that meet other criteria of the command. May not be specified with -net. Connectivity extraction must have been run in order for this argument set to be used.

Return Values

Iterator.

Examples

Example 1

```
# get all flattened M1 geometries 1 in top-cell hierarchy
set geoFlatIter [dfm::get_flat_geometries [dfm::get_top_cell] \
                 -layers {M1 M2}]
# process the shapes
while {$geoFlatIter != ""} {
    set cName       [dfm::get_data $geoFlatIter -cell_name]
    set lName       [dfm::get_data $geoFlatIter -layer_name]
    set vertices    [dfm::get_data $geoFlatIter -vertices]
# process data
    ...
    dfm::inc geoFlatIter
}
```

Example 2

```
# get all flattened geometries on layers M1 and M2 in top-cell hierarchy
# that intersect with the specified window and belong to the net NET1
set queryRect {0 0 123456 123456}
set topCell [dfm::get top cell]
set geoFlatIter [dfm::get_flat_geometries $topCell -layers "M1 M2" \
    -window $queryRect -net NET1]
# process the shapes
while {$geoFlatIter != ""} {
   set vertices [dfm::get_data $geoFlatIter -vertices]
# process vertices
    ...
   dfm::inc geoFlatIter
}
```

Related Topics

dfm::get_geometries

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_flat_placements

Returns an iterator referencing placements in a cell's context, including sub-hierarchy. The placements are referenced as though all sub-cells within a cell are expanded. Net names are unique throughout the design hierarchy; however, node IDs are not unique, and they are not flattened by this command.

Usage

dfm::get_flat_placements {***cell_iterator*** | ***cell_name***} [-window *x1 y1 x2 y2*]
[-hierarchy_level *number*] [-net *net_name* | -node *node_ID*]

Arguments

- ***cell_iterator***

  An argument that specifies a cell iterator created with 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.

- -window *x1 y1 x2 y2*

  An optional argument set that specifies a rectangular region. When this argument is specified, the returned iterator references geometries of the specified cell and its hierarchy interacting with the specified region. The window coordinates *x1 y1 x2 y2* are integers specified in database units.

- -hierarchy_level *number*

  An optional argument set that limits the hierarchical levels, in the context of the specified cell, from which to reference placements. By default, all placements throughout the cell's hierarchy are referenced. The *number* is a non-negative integer, with 0 being the cell's primary level.

- -net *net_name*

  An optional argument set that specifies a name of a net. Only placements containing the net and that meet other criteria of the command are referenced in the output. A net instance path with the following format is permitted: X<*placement-number*>/X<*placement-number*>/…/<*net*>. May not be specified with -node. Connectivity extraction must have been run in order for this argument set to be used.

- -node *node_ID*

  An optional argument set that specifies a numeric net ID. Only geometries belonging to the node ID are referenced in the output. Node IDs are not necessarily unique per cell, so placements are returned from any cell containing the *node_id* and that meet other criteria of the command. May not be specified with -net. Connectivity extraction must have been run in order for this argument set to be used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

Iterator.

Examples

Example 1

```
# get all flattened placements that interact with the specified window
set queryRect {0 0 123456 123456}
set topcell [dfm::get_top_cell]
set p_iter [dfm::get flat placements $topcell –window $queryRect]
# process flattened placements for the containing cell and extent
while {$p_iter != ""} {
  set c_cell    [dfm::get_data $p_iter -cell_name]
  set extent   [dfm::get_data $p_iter -extent]
# process cell and extent
    ...
  dfm::inc p_iter
}
```

Example 2

```
# get all flat placements in top-cell hierarchy connected to NET1
set topCell [dfm::get top cell]
set p_iter [dfm::get flat placements $topCell -net NET1]
...
```

Related Topics

dfm::get_placements

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_gds_file_info

Returns the database precision for the specified GDS file.

Usage

dfm::get_gds_file_info *gds_file_path* -precision

Arguments

- ***gds_file_path***

  A required argument that specifies the pathname of the GDS file.

- **-precision**

  A required argument that specifies returning the database precision.

Return Values

Floating-point number.

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_geometries

Returns a geometry iterator referencing shapes on a layer. By default, shapes are referenced hierarchically.

Usage

dfm::get_geometries {***layer_iterator*** | ***layer_name***} [-cell {*cell_iterator* | *cell_name*} | -cell_list *list* | -cell_list_name *name*] [-nopseudo]

Arguments

- ***layer_iterator***

  A argument specifying a layer iterator created with dfm::get_layers. Either this argument or ***layer_name*** must be specified.

- ***layer_name***

  An argument specifying a layer name. Either this argument or ***layer_iterator*** must be specified.

- -cell {*cell_iterator* | *cell_name*}

  An optional argument set that specifies a cell in which to reference geometries. The context is the cell itself, not any sub-hierarchy. By default, the top-level cell is assumed. May not be specified with -cell_list or -cell_list_name. One of these arguments is used:

  cell_iterator — A cell iterator created with dfm::get_cells.

  cell_name — A name of a cell.

- -cell_list *list*

  An optional argument that specifies a Tcl list of cell names. Geometries from these cells are referenced, but not any sub-hierarchy. By default, the top-level cell is assumed. May not be specified with -cell or -cell_list_name.

- -cell_list_name *name*

  An optional argument set that specifies a Layout Cell List statement's list name. Geometries from the list's cells are referenced, but not any sub-hierarchy. By default, the top-level cell is assumed. May not be specified with -cell or -cell_list.

- -nopseudo

  An optional argument that specifies pseudo-cells are expanded. Geometries contained in pseudo-cells are promoted into the lowest-level design cells that completely contain the geometries.

Return Values

Iterator.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This example gets all geometries on layer M1 and returns statistics about them to the transcript. Using the -nopseudo option restricts the output to non-pseudo cells.

```
set geometries [dfm::get_geometries M1]
set total_geo_count [dfm::get_geometry_count M1]
set i 1
set old_cell_name ""
puts "Total Geometry Count: $total_geo_count"

while {$geometries ne "" } {
  set vertices [dfm::get_data $geometries -vertices]
  set node_number [dfm::get_data $geometries -node]
  set net_name [dfm::get_data $geometries -net_name]
  set cell_name [dfm::get_data $geometries -cell_name]
  set area [dfm::area $geometries]
  set perimeter [dfm::perimeter $geometries]
  if {$cell_name ne $old_cell_name} {
     set old_cell_name $cell_name
     set layer_name [dfm::get_data $geometries -layer_name]
     set cell_geometry_count [dfm::get_geometry_count $layer_name \
      -cell $cell_name]
     set placement_count [dfm::get_placement_count $cell_name]
     puts "Cell: $cell_name Geometry_count: $cell_geometry_count \
       Placement_count: $placement_count"
     puts "Layer: $layer_name"

  }
  puts "Polygon $i: "
  puts "Net: $node_number $net_name"
  puts "Coords: ($vertices)"
  puts "Area: $area, Perimeter: $perimeter \n"
  dfm::inc geometries
  incr i
}
```

Example 2

This example shows how to print DFM properties on geometries from layer property::prop_out_multi to the transcript.

```
set geometries [dfm::get_geometries property::prop_out_multi]
set prop_list [dfm::list_properties property::prop_out_multi]

while {$geometries ne "" } {
  foreach k $prop_list {
# output only if the property is really present
    if {[dfm::get_data $geometries -has_property $k] != 0} {
      set prop_value [dfm::get_data $geometries -geometry_property $k]
      puts "PROPERTY $k: $prop_value"
    }
  }
  dfm::inc geometries
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

dfm::get_flat_geometries

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_geometry_count

Returns the total number of geometries on a layer.

Usage

dfm::get_geometry_count *layer_name* [-cell *cell_name*]

Arguments

- ***layer_name***

  A required argument specifying a layer name.

- -cell *cell_name*

  An optional argument set that specifies a cell name. Only geometries local to this cell (not the sub-hierarchy) contribute to the count that the command otherwise calculates. By default, geometries in all cells are counted.

Return Values

Integer.

Examples

```
# count M1 shapes
set m1_shape_count [dfm::get_geometry_count M1]
```

Related Topics

Layout Data Query Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_geometry_property

Returns a specified property value from a DFM Analyze _detail layer geometry.

Usage

dfm::get_geometry_property ***layer_name*** **-polygon** ***polygon_number***
-property ***property_name*** [-cell *cell_name*]

Arguments

- ***layer_name***

  A required argument that specifies a DFM Analyze _detail layer.

- **-polygon** ***polygon_number***

  A required argument that specifies the polygon number of the geometry from which to retrieve the property value. Values can be obtained from dfm::get_data -polygon_number.

- **-property** ***property_name***

  A required argument that specifies the name of the property with the value of interest.

- -cell *cell_name*

  An optional argument that specifies the cell containing the geometry with the property. The context is the cell itself, not any sub-hierarchy.

Return Values

Floating-point number.

Examples

Example 1

```
# store the value of the AREA_M1 property from geometry 3 on met1_detail
set m1_area [dfm::get_geometry_property met1_detail -polygon 3 \
             -property "AREA_M1"]
```

For layers with DFM properties other than DFM Analyze _detail layers, use dfm::get_data -has_property. See “Example 2” on page 156.

Related Topics

dfm::get_data

dfm::add_geometry_property

DFM Property Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_lay_vs_netlist_net_name

Returns a list of layout net names by cell that differ from the extracted netlist.

Usage

dfm::get_lay_vs_netlist_net_name -cell {***cell_name*** | ***cell_iterator***} [-all]

Arguments

- **-cell** {***cell_name*** | ***cell_iterator***}

  A required argument set that specifies the cell to query. The context is the cell itself, not any sub-hierarchy. One of these arguments must be used:

  cell_name — Name of a cell.

  cell_iterator — An iterator created by dfm::get_cells. The current cell referenced by the iterator is used.

- -all

  An optional argument that specifies to return all net names from the layout cell. The default is to return only the net names that differ.

Return Values

List, or list of lists if -all is used. In the latter case, each sub-list contains net names that correspond in the layout and extracted netlist, if any.

Description

By default, this command returns a list of net names that appear in a physical cell but do not appear in the extracted netlist. The -all option changes the behavior to report all net names in the cell.

Net naming differences detected by this command arise when Port Text is used. A Port Text object is simply called a “named port” for this discussion. Layout and extracted netlist net names can differ under these conditions:

- An untexted net in the top-level cell has an associated named port. The layout net is unnamed, but the net in the extracted netlist will be named like the associated port, unless the port’s name matches some (other) net name. In the latter case, the net in the extracted netlist remains unnamed.
- A texted net has an invalid SPICE name. The layout net is named, but the net in the extracted netlist is unnamed.

Circuit extraction must have been run in order for this command to be used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

```
# get any net label mismatches for MY_CELL
set mismatch [dfm::get_lay_vs_netlist_net_name -cell MY_CELL]
# report to transcript any differences
if {$mismatch != ""} {
  puts "LABELS IN LAYOUT CONTEXT THAT DIFFER FROM NETLIST CONTEXT:"
  puts "$mismatch"
}
```

Related Topics

Connectivity Commands

Netlist Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_layers

Returns a layer iterator.

Usage

dfm::get_layers [*layer_iterator* | *layer_name*]

Arguments

- *layer_iterator*

  An optional argument that specifies a layer iterator generated by a previous dfm::get_layers command. May not be specified with *layer_name*.

- *layer_name*

  An optional argument that specifies a layer name. May not be specified with *layer_iterator*.

Return Values

Iterator.

Description

Returns a layer iterator. When used without any arguments, all layers in the database are referenced by the returned iterator. When either of the optional arguments is used, the returned iterator references one layer. For the *layer_iterator* argument, the current layer pointed to by that iterator is the one that is referenced by the output iterator.

Examples

This example writes all layer names in the database along with layer types and configurations to the transcript.

```
set layers [dfm::get_layers]
set i 1
while {$layers ne "" } {
   set name [dfm::get_data $layers -layer_name]
   set layer_type [dfm::get_data $layers -layer_type]
   set layer_config [dfm::get_data $layers -layer_configuration]
   puts "$i. $name Type: $layer_type Configuration: $layer_config"
   dfm::inc layers
   incr i
}
```

Related Topics

Layer Management Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_layout_magnify

Also used in the Query Server Tcl shell.

Returns the value of the Layout Magnify specification statement in the rule file that generated the input database.

Usage

dfm::get_layout_magnify

Arguments

None.

Return Values

String.

Examples

```
# returns the value of Layout Path from the rule file
puts "Layout Magnification: [dfm::get_layout_magnify]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_layout_name

Also used in the Query Server Tcl shell.

Returns corresponding layout information for source design objects in a cross-reference (XDB) database.

Usage

dfm::get_layout_name {**-top_cell** |
{***source_name*** {{**-cell_name** [-pin_info]} | {**-device_instance** [-print_pin_swap]} |
-instance | **-net**}}}
[-hcell *layout_cell source_cell*]

Arguments

- **-top_cell**

  An argument that specifies to return the layout primary cell name. May not be specified with ***source_name***.

- ***source_name***

  An argument that specifies a source object name or pathname. The type of object is specified by one of the remaining required arguments. May not be specified with **-top_cell**.

- **-cell_name**

  An argument that specifies the ***source_name*** is the name of a cell. This option corresponds to the Query Server CELL CORRESPONDING LAYOUT command.

- -pin_info

  An optional argument that returns the layout cell pin count. This argument must be specified with **-cell_name**.

- **-device_instance**

  An argument that specifies the ***source_name*** is a device instance name. This option corresponds to the Query Server DEVICE LAYOUT command.

- -print_pin_swap

  An optional argument that appends a 1 to the output name if the layout instance pins have been swapped and 0 otherwise. This argument must be specified with **-device_instance**.

- **-instance**

  An argument that specifies the ***source_name*** is a cell instance name. This option corresponds to the Query Server PLACEMENT LAYOUT command.

- **-net**

  An argument that specifies the ***source_name*** is a net name. This option corresponds to the Query Server NET LAYOUT command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -hcell *layout_cell source_cell*

  An optional argument that restricts the context of the command to the specified hcell pair. This option is not used with **-top_cell** or **-cell_name**.

Return Values

Tcl list.

Description

Returns the corresponding layout object name or pathname for a source object pathname, if the correspondence exists. In Calibre YieldServer, this command is valid only if the dfm::run_compare command has been executed to create an XDB in the DFM database.

The dfm::write_ixf and dfm::write_nxf commands are also useful in determining instance and net correspondence between source and layout.

The dfm::get_source_name command performs the inverse function of this one.

Examples

See the example under dfm::run_compare.

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_layout_path

Also used in the Query Server Tcl shell. Corresponding Query Server command: RULES LAYOUT PATH.

Returns the value of the Layout Path specification statement in the rule file that generated the input database.

Usage

dfm::get_layout_path

Arguments

None.

Return Values

String.

Examples

```
# returns the value of Layout Path from the rule file
puts "Layout: [dfm::get_layout_path]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_layout_path2

Also used in the Query Server Tcl shell.

Returns the value of the Layout Path2 specification statement in the rule file that generated the input database.

Usage

dfm::get_layout_path2

Arguments

None.

Return Values

String.

Examples

```
# returns the value of Layout Path2 from the rule file
puts "Layout Path2: [dfm::get_layout_path2]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_layout_system

Also used in the Query Server Tcl shell. Corresponding Query Server command: RULES LAYOUT SYSTEM.

Returns the value of the Layout System specification statement from the rule file that generated the input database.

Usage

dfm::get_layout_system

Arguments

None.

Return Values

String.

Examples

```
# returns the value of Layout System from the rule file
puts "Layout System: [dfm::get_layout_system]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_layout_system2

Also used in the Query Server Tcl shell.

Returns the value of the Layout System2 specification statement from the rule file that generated the input database.

Usage

dfm::get_layout_system2

Arguments

None.

Return Values

String.

Examples

```
# returns the value of Layout System2 from the rule file
puts "Layout System2: [dfm::get_layout_system2]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_net_name

Returns the name of the net (if any) for a particular node number and cell.

Usage

dfm::get_net_name *node_ID* {**-cell** {***cell_name*** | ***cell_iterator***}} [-netlist]

Arguments

- ***node_ID***

  A required argument that specifies the node number assigned by the connectivity extractor. Node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.

- **-cell** {***cell_name*** | ***cell_iterator***}

  A required argument set that specifies the cell to query. The context is the cell itself, not any sub-hierarchy. One of these arguments must be used:

  cell_name — Name of a cell.

  cell_iterator — An iterator created by dfm::get_cells. The current cell referenced by the iterator is used.

- -netlist

  An optional keyword that returns netlist net names. These net names are as reported in the SPICE netlist written with the dfm::write_spice_netlist command.

Return Values

String.

Examples

See example under “dfm::get_nets” on page 173.

Related Topics

Connectivity Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_net_shapes

Returns a geometry iterator referencing geometries on a specified net.

Usage

dfm::get_net_shapes -net *node_ID* -cell {***cell_iterator*** | ***cell_name***} [-layers *layer_list*] [-nopseudo] [-window *x1 y1 x2 y2*]

Arguments

- **-net *node_ID***

  A required argument that specifies a node ID. The ***node_ID*** is a non-negative integer corresponding to the node number assigned by the connectivity extractor. Only geometries with IDs matching this value are returned. Node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.

- **-cell** {***cell_name*** | ***cell_iterator***}

  A required argument set that specifies the cell to query. The context is the cell itself, not any sub-hierarchy. One of these arguments must be used:

  cell_iterator — An iterator created by dfm::get_cells. The current cell referenced by the iterator is used.

  cell_name — Name of a cell.

- -layers *layer_list*

  Optional argument that causes only nets from specified layers to be referenced. The *layer_list* is a Tcl list of layer names.

- -nopseudo

  Optional argument that causes geometries having the ***node_ID*** in all pseudo-cell placements within the specified cell to be referenced. By default, these geometries are not referenced in the returned iterator.

- -window *x1 y1 x2 y2*

  An optional argument set that specifies a rectangular region. When this argument is specified, the returned iterator references geometries of the specified cell interacting with the specified region. The window coordinates *x1 y1 x2 y2* are integers specified in database units and define the lower-left and upper-right vertices of a rectangle.

Return Values

Iterator.

Examples

This code collects “vdd” net shapes by cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# get vdd shapes by cell
set cells [dfm::get_cells]
while { $cells ne ""} {
  set nets [dfm::get_nets $cells]
  while { $nets ne "" } {
    set node_ID [dfm::get_data $nets -node]
    if { [dfm::get_net_name $node_ID -cell $cells -netlist] == "vdd" } {
      set vdd_shapes [dfm::get_net_shapes -net $node_ID -cell $cells]
      # process vdd_shapes geometry iterator
      ...
    }
    dfm::inc nets
  }
  dfm::inc cells
}
```

Related Topics

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_nets

Also used in the Query Server Tcl shell.

Returns a net iterator.

Usage

dfm::get_nets [*cell_iterator* | *cell_name*] [-flat] [-use_value *use_string*]

Arguments

- *cell_iterator*

  An optional argument specifying a cell iterator created with dfm::get_cells. Only nets from the current cell pointed to by the iterator are referenced. May not be specified with *cell_name*.

- *cell_name*

  An optional argument specifying a cell name. Only nets from this cell are referenced. May not be specified with *cell_iterator*.

- -flat

  An optional argument that specifies to return a flat net iterator rather than a hierarchical one.

- -use_value *use_string*

  An optional argument set that limits referenced nets to those having a LEF/DEF "use" construct that matches *use_string*.

Return Values

Iterator.

Description

Returns a net iterator that references database nets. By default, all nets in the design are referenced. Specifying either of the optional cell arguments limits the iterator to the specified cell and its sub-hierarchy. By default, the top-level cell is assumed.

When used in conjunction with dfm::get_net_name, this command is analogous to the Query Server NET NAMES command.

Examples

Example 1

This code reports net names and node IDs by cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# report net names and ids by cell
set cells [dfm::get_cells]
while { $cells ne ""} {
  set nets [dfm::get_nets $cells]
  while { $nets ne "" } {
    set net_id [dfm::get_data $nets -node]
    puts "CELL: [dfm::get_data $cells -cell_name] \
NET: [dfm::get_net_name $net_id -cell $cells -netlist] id $net_id"
    dfm::inc nets
  }
  dfm::inc cells
}
```

The preceding script could produce output like this:

```
CELL: CellA  NET: INPUT1 id 1
CELL: CellA  NET: 2 id 2
CELL: CellA  NET: INPUT2 id 3
CELL: CellA  NET: OUTPUT id 4
```

Example 2

The -flat option causes nets to be flattened:

```
set net_itr [dfm::get_nets -flat]
while {$net_itr ne ""} {
  puts [dfm::get_data $net_itr -net]
  dfm::inc net_itr
}
```

Nets from Example 1 could be reported like this:

```
X0/INPUT1
X0/2
X0/INPUT2
X0/OUTPUT
```

Related Topics

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_path_context

Also used in the Query Server Tcl shell.

Returns a hierarchical layout path context navigation object.

Usage

dfm::get_path_context {***cell_iterator*** **|** ***cell_name*** **|** ***netlist_object***}
{**-path** ***placement_path*** **|** **-net** ***net_path*** **|** **-device** ***instance_path***}

Arguments

One of the following three arguments must be specified:

- ***cell_iterator***

  An argument that specifies a cell iterator created with dfm::get_cells. The current cell referenced by the iterator is the context cell.

- ***cell_name***

  An argument that specifies a cell name. This cell is the context cell.

- ***netlist_object***

  An argument that specifies a netlist object created with dfm::read_netlist. The top-level cell of the netlist is the context cell.

  One of these three arguments must be specified:

- **-path** ***placement_path***

  An argument set that specifies a cell placement path such as X*m*/X*n*/X*o*.... The root of the path is the specified context cell.

- **-net** ***net_path***

  An argument set that specifies a net path such as X*m*/X*n*/.../<net>. The root of the path is the specified context cell.

- **-device** ***instance_path***

  An argument set that specifies a device instance path such as X*m*/X*n*/.../<device>. The root of the path is the specified context cell.

Return Values

Tcl object.

Description

Returns a hierarchical layout path context navigation object. When the object is created, it is indexed to the final instance in the path.

Each placement (instance of a cell inside another cell) in Calibre is named X<*placement_number*> (for example, placement number 4 is named X4). Placements are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

numbered in order starting from 0. However, if there are any device instances in a cell, they are numbered first. For example, inside a cell, if there are three device instances and three placements of cellA, the cellA placements are numbered X3, X4, and X5, since the numbers 0, 1, and 2 are used for the device instances. If devices have SPICE built-in element names, the letter preceding the number corresponds to the built-in device type.

Placement names containing "/" are paths relative to the specified context cell. For example, if the context cell is TOP, then X0/X1/X2 refers to the X2 placement inside the cell referred to by the X1 placement. The X1 placement is inside the cell referred to by the X0 placement. The X0 placement is inside TOP.

Nets in Calibre are identified by unique numbers called node IDs. Net names are unique across the hierarchy, but node IDs are unique within each cell. If a node number does not have an associated net name, Calibre uses two conventions to refer to it: either the node number and the cell context or a hierarchical path similar to a placement path (for example, X1/X2/4).

The returned Tcl object can be used in dfm::ascend_path_context and the dfm::descend_path_context commands, except when **-device** is used.

The dfm::get_data -path_context option returns a layout path context object. A placement iterator is used as input to the command in that case. The dfm::get_data options that process path context objects include -context_cell_name, -depth, -extent, -leaf_cell_name, -net, -net_is_epin, -net_is_original, -node, -path_name, and -xform.

Other commands that use path context objects include dfm::get_device_data, dfm::get_device_pins, and dfm::get_ports.

Examples

Example 1

Assume this netlist represents the design hierarchy and connectivity:

```
***************************************
.SUBCKT U p1 p2
.ENDS
***************************************
.SUBCKT dev 1 2
X0 1 2 U
.ENDS
***************************************
.SUBCKT C
X0 1 2 dev
X1 1 2 dev
X2 1 2 dev
.ENDS
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
***************************************
.SUBCKT B
X0 C
X1 C
.ENDS
***************************************
.SUBCKT TOPCELL
X0 B
.ENDS
***************************************
```

This code:

```
set path [dfm::get_path_context [dfm::get_top_cell] -path X0/X1/X2]
puts "initial path: [dfm::get_data $path -path_name]"
dfm::ascend_path_context $path
puts "ascent path:  [dfm::get_data $path -path_name]"
```

returns the following:

```
initial path: X0/X1/X2
ascent path:  X0/X1
```

Example 2

Assume the same netlist as in the previous example. This code:

```
proc find_placements { path } {
  set plItr [dfm::get_placements $path]
  while { $plItr ne "" } {
    dfm::descend_path_context $path $plItr
    puts "descent path: [dfm::get_data $path -path_name]"
    find_placements $path
    dfm::ascend_path_context $path
    set ascentPath [dfm::get_data $path -path_name]
    if {$ascentPath != ""} {
      puts "ascent path:  $ascentPath"
    } else {
      puts "ascend path: [dfm::get_top_cell]"
    }
    dfm::inc plItr
  }
}
set inPath "X0/X1"
set path [dfm::get_path_context [dfm::get_top_cell] -path "$inPath"]
find_placements $path
```

returns the following:

```
descent path: X0/X1/X0
ascent path:  X0/X1
descent path: X0/X1/X1
ascent path:  X0/X1
descent path: X0/X1/X2
ascent path:  X0/X1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Depending on the design size and hierarchy complexity, doing this type of hierarchy traversal can take a long time and produce much output. If you are simply looking for placements of a particular cell, see Example 3 under dfm::get_placements for a more efficient search method.

Example 3

Assume the same netlist as in Example 1. This code uses a net path:

```
set path [dfm::get_path_context [dfm::get_top_cell] -net X0/X1/X2/2]
puts "initial path: [dfm::get_data $path -path_name]"
dfm::ascend_path_context $path -to_top
puts "ascent path:  [dfm::get_data $path -path_name]"
```

and returns the following:

```
initial path: X0/X1/X2
ascent path:  X0
```

Also see Example 1 and Example 2 under dfm::get_data and "Example: Report Net Instance Connections Up to a Context Cell" on page 311.

In the Query Server Tcl shell, the qs::net_trace command can be useful in this context.

Related Topics

Hierarchy Traversal Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_pins

Returns a pin iterator referencing pins of a cell placement.

Usage

dfm::get_pins *placement_iterator* [-net {*net_iterator* | *node_ID*}]

Arguments

- ***placement_iterator***

  A required argument specifying a placement iterator created with dfm::get_placements.

- -net {*net_iterator* | *node_ID*}

  An optional argument that causes only pins from the specified net to be referenced. One of these arguments is used.

  net_iterator — A net iterator created by dfm::get_nets. The current net referenced by the iterator is used.

  node_ID — A non-negative integer corresponding to the net ID assigned by the Calibre circuit extractor. Node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.

Return Values

Iterator.

Examples

This example finds all placements in the top-level cell and gets their pins. It then descends the nets connected to those pins and reports the net connections in the placements.

```
# initialize a list containing placed cells at the top level
set p [dfm::get_placements [dfm::get_top_cell]]
# go through placements, descend nets connected further down,
# and report the connections in the transcript.
while {$p ne ""} {
  set ppins [dfm::get_pins $p]
  set pname [dfm::get_data $p -placement_name]
  while {$ppins ne ""} {
    set top_net [dfm::get_data $ppins -node]
    set n [dfm::descend_net $ppins]
    set pnet [dfm::get_data $n -node]
    puts "   Net $top_net in TOP is connected to $pname/$pnet"
    dfm::inc ppins
  }
  dfm::inc p
}
```

Related Topics

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_placement_count

Returns the total number of placements of a cell.

Usage

dfm::get_placement_count ***cell_name***

Arguments

- ***cell_name***

  A required argument defining the cell whose total number of placements are counted. For the top-level cell, the placement count is 0.

Return Values

Non-negative integer.

Examples

```
# store the count of AND4 cells
set AND4_count [dfm::get_placement_count AND4]
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_placements

Also used in the Query Server Tcl shell.

Returns a placement iterator referencing placements in the immediate context of a cell, not its sub-hierarchy. This can be changed with the -flat option.

Usage

dfm::get_placements {***cell_iterator*** | ***cell_name***} [[-of_cell *placed_cell_name*] | [-net {*net_iterator* | *node_ID*}]] [-flat [-list]]

Arguments

- ***cell_iterator***

  An argument that specifies a cell iterator created with 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.

- -of_cell *placed_cell_name*

  An optional argument set that causes only placements of the cell having the *placed_cell_name* to be referenced. May not be specified with -net. By default, the placed cell must be referenced at the primary level of the cell referenced by the ***cell_iterator*** or ***cell_name*** argument, or the output will be empty. The -flat option changes this behavior.

- -net {*net_iterator* | *node_ID*}

  An optional argument set that causes only placements connected to the specified net to be referenced. One of these arguments is used:

  net_iterator — A net iterator created by dfm::get_nets. The current net from the iterator is used.

  node_ID — A non-negative integer corresponding to the net ID assigned by the connectivity extractor. Node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.

- -flat

  Argument that specifies to flatten the placement sub-hierarchy in the referenced ***cell_iterator*** or ***cell_name***. The output references the flattened placements.

- -list

  Argument specified with -flat that returns a Tcl list instead of an iterator.

  If -of_cell is not also specified, then the list is a list of lists. For each sub-list, a path to every flat placement within the current cell is included, along with the cell name of the placement: {X*i*/X*j*/... X*m* cell_name}.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If -of_cell is also specified, then a simple list of placement names is returned. The placement names correspond to the instances of the -of_cell argument in the specified cell context.

The practical limit of the list size is approximately 4E5 elements, so this option should not be used for very large numbers of placements.

Return Values

Iterator, or list if -list is used.

Description

Returns an iterator referencing placements within a cell by default. The placements are those referenced at the primary level of the input cell (not its sub-hierarchy) by default.

The -of_cell option causes only placements of the referenced *placed_cell_name* to be returned. Again, the placements are those referenced at the primary level of the input cell (not its sub-hierarchy) by default.

The -flat option returns a flat placement path iterator to referenced placements in the input cell and its sub-hierarchy. This is not the same as the usual placement iterator, and cannot necessarily be used in the same way. The paths can be accessed through the dfm::get_data -path_name option, and the cell name of the leaf cell can be accessed through the dfm::get_data -cell_name option.

When -of_cell is used with -flat, then placements of the *placed_cell_name* within the context cell's sub-hierarchy are included in the output.

The -list option causes the command to return a list of placements rather than an iterator. The output is similar information to the Query Server PLACEMENT NAMES FLAT command. If the -of_cell option is additionally used, the output is similar to the PLACEMENTS OF ... FLAT option.

The dfm::ascend_net and dfm::descend_net commands are useful in traversing nets across hierarchical cell boundaries from placements. This can be used in the context of the -net option output.

Examples

Example 1

See the examples under dfm::get_pins.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

These procs show two ways of returning the same data using flat placement names, one with an iterator, and the other with a list:

```
proc list_paths1 {} {
  set iterator [dfm::get_placements [dfm::get_top_cell] -flat]
  while {$iterator ne ""} {
    set path [dfm::get_data $iterator -path_name]
    set name [dfm::get_data $iterator -cell_name]
    puts "$path $name"
    dfm::inc iterator
  }
}

proc list_paths2 {} {
  foreach path_name [dfm::get_placements [dfm::get_top_cell] \
    -flat -list] {
    puts $path_name
  }
}
```

Calling either proc returns data like this to the transcript:

```
X0 CellA
X0/X0 nand
X1 CellA
X1/X0 nand
X2 CellB
X2/X0 inv
```

Adding the option “-of_cell CellA” to the code causes instance names of that cell to be returned instead:

```
X0
X1
```

Example 3

This code finds and reports all flat placement paths of a given cell C:

```
set plItr [ dfm::get_placements [dfm::get_top_cell] -of_cell C -flat ]
while { $plItr ne "" } {
  puts "C path: [dfm::get_data $plItr -path_name]"
  dfm::inc plItr
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

This example reports extents of cell instances in top-level coordinates along with transformation information:

```
set target_cell "nand"
set pl_list [dfm::get_placements [dfm::get_top_cell] \
  -of_cell $target_cell -flat -list]
set list_size [llength $pl_list]
set idx [expr $list_size - 1]
puts "Placements of $target_cell"

while {$idx >= 0} {
  set inst [lindex $pl_list $idx]
  puts "  -- $inst"
  set pc [dfm::get_path_context [dfm::get_top_cell] -path $inst]
  set extent [dfm::get_data $pc -extent]
  puts "     Extent    :   $extent"
  set xobj [dfm::get_data $pc -xform]
  puts "     Rotation  :   [dfm::get_xform_data $xobj -rotation]"
  puts "     Reflection:   [dfm::get_xform_data $xobj -reflection"
  incr idx -1
}
```

Related Topics

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_port_data

Also used in the Query Server Tcl shell. This command provides a superset of the information given by the dfm::get_data command's port options and the Query Server PORT INFO command.

Returns information about ports. One of the data type arguments must be specified in addition to a port iterator.

Usage

dfm::get_port_data ***port_iterator*** [-filter *filter_object*] {**-cell_name** | **-connectivity_layer** | **-layer** | **-location** | **-net** | **-net_name** | **-node** | **-object_type** | **-port_dir** | **-port_info** | **-port_name** | **-use_value**}

Arguments

- ***port_iterator***

  A required port iterator created with dfm::get_ports.

- -filter *filter_object*

  An optional argument set specifying a dfm::create_filter object containing transformation information. When used, this argument set must precede the **-location** or **-port_info** argument. Only those arguments observe filters.

- **-cell_name**

  Returns the name of the current port's parent cell.

- **-connectivity_layer**

  Returns the name of the connectivity layer the port is attached to.

- **-layer**

  Returns the name of the port's defined layer.

- **-location**

  Returns the x,y location of the port in database units in the coordinate space of the queried cell. If a path context object was used to create the ***port_iterator***, then the coordinate space is that of the root cell of the path context object.

- **-net**

  Returns the current port's net path.

- **-net_name**

  Returns the current port's net name, if any.

- **-node**

  Returns the current port's node ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-object_type**

  Returns the type of iterator used for the command.

- **-port_dir**

  Returns the current port's direction of current flow such as, in, out, in/out, or unknown.

- **-port_info**

  Returns the current port's concise statistics. See the Examples section for a use case.

- **-port_name**

  Returns the current port's name.

- **-use_value**

  Returns the current port's "use" construct name. This is for LEF/DEF designs only.

Return Values

The return values depend on the data type requested from the command. In most cases, it is a simple string. If a port occurs more than once, a list of values from all locations can be returned. The **-port_info** return value is shown in the Examples.

Examples

The **-port_info** option returns statistics about the current port in the ***port_iterator***. This shows the output from an interactive session:

```
> set ports [dfm::get_ports TOPCELL]
Loading hierarchy and connectivity
Port Iterator
> dfm::get_port_data $ports -port_info
vss {2290 3380} 1 port metal1
> dfm::inc ports
> dfm::get_port_data $ports -port_info
vdd {1330 82450} 2 port metal1
```

The general form of the output is this:

```
<port_name> {<x> <y>} <node_number> <port_layer> <connectivity_layer>
```

Related Topics

dfm::get_data

Connectivity Commands

Iterator Processing Commands

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_ports

Also used in the Query Server Tcl shell.

Returns a port iterator for a given cell.

Usage

dfm::get_ports {{***cell_iterator*** | ***cell_name*** | ***path_context_object***} [-filter *filter_object*] [-flat] [-layer *layer_name*] [-location '{'*x y*'}'] [-name *port_name*] [-net {*net_name* | *node_ID* | *net_iterator*}] [-spice_name {valid | invalid}] [-use_value *use_string*] [-window *x1 y1 x2 y2*]} | {**-list** [-cells]}

Arguments

The first three arguments may only be used in YieldServer.

- ***cell_iterator***

  An argument that specifies a cell iterator created with dfm::get_cells.

- ***cell_name***

  An argument that specifies a cell name.

- ***path_context_object***

  An argument that specifies a path context object created by dfm::get_path_context. The placement context cell, if any, is the cell in which the query for ports takes place. (If there is no placement context, the object's root cell is used.) However, any input or output coordinates are always in the coordinate space of the root cell of the path context. If the path context object encapsulates a net, results are constrained to ports on that net.

- -filter *filter_object*

  An optional argument set that limits referenced ports to those on layers referenced in the filter object generated by dfm::create_filter. Any geometric transforms referenced in the filter object are used when the -location option is also specified.

- -flat

  An optional argument that specifies to return a flat instance iterator rather than a hierarchical one.

- -layer *layer_name*

  An optional argument set that limits referenced ports to those on specified layer name.

- -location '{'*x y*'}'

  An optional argument set that limits referenced ports to the ones at a given location. The x and y coordinates are specified in a Tcl list. Coordinates are in database units of the coordinate space of the cell being queried, or, when a ***path_context_object*** has been provided, in the coordinate space of the root cell of that object. If a candidate port has multiple locations, only results having the given location are included. Typically the result

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set includes either zero or one entries. This option corresponds to the Query Server PORT LOCATION command.

If the -filter option is also used, the coordinates are modified by any geometric transformation information in the *filter_object*.

- -name *port_name*

  An argument set that limits referenced ports to the ones with the given name. Typically this gives zero or one result, but there can be multiple results in cases where the given port has multiple locations.

- -net {*net_name* | *node_ID* | *net_iterator*}

  An argument set that limits referenced ports to the ones on the given net. A name, node number (node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.), or a net iterator (from dfm::get_nets) is specified to identify the net. When the connections exist within the context of the referenced cell or the -flat is also specified, the *net_name* may be a hierarchical path. A path is traced to its highest hierarchical context before searching for connected ports.

  This option may not be used when a net-based ***path_context_object*** is specified.

  This option corresponds to the Query Server NET PORTNAMES and NET PORTS commands.

- -spice_name {valid | invalid}

  An optional argument set that limits referenced ports to those having either valid or invalid SPICE names. Invalid names include the null name case, such as for polygon ports. By default, both port name types are referenced. This option corresponds to the Query Server PORT TEXT MAP command.

- -use_value *use_string*

  An optional argument set that limits referenced ports to those having a LEF/DEF “use” construct that matches *use_string*.

- -window *x1 y1 x2 y2*

  An optional argument set that limits referenced ports to those lying within a specified rectangular region. The *x1 y1 x2 y2* arguments specify vertices on opposite corners of the rectangle. Coordinates are in user units. Port locations are in the coordinate space of the cell being queried, or, when a ***path_context_object*** is specified, in the coordinate space of the root cell of that object. If a candidate port has multiple locations, only those locations that fall within the specified window are included in the result set.

- **-list**

  An argument that causes the output to be a Tcl list of lists. This argument is required in Query Server Tcl shell. This option requires a Calibre Connectivity Interface license. The Mask SVDB Directory CCI option must have been used in the rules for the run that generated the currently loaded database. This option corresponds to the Query Server PORT NAMES command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -cells

  An argument specified with **-list** that causes ports in all cells in the hierarchy of the current cell to be output. By default, only top-level ports are returned.

Return Values

Port iterator by default. Tcl list of lists if **-list** is specified.

When **-list** is specified, each sub-list in the output is of this form:

```
{port_name node_ID net_name x y 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>.

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 — Real numbers corresponding to the port's coordinates. By default, these are in top-level space. If -cells is specified, then cell space is used.

layer_name — Name of the layer to which the port is attached.

cell_name — Name of the cell in which the port appears when -cells is used. By default, this field is empty.

Description

Initializes an iterator that can be used to access cell ports. In order for this command to be used, the DFM database must have connectivity extraction information. The various options other than **-list** constrain which ports are present in the iterator.

When -net is used, a given port may have multiple locations on the net, and each port instance is returned.

When used with **-list**, the command returns concise port information similar to what PORT TABLE WRITE provides in the Calibre Connectivity Interface (CCI). The qs::port_table command gives a complete report like PORT TABLE WRITE.

Examples

Example 1

This example creates a port iterator for top-level ports and then builds a list of lists of port data in a loop.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set p_itr [dfm::get_ports [dfm::get_top_cell]]
while {$p_itr ne ""} {
  set port_info [lappend port_info "[dfm::get_port_data $p_itr\
    -port_info] "]
  dfm::inc p_itr
}
```

Example 2

This shows the **-list** output:

```
dfm::get_ports -list
{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. Cell names are not reported in this case. Ports generated from Port Layer Polygon are shown as <UNNAMED>.

Related Topics

Connectivity Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_revision_info

Also used in the Query Server Tcl shell.

Returns revision information of a DFM database.

Usage

dfm::get_revision_info [-db *dfmdb_name* [-db_revision *revision*]] ***info_type***

Arguments

- -db *dfmdb_name*

  An optional argument set that specifies a DFM database that is not currently loaded. The *dfmdb_name* is the path to the DFM database. By default, the currently loaded database is queried.

- -db_revision *revision*

  An optional argument set that specifies a revision of the *dfmdb_name* to query. The *revision* is a revision name or ID number.

- ***info_type***

  A required argument that specifies the type of revision information to return from the DFM database:

  -parent — Returns the parent revision.

  -tree — Returns the current revision and parent revision (if any).

  -children — Returns a list of direct children of the revision (if any).

  -lock_info — Returns the host name and process ID of the revision lock holder.

  -status — Returns a list of state information regarding the revision. Possible return values include MASTER, DEFAULT, LOCKED, and FROZEN.

  -calibre_build — Returns the Calibre build that was used to save the revision.

Return Values

Tcl list.

Examples

```
# return revision status information on the current DFM database
> dfm::get_revision_info -status
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_source_name

Also used in the Query Server Tcl shell.

Returns corresponding source information for layout design objects in a cross-reference database (XDB).

Usage

dfm::get_source_name {**-top_cell** |
{***layout_name*** {{**-cell_name** [-pin_info]} | {**-device_instance** [-print_pin_swap]} |
-instance | **-net**}}} [-hcell *layout_cell source_cell*]

Arguments

- **-top_cell**

  An argument that specifies to return the source primary cell name. May not be specified with ***layout_name***.

- ***layout_name***

  A argument that specifies a layout object name or pathname. The type of object is specified by one of the remaining required arguments. May not be specified with **-top_cell**.

- **-cell_name**

  An argument that specifies the ***layout_name*** is the name of a cell or a cell iterator. This option corresponds to the Query Server CELL CORRESPONDING SOURCE command.

- -pin_info

  An optional argument that returns the source cell pin count. This argument may only be specified with **-cell_name**.

- **-device_instance**

  An argument that specifies the ***layout_name*** is a device instance name. This option corresponds to the Query Server DEVICE SOURCE command.

- -print_pin_swap

  An optional argument that appends a 1 to the output name if the source instance pins have been swapped and 0 otherwise. This argument must be specified with **-device_instance**.

- **-instance**

  An argument that specifies the ***layout_name*** is a cell instance name. This option corresponds to the Query Server PLACEMENT SOURCE command.

- **-net**

  An argument that specifies the ***layout_name*** is a net name. This option corresponds to the Query Server NET SOURCE command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -hcell *layout_cell source_cell*

  An optional argument that restricts the context of the command to the specified hcell pair. This option is not used with **-top_cell** or **-cell_name**.

Return Values

Tcl list.

Description

Returns the corresponding source object name for a layout object name or pathname, if the correspondence exists. In Calibre YieldServer, this command is valid only if the dfm::run_compare command has been executed to create an XDB in the DFM database.

The dfm::write_ixf and dfm::write_nxf commands are also useful in determining instance and net correspondence between source and layout.

The dfm::get_layout_name command performs the inverse function of dfm::get_source_name.

Examples

```
# Get the source cell name for layout cell AD4FUL
set source_cell [dfm::get_source_name "AD4FUL" -cell_name]

# Get the pin count along with the source cell name
set source_cell_p [dfm::get_source_name "AD4FUL" -cell_name -pin_info]

# Create an hcell list using source and layout cell
set hcell_list [list "AD4FUL" $source_cell]

# Get the source device instance name corresponding to X20/X45/X5/M0
# along with pin swap status.
set sdi [dfm::get_source_name X20/X45/X5/M0 -device_instance \
  -hcell $hcell_list -print_pin_swap]

# Get the source net name for layout net X20/4 in layout cell "AD4FUL"
set snn [dfm::get_source_name X20/4 -net -hcell $hcell_list]
```

Related Topics

LVS and CCI Commands

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_source_path

Also used in the Query Server Tcl shell. Corresponding Query Server command: RULES SOURCE PATH.

Returns the value of the Source Path specification statement in the rule file that generated the cross-reference (XDB) database.

Usage

dfm::get_source_path

Arguments

None.

Return Values

String.

Examples

```
# returns the value of Source Path from the rule file
puts "Source: [dfm::get_source_path]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_source_system

Also used in the Query Server Tcl shell. Corresponding Query Server command: RULES SOURCE SYSTEM.

Returns the value of the Source System specification statement from the rule file that generated the XDB database. LVS comparison must have been run for an XDB to exist.

Usage

dfm::get_source_system

Arguments

None.

Return Values

String.

Examples

```
# returns the value of Source System from the rule file
puts "Source System: [dfm::get_source_system]"
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_svrf_data

Retrieves information from an SVRF analyzer or SVRF layer trace object.

Usage

dfm::get_svrf_data {***svrf_analyzer*** | ***svrf_connect_trace*** | ***svrf_layer_trace***} ***option***

Arguments

One of the following four arguments must be specified.

- ***svrf_analyzer***

  An argument that specifies an SVRF analyzer object created with dfm::create_svrf_analyzer.

- ***svrf_connect_trace***

  An argument that specifies an object created by the dfm::get_svrf_data -connect_trace option.

- ***svrf_layer_trace***

  An argument that specifies an object created by the dfm::get_svrf_data -layer_trace option.

- ***option***

  A required option from the following table.

Table 3-21. Options for get_svrf_data

Option	Applies to object	Return Values
-alias	***svrf_layer_trace***	The alias for the current layer in the layer trace object.
-connectivity	***svrf_connect_trace***	The Connect statements referencing the layer currently pointed to by the connect trace object.
-connect_trace *layername*	***svrf_analyzer***	Tcl object of the connect trace for the specified layer.
-depth	***svrf_layer_trace***	The depth level layer trace location in the layer derivation tree.
-freeze	***svrf_analyzer***	Outputs the (active) rule file text that was used to generate the database. The output is identical to the calibre -svrf FREEZE command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-21. Options for get_svrf_data (cont.)

Option	Applies to object	Return Values
-full_specification "*specfilter*"	***svrf_analyzer***	Tcl list of all the specification statements that match the filter conditions. The *specfilter* is a quoted string that specifies an SVRF specification statement to return. Asterisk (*) wild cards are supported and match zero or more characters. See "Example 2" on page 198.
-is_connect	***svrf_layer_trace***	Returns 1 if the current layer in the layer trace object is a Connect layer, and 0?otherwise.
-is_derived	***svrf_layer_trace***	Returns 1 if the current layer in the layer trace object is a derived layer, and 0 otherwise.
-is_original	***svrf_layer_trace***	Returns 1 if the current layer in the layer trace object is an original layer, and 0?otherwise.
-layer_name	***svrf_layer_trace***	The layer name currently pointed to by the layer trace object.
-layer_number	***svrf_layer_trace***	The layer number currently pointed to by layer trace object.
-layer_trace *layername*	***svrf_analyzer***	Tcl object to the layer trace for the specified layer. The object contains all layers involved in producing the specified layer.
-list_alias	***svrf_analyzer***	List of layer names and corresponding aliases in the rule file.
-list_checks	***svrf_analyzer***	List of rule checks in the rule file.
-list_devices	***svrf_analyzer***	List of Device statements in the rule file. Each line ends with a comment string indicating the ordinal number of each Device statement as it appears in the rule file indexed from 0. This corresponds to the $D property in an extracted netlist. Additionally, the rule file name and line number appears in the comment string.
-list_layers [*check_name*]	***svrf_analyzer***	List of layers in the rule file. If a *check_name* is specified, then only the layers belonging that check are returned.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-21. Options for get_svrf_data (cont.)

Option	Applies to object	Return Values
-op_text	***svrf_layer_trace***	The operation string of the layer currently pointed by the layer trace object.
-variable *variable_name*	***svrf_analyzer***	The value for the SVRF *variable_name* defined in the rule file.

Examples

Example 1

```
# returns a Tcl list of all checks in the rule file "rules"
set rules_analyzer [dfm::create_svrf_analyzer rules]
set check_names [dfm::get_svrf_data $rules_analyzer -list_checks]
```

Example 2

The following example returns a list of all Layer Map statements:

```
set sa [dfm::create_svrf_analyzer rules]
set spec_list [dfm::get_svrf_data $sa -full_specification "LAYER MAP*"]
```

A returned list contains elements similar to this:

```
{LAYER MAP > 11 DATATYPE >= 0 12}
```

Also see the “Examples” section of dfm::run_compare.

Example 3

This example shows how to set up a layer trace object and iterate through it in a loop. All of the layers in the rule file that contribute to the derivation of the psd layer are checked for existence in the DFM database.

```
# all layer names in the DB
set dbLayers [dfm::list_layers -no_ids]

# SVRF analyzer object
set sa [dfm::create_svrf_analyzer rules]

# a layer name to trace
set layerOfInterest psd

# trace the ancestor layers of layerOfInterest
set aliasTrace [dfm::get_svrf_data $sa -layer_trace $layerOfInterest]

while {$aliasTrace ne ""} {
  set traceLayer [dfm::get_svrf_data $aliasTrace -layer_name]
# if the search value is other than -1, then the layer is in the DB.
  puts ">>>>>TRACED $traceLayer WITH SEARCH VALUE: \
    [lsearch $dbLayers $traceLayer]"
  dfm::inc aliasTrace
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The code produces output like this:

```
>>>>>TRACED psd WITH SEARCH VALUE:  5
>>>>>TRACED pdiff WITH SEARCH VALUE:  -1
>>>>>TRACED diff WITH SEARCH VALUE:  3
>>>>>TRACED pplus WITH SEARCH VALUE:  2
>>>>>TRACED poly WITH SEARCH VALUE:  4
```

The pdiff layer is in the rules, but it is not in the DFM database.

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_timer_data

Also used in Query Server Tcl shell.

Returns time and memory use information from a timer object. Time and memory information is as what appears in the Calibre run transcript.

Usage

dfm::get_timer_data ***timer_object***
{**-object_type** | **-cpu_time** | **-wall_time** | **-elapsed_time** | **-lvheap** | **-malloc** | **-print**}

Arguments

- ***timer_object***

  A required Tcl object for the timer created with the dfm::create_timer command.

- **-object_type**

  An argument that returns the type of object.

- **-cpu_time**

  An argument that returns the cumulative CPU TIME usage since timer creation.

- **-wall_time**

  An argument that returns the cumulative REAL TIME usage since timer creation.

- **-elapsed_time**

  An argument that returns the ELAPSED TIME usage. This is global elapsed time since the process started.

- **-lvheap**

  An argument that returns the LVHEAP usage.

- **-malloc**

  An argument that returns the MALLOC usage.

- **-print**

  An argument that prints the Calibre transcript format time and LVHEAP memory data. The CPU and real time are reported as the difference since the last call to -cpu_time, -wall_time, or -print. LVHEAP is reported as it normally is.

Return Values

String.

Examples

Prints the transcript form of time and memory data:

```
set timer [dfm::create_timer]
puts "LOG DATA: [dfm::get_timer_data $timer -print]"
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Timer Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_top_cell

Also used in the Query Server Tcl shell.

Returns the name of the layout top-level cell.

Usage

dfm::get_top_cell

Arguments

None.

Return Values

String.

Examples

```
# get child cells of top level
set child_cells [dfm::list_children [dfm::get_top_cell]]
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_unit_length

Returns the Unit Length statement value from the rule file. By default, the value is 1e-06.

Usage

dfm::get_unit_length

Arguments

None.

Return Values

String.

Examples

```
# returns the user unit of length to the transcript
set ul [dfm::get_unit_length]
puts "User unit: $ul"
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_xform_data

Retrieves data from a specified transformation or creates a new transform object.

Usage

dfm::get_xform_data ***transform*** {**-rotation** | **-reflection** | **-composite** ***parent_transform***}

Arguments

- ***transform***

  A Tcl object created by the dfm::get_data -xform or dfm::get_xform_data -composite command.

  One of the following must be specified.

- **-rotation**

  An argument that returns the rotation angle in degrees.

- **-reflection**

  An argument that returns 1 if the transform is a reflection and 0 otherwise.

- **-composite** ***parent_transform***

  Returns the composite of the ***transform*** and that of a parent cell (specified with ***parent_transform***). This creates a new transform Tcl object.

Return Values

Angle of rotation, reflection condition, or the composite of the transform and its parent as a new Tcl object.

Description

Retrieves data from a specified DRC transform.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

```
# this code gives output similar to the Query Server PLACEMENT TRANSFORM
# command.
# path context navigation object for top-level instance X27.
set pc [dfm::get_path_context [dfm::get_top_cell] -path X27]
# write the cell name.
puts "[dfm::get_data $pc -leaf_cell_name]"
# get an xform object.
set xobj [dfm::get_data $pc -xform]
# get the rotation and reflection.
set rot [dfm::get_xform_data $xobj -rotation]
set ref [dfm::get_xform_data $xobj -reflection]
# get the extent and coords of lower-left vertex.
set extent [dfm::get_data $pc -extent]
set x [lindex $extent 0]
set y [lindex $extent 1]
# print the result in Query Server order.
puts "$x $y $ref $rot"
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_xref_cell_data

Also used in the Query Server Tcl shell. Corresponding Query Server commands: CELLS CORRESPONDING, CELLS LAYOUT, and CELLS SOURCE.

Returns information from a cell cross-reference iterator generated by dfm::get_xref_cells. If dfm::get_xref_cells is used with the -layout or -source options, then the iterator contains references to single cell names. If dfm::get_xref_cells is used with the -corresponding option, then the iterator contains references to hcell name pairs.

Usage

dfm::get_xref_cell_data ***xref_iterator*** {**-cell_name** | **-hcell_pair** | **-layout_name** | **-layout_pin_count** | **-pin_count** | **-source_name** | **-source_pin_count**}

Arguments

- ***xref_iterator***

  A required cross-reference cell name iterator created with dfm::get_xref_cells.

- **-cell_name**

  Returns the name of the current cell referenced by the ***xref_iterator***. This option is used when the dfm::get_xref_cells -layout or -source option is used to create the iterator.

- **-hcell_pair**

  Returns the names of the current cells referenced by the ***xref_iterator*** in a Tcl list. This option is used when the dfm::get_xref_cells -corresponding option is used to create the iterator.

- **-layout_name**

  Returns the name of current layout cell name referenced by the ***xref_iterator***. This option is used when the dfm::get_xref_cells -corresponding option is used to create the iterator.

- **-layout_pin_count**

  Returns the pin count of current layout cell referenced by the ***xref_iterator***. This option is used when the dfm::get_xref_cells -corresponding option is used to create the iterator.

- **-pin_count**

  Returns the pin count of the current cell referenced by the ***xref_iterator***. This option is used when the dfm::get_xref_cells -layout or -source option is used to create the iterator.

- **-source_name**

  Returns the name of current source cell name referenced by the ***xref_iterator***. This option is used when the dfm::get_xref_cells -corresponding option is used to create the iterator.

- **-source_pin_count**

  Returns the pin count of current source cell referenced by the ***xref_iterator***. This option is used when the dfm::get_xref_cells -corresponding option is used to create the iterator.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

String, or if **-hcell_pair** is specified, a Tcl list.

Examples

This example shows code that outputs corresponding cell names and pin counts:

```
set xrefItr [dfm::get_xref_cells -corresponding]
while {$xrefItr ne ""} {
  puts "Source cell: [dfm::get_xref_cell_data $xrefItr -source_name]\n \
Pins: [dfm::get_xref_cell_data $xrefItr -source_pin_count]";
  puts "Layout cell: [dfm::get_xref_cell_data $xrefItr -layout_name]\n \
Pins: [dfm::get_xref_cell_data $xrefItr -layout_pin_count]";
  puts "";
  dfm::inc xrefItr;
}
```

Related Topics

Iterator Processing Commands

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::get_xref_cells

Also used in the Query Server Tcl shell. Corresponding Query Server commands: CELLS CORRESPONDING, CELLS LAYOUT, and CELLS SOURCE.

Returns an iterator that references cell names in an LVS comparison cross-reference database (XDB). An XDB must exist for the command to be used. In Calibre YieldServer usage, dfm::run compare is used to generate an XDB. The iterator output of this command is used by dfm::get_xref_cell_data.

Usage

dfm::get_xref_cells {**-corresponding** | **-layout** | **-source**}

Arguments

- **-corresponding**

  Option that specifies to return references to names of corresponding layout and source cell pairs.

- **-layout**

  Option that specifies to return references to layout cell names.

- **-source**

  Option that specifies to return references to source cell names.

Return Values

Iterator.

Examples

```
# create an xref cell iterator and list all corresponding
# cells with their pin counts
set xrefItr [dfm::get_xref_cells -corresponding]
while {$xrefItr ne ""} {
  puts "Source cell: [dfm::get_xref_cell_data $xrefItr -source_name]\n \
    Pins: [dfm::get_xref_cell_data $xrefItr -source_pin_count]";
  puts "Layout cell: [dfm::get_xref_cell_data $xrefItr -layout_name]\n \
    Pins: [dfm::get_xref_cell_data $xrefItr -layout_pin_count]";
  puts "";
  dfm::inc xrefItr;
}
```

Related Topics

Iterator Processing Commands

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::help

Lists all command summaries.

Usage

dfm::help [-gui]

Arguments

- -gui

  Optional argument that displays command summaries in an external window.

Return Values

Tcl object.

Examples

Example 1

Interactive session usage for returning the entire help page to the terminal:

```
> dfm::help
Yield Server Commands:
   (For usage help on commands try <command name> -help)   Annotation
Commands:
       =================================================================
   dfm::add_annotation                     Adds an annotation to
                                           layer/db_revision.
   dfm::delete_annotation                  Deletes an annotation from
                                           layer/db_revision.
...
```

or saving the help page to a variable for later access:

```
> set help [dfm::help]
```

Example 2

Interactive usage for searching the help page for lines containing a string:

```
> puts [exec echo "[dfm::help]" | awk {/dfm::get_dev/}]
  dfm::get_device_data          Returns device-specific information on an
  dfm::get_device_geometries    Returns an iterator to geometries of a
  dfm::get_device_instances     Returns an iterator to device instances in
  dfm::get_device_pins          Creates an iterator to device pins in a
  dfm::get_devices              Returns an iterator to each device type in
```

Related Topics

Server Administration Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::inc

Also used in the Query Server Tcl shell.

Increments the specified iterator or SVRF layer trace object.

Usage

dfm::inc {***iterator*** | ***svrf_object***}

Arguments

- ***iterator***

  An argument that specifies an iterator. May not be specified with ***svrf_object***.

- ***svrf_object***

  An argument that specifies a Tcl object created by dfm::get_svrf_data -layer_trace.

Return Values

None.

Description

Increments the specified ***iterator*** or layer trace ***svrf_object*** to point at the next member of the list. When the ***iterator*** or ***svrf_object*** reaches the end, it resets the string representation to an empty string (“ ”). It is always a good practice to test for the empty string after incrementing an ***iterator*** or ***svrf_object***.

It is not permitted to have multiple references to an iterator and increment it. For example, this code is problematic because $itr is referenced multiple times:

```
proc report {itr} {
  puts "Instance: [dfm::get_data $itr -path_name]";
  dfm::inc itr;
  return;
}
proc cycle {itr} {
  puts "\nDepth:    [dfm::get_data $itr -depth]";
  while {$itr ne ""} {
    report $itr;
  }
  return;
}
set itr [dfm::get_placements $top];
cycle $itr;
```

and gives this error:

```
Error 186: Iterator has multiple reference, cannot perform dfm::inc
```

This in enforced to prevent the iterator from being incremented from different contexts independently.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When this command is used in the Query Server Tcl shell, ***svrf_object*** arguments do not apply.

Examples

```
# get all the cells in the database
set cells [dfm::get_cells]
while {$cells ne ""} {
# process each cell in the hierarchy
   ...
  dfm::inc cells
}
```

Related Topics

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::is_rev_frozen

Returns 1 if the current database revision is frozen and 0 otherwise.

Usage

dfm::is_rev_frozen

Arguments

None.

Return Values

Integer.

Examples

This shows interactive command execution. The current database revision is not frozen.

```
> dfm::is_rev_frozen
0
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::length

Returns the length of the current edge in user units.

Usage

dfm::length *iterator*

Arguments

- ***iterator***

  A required argument that specifies a geometry iterator containing edge data such as is created with dfm::get_geometries, dfm::get_flat_geometries, or dfm::get_net_shapes. The command reports the length for the *current object* for this iterator.

  This command results in an error when passed an iterator for accessing objects other than edge layers (type 2 layer).

Return Values

Floating-point number.

Examples

```
# process all shapes on derived edge layer
set geo_iter [dfm::get_geometries poly_less_0.2]
while { $geo_iter != ""} {
# compute length of current geometry
  set len [dfm::length $geo_iter]
# process the edges
  ...
  dfm::inc geo_iter
}
```

Related Topics

Layout Data Query Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_annotated_layers

Returns a list of layers having annotations that match the specified name and value pairs.

Usage

dfm::list_annotated_layers -annotation_name_value ‘**{{**’***name1 value1***‘**}**’
['{'*nameN valueN*'}']'**}**'… [-analyze_detail_layers_only]

Arguments

- **-annotation_name_value** ‘**{{**’***name1 value1*** ‘**}**’ [‘{’*nameM valueN* ‘}’]‘**}**’

  An argument set specifying the annotation name and value pairs of interest. The ***name value*** pairs are specified in a list of lists, with each pair being in its own list. The argument set must be specified as a Tcl list of lists.

- -analyze_detail_layers_only

  An optional keyword that specifies to check for the specified annotations only on the “_detail” layers generated by DFM Analyze.

Return Values

Tcl list.

Examples

```
# get list of layers having the LAYER_TYPE annotation with value DEV
set dev_layers [dfm::list_annotated_layers \
  -annotation_name_value {{LAYER_TYPE DEV}}]
```

Related Topics

Annotations in DFM Databases

Annotation Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_annotation_names

Returns a list of names of all annotations found on the specified layer or database, or both.

Usage

dfm::list_annotation_names [-db | -db_revision | -layer]

Arguments

- -db

  An optional argument returns the names of annotations defined for the entire database. This is the default.

- -db_revision

  An optional argument that returns the names of annotations defined for the open database revision.

- -layer

  An optional argument that returns the names of annotations defined for layers in the database.

Return Values

Tcl list.

Examples

```
# get list of annotation names on METAL1
set m1_annot [dfm::list_annotation_names -layer METAL1]
```

Related Topics

Annotations in DFM Databases

Annotation Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_annotation_values

Returns a list of the name and value pairs for annotations on the specified layer or database.

Usage

dfm::list_annotation_values {**-layer** ***layer_name*** | **-db** | **-db_revision**}
{**-annotation** ***name*** [*name* …] | **-all_annotations**}

Arguments

One of the following three arguments must be specified:

- **-layer *layer_name***

  An argument set that specifies to return only the annotation name and value pairs defined for the layer specified by ***layer_name***.

- **-db**

  An argument that specifies to return only the annotation name and value pairs defined for the current database.

- **-db_revision**

  An argument that specifies to return only the annotation name and value pairs defined for the current database revision.

  One of the following arguments must be specified:

- **-annotation *name*** [*name* …]

  An argument set that specifies to return only the name and value pairs for the specified annotation names.

- **-all_annotations**

  An argument set that specifies to return all relevant name and value pairs.

Return Values

Tcl list of lists.

Description

Returns annotation names and values from a layer, a database, or a database revision. If **-annotation** is specified, the list contains only the name and value pairs for the specified annotations. Otherwise, the list contains the name and value pairs for each of the annotations relevant to **-layer**, **-db**, or **-db_revision**.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Assume the database contains a layer named GaExt#rra#dfm_score#_e and this layer has seven annotations as shown:

Annotation Name	Annotation Value
DFM_RULE	GaExt
DFM_GROUP	Gate
DFM_GROUP	Extension
DFM_TYPE	rra
DFM_METRIC	dfm_score
DFM_LEVEL	error
DFM_BIN	""

The following command in an interactive shell returns a list of lists of name-value pairs:

```
> dfm::list_annotation_values -layer GaExt#rra#dfm_score#_e \
-all_annotations
{{DFM_RULE GaExt} {DFM_GROUP Gate Extension} {DFM_TYPE rra} {DFM_METRIC
dfm_score} {DFM_LEVEL error} {DFM_BIN ""}}
```

Related Topics

Annotations in DFM Databases

Annotation Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_annotation_values_for_layers

Returns a list of annotation names and values for specified lists of layers and annotation names. The values are returned per layer.

Usage

dfm::list_annotation_values_for_layers -layers *layer_list*
-annotation *annotation_name_list*

Arguments

- **-layers *layer_list***

  A required argument set that specifies a Tcl list of layer names.

- **-annotation *annotation_name_list***

  A required argument set that specifies a Tcl list of annotation names.

Return Values

Tcl list of lists.

Examples

In this example, both layers metal1 and metal2 have annotations of the name LAYER_TYPE with the value METAL, but only one of the layers has an annotation of the name DEV.

```
> dfm::list_annotation_values_for_layers -layers {metal1 metal2} \
-annotation {LAYER_TYPE DEV}
{DEV PIN} {LAYER_TYPE METAL METAL}
```

Related Topics

Annotations in DFM Databases

Annotation Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_annotation_values_for_name

Returns a list of values associated with the specified annotation.

Usage

dfm::list_annotation_values_for_name ***annotation_name***

Arguments

- ***annotation_name***

  A required argument that specifies the name of the annotation.

Return Values

Tcl list.

Examples

This shows a query in the interactive shell for values associated with the DEV annotation name:

```
> dfm::list_annotation_values_for_name DEV
PIN
```

Related Topics

Annotations in DFM Databases

Annotation Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_checks

Returns a list of rule checks that have layers in the DFM database.

Usage

dfm::list_checks [-layer *layer_name*]

Arguments

- -layer *layer_name*

  An optional argument set that specifies a rule check output layer. Only rule check names that output this layer are returned. By default, names of all rule checks having output to the DFM database are returned.

Return Values

Tcl list.

Examples

Example 1

This command lists all rule checks with database layers.

```
> dfm::list_checks
new_layer new_layer2
```

Example 2

Assume the rule file has this code:

```
metal1 {COPY metal1 }
checkname {DFM RDB metal1 m1.rdb CHECKNAME "rdbcheck" }
```

Here is layer-specific output:

```
> dfm::list_checks -layer metal1
metal1 rdbcheck
```

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_children

Returns a list of names of cells placed in the context cell.

Usage

dfm::list_children {***cell_iterator*** | ***cell_name***}

Arguments

- ***cell_iterator***

  An argument that specifies a cell iterator created with 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.

Return Values

Tcl list.

Examples

```
> dfm::list_children TOPCELL
CellA CellB
```

Related Topics

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_layers

Lists all the layers in the database and internally generated layer IDs. Other information related to the layers can be included. IDs can be excluded.

Usage

dfm::list_layers [-no_ids] [-op_text] [-check_text] [-check *check_name*] [-in_connect] [-in_lvs_db_layer]

Arguments

- -no_ids

  An optional argument that excludes layer IDs in the return value. A layer ID is an arbitrary number assigned to a layer by Calibre YieldServer. If this is specified with no other options, a simple list of layer names is returned.

- -op_text

  An optional argument that includes the text of the original rule file operation that generated each layer.

- -check_text

  An optional argument that includes check text comments for each layer, if any.

- -check *check_name*

  An optional argument that lists only the layers in the specified rule *check_name*.

- -in_connect

  An optional argument that lists layers that appear in Connect or Sconnect statements.

- -in_lvs_db_layer

  An optional argument that lists layers specified in LVS DB Layer statements.

Return Values

Tcl list of lists. The default form of a sub-list is {*layer_ID layer_name*}. The *layer_ID* is an internally assigned integer. The form of each sub-list varies with the specified options.

Examples

This shows an excerpt from an interactive shell session:

```
> dfm::list_layers
{20 new_layer} {21 new_layer2}
> dfm::list_layers -op_text
{20 new_layer {diff AND pwell}} {21 new_layer2 {COPY new_layer}}
```

Related Topics

Layer Management Commands

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_layout_netlist_options

Also used in the Query Server Tcl shell. Corresponding Query Server command: STATUS LAYOUT NETLIST.

Returns a Tcl list of dfm::set_layout_netlist option settings.

Usage

dfm::list_layout_netlist_options

Arguments

None.

Return Values

Tcl list of lists.

Examples

Assume this command is used:

```
dfm::set_layout_netlist_options \
  -filter [dfm::create_filter -device_ids 1 -magnify 3.14 -rotate -90 \
  -reflect_x -translate 123 456 -db_units]
```

This is the output of dfm::list_layout_netlist_options:

```
{annotated_devices NO}
{cell_prefix {}}
{comment_properties NO}
{device_location VERTEX}
{device_lowercase NO}
{device_prefix YES}
{device_templates NO}
{empty_cells NO}
{filter_devices 1}
{hier_separator /}
{hierarchy ALL}
{hierarchy_prefix YES}
{hspice_cr NO}
{hspice_user NO}
{instance_prefix YES}
{magnify 3.14}
{names {{LAYOUT NETS} {LAYOUT INSTANCES} {LAYOUT CELLS}}}
{pin_locations NO}
{primitive_device_subckts YES}
{reflect_x YES}
{rotate -90}
{seed_promoted_trivial_pins NO}
{separated_properties NO}
{string_properties YES}
{translate {123 456}}
{trivial_pins NO}
{type SPICE}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Netlist Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_original_layers

Returns a Tcl list of original layers in the database.

Usage

dfm::list_original_layers [-gds_info]

Arguments

- -gds_info

  Optional argument that returns the layer number(s) in addition to the layer names in a list of lists. If a layer number is the target of a Layer Map statement, then the mapping constraints are shown instead of the target layer number.

Return Values

By default, a list of layer names is output. If -gds_info is specified, a Tcl list of lists is output like this:

```
{layer_name layer_number [layer_number ...]} ...
```

If a layer number is the target of a Layer Map statement, then the *layer_number* is itself a list of lists containing sub-lists with the layer number and datatype mapping constraints as shown in the Examples section.

Examples

If the rule file contains this:

```
LAYER MAP > 2 < 15 DATATYPE > 33 < 50 100
LAYER MAP > 30 < 50 DATATYPE > 3 < 23 100
LAYER TEST1 100 15 // layer 15 is not a mapped layer
LAYER TEST2 20
```

The dfm::list_original_layers -gds_info command returns:

```
{TEST1 15 {{> 2 < 15} {> 33 < 50}} {{> 30 < 50} {> 3 < 23}}} {TEST2 20}
```

Related Topics

Layer Management Commands

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_properties

Lists all property names defined on a layer or geometry.

Usage

dfm::list_properties {***layer_name*** | ***iterator***} [-min_max | -show_prop_target]

Arguments

- ***layer_name***

  An argument that specifies a layer name. The command reports the properties on this layer. May not be specified with ***iterator***.

- ***iterator***

  An argument specifying a layer or geometry iterator. The command reports the properties on the current object for this iterator. May not be specified with ***layer_name***.

- -min_max

  Optional argument that returns the minimum and maximum values for numerical properties, along with the property target type.

- -show_prop_target

  Optional argument that shows the target of the property when DFM Property originally assigned the property. The target can be POLY_PROPERTY (shape property), NET_PROPERTY, or CELL_PROPERTY.

Return Values

Tcl list.

Examples

Example 1

```
> dfm::list_properties metal1c -min_max
{AREA POLY_PROPERTY 40.0 415.5}
# property name is AREA, originally assigned to polygons,
# with minimum being 40 and maximum being 425.5 square microns
```

Example 2

See “Example 2” under dfm::get_geometries.

Related Topics

DFM Property Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::list_revs

Returns a list of revisions for a DFM database.

Usage

dfm::list_revs [-db *db_name*] [-name]

Arguments

- -db *db_name*

  Optional argument that specifies the path to a different DFM database than the currently loaded one. By default, the currently loaded one is queried.

- -name

  Optional argument that displays the names of the revisions, if they exist. By default, numbers are returned.

Return Values

Tcl list.

Examples

```
> dfm::list_revs
0
# current database only has the initial version
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::move_layer

Copies the contents of a layer to an existing target layer, then deletes the copied layer.

Usage

dfm::move_layer -from *layer1* -to *layer2*

Arguments

- **-from *layer1***

  A required argument that specifies the copied layer.

- **-to *layer2***

  A required argument that specifies the destination layer. This layer must already exist in the database and must be of the same type and configuration as ***layer1***.

Return Values

None.

Description

Copies the contents of ***layer1*** to ***layer2***, including any node numbers or properties, then deletes ***layer1*** if the copy is successful. Any properties already attached to ***layer2*** are preserved.

This command causes an error if ***layer1*** and ***layer2*** are not of an identical type and configuration, or the specified layers are not in the database. Layer type and configuration can be queried using dfm::get_data.

The. dfm::copy_layer command is related to dfm::move_layer and copies the contents of one layer to another layer that does not currently exist.

Examples

```
# copies the contents of l1 to l2 and deletes l1 upon a successful copy
dfm::move_layer l1 l2
```

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::net_is_epin

Returns 1 if a cell's net is connected farther up in the hierarchy and 0 otherwise.

Usage

dfm::net_is_epin *node_ID* -cell *cell_name*

Arguments

- ***node_ID***

  A required argument that specifies a net number assigned by the connectivity extractor. Node IDs can be queried using dfm::get_data -node, dfm::get_device_data -net, or dfm::get_port_data -node.

- **-cell *cell_name***

  A required argument that specifies a cell name. The corresponding cell should contain the net associated with ***node_ID***.

Return Values

Integer.

Examples

This example reports nets that are external pins by cell.

```
# report net IDs and names in all cells for nets that are epins
set cells [dfm::get_cells]
while {$cells ne ""} {
  set nets [dfm::get_nets $cells]
  while {$nets ne ""} {
    set node_ID [dfm::get_data $nets -node]
    if { [dfm::net_is_epin $node_ID -cell \
         [dfm::get_data $cells -cell_name]] } {
         puts "CELL: [dfm::get_data $cells -cell_name] \
PIN NET: [dfm::get_net_name $node_ID -cell $cells -netlist] id $node_ID"
    }
    dfm::inc nets
  }
  dfm::inc cells
}
```

Related Topics

Connectivity Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::new_layer

Creates a new layer as the result of processing a rule script. SVRF and TVF syntax are supported for the rules.

Usage

dfm::new_layer {**-svrf** ***svrf_cmds*** | **-tvf** ***tvf_cmds***} [-keep_all_layers | -dfm | -drc]
[-comments *comments_string*]
[-keep_analyze_inputs]
[-keep_measure_inputs]
[-keep_property_inputs]
[-keep_all_inputs]
[-list_checks]
[-rdbs_as_files | -rdbs_as_layers]
[-make_nodal] [-overwritable]

Arguments

- **-svrf *svrf_cmds***

  A required argument set defining rule file code that generates a new layer. The new layer can be created in the global scope or within the scope of a rule check. The ***svrf_cmds*** argument is a Tcl list containing a valid SVRF script. Not compatible with the **-tvf** option.

  Rule file elements that set up the Calibre hierarchical database (CALIBRE LAYOUT DATA INPUT MODULE stage in the transcript) are generally ignored. Specification statements that set up drawn layers or that process the layout before the DRC or DFM executive modules fall into this category.

  [Not] Inside Cell layer operations cause errors.

  The Polygon and Layout Polygon statements cause errors. DFM Create Layer Polygons can be used instead.

  Statements or operations other than those used by the DRC or DFM executive modules are generally ignored; however, using Device causes an error. Pathchk can cause errors if LVS supply names were not declared in the rule file that generated the current DFM database.

  Connect and Sconnect are supported as described later.

  Other specifics about rule file element behaviors are covered under the -keep_all_layers, -drc, and -dfm option descriptions.

- **-tvf *tvf_cmds***

  A required argument set defining compile-time TVF rule file code that generates a new layer. Not compatible with the **-svrf** option.

  The semantics of the ***tvf_cmds*** argument are essentially the same as for ***svrf_cmds***, except that compile-time TVF syntax is used instead of SVRF syntax. The ***tvf_cmds*** argument cannot contain the "#!tvf" statement or any of the tvf:: namespace counterparts to elements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

that cannot be used in ***svrf_cmds***. See the "Compile-Time TVF" section of the *SVRF Manual* for details about TVF elements.

- -keep_all_layers | -dfm | -drc

  An optional argument that specifies the processing performed for the rules. SVRF syntax is shown, but the TVF equivalents are also supported.

  - -keep_all_layers (default) — Uses a processing behavior unique to dfm::new_layer, as follows:

    - All layer operations in the rule file are executed, regardless of any statements in the DRC or DFM [Un]Select Check families.
    - DFM RDB operations not processed (because all layers are kept anyway) unless the -rdbs_as_files option is also present.
    - All layers generated by the rule script are kept after execution of that script, except for implicit (TMP<n>) layers and encrypted layers.
    - RDB outputs from DFM Analyze and DFM Measure operations are kept after execution of the rule script unless the -rdbs_as_files option is also present.
    - All layers are configured with node numbers if connectivity can be passed to the layer.

  - -dfm — Causes the command to behave like "calibre -dfm" as follows:

    - Checks are executed as with calibre -dfm: DFM Select Check and DFM Unselect Check statements are respected.
    - RDB outputs from DFM Analyze and DFM Measure operations are converted to new layers unless the -rdbs_as_files option is also present.
    - DFM RDB operations cause their input layers to be kept after execution of the rule script unless the -rdbs_as_files option is also present.
    - Layers created by output operations in checks are kept after execution of the rule script.
    - Stand-alone Copy operations in rule checks cause their input layers to be kept after execution of the rule script.
    - Input layers to DFM Analyze, DFM Measure, and DFM Property operations are not kept (this is done with the -keep_*_inputs options).
    - Layers are configured with node numbers as required by their use as inputs to nodal operations, or as specified by DFM Select Check NODAL statements.

  - -drc — Causes the command to behave like "calibre -drc" as follows:

    - Checks are executed as with calibre -drc: If there are DRC [Un]Select Check statements, they are treated as with calibre -drc.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

    - RDB outputs from DFM Analyze, DFM Measure, and DFM RDB operations are saved to files unless the -rdbs_as_layers option is also present.
    - Input layers to DFM Analyze, DFM Measure, and DFM Property operations are not kept (this is done with the -keep_*_inputs options).
    - Layers are configured with node numbers as required by their use as inputs to nodal operations.
    - The DRC Results Database statement and the -rdbs_as_layers option are handled as follows:

      If DRC Results Database statement is used, it is observed along with any DRC Check Map statements when -rdbs_as_layers is not specified. Layers from output operations are not generated by dfm::new_layer since they are saved in the specified results databases.

      If -rdbs_as_layers is specified, neither DRC Results Database nor DRC Check Map statements are used, and layers from output operations are generated by dfm::new_layer.

- -comments "*comments_string*"

  An optional argument set that assigns *comment_string* as a property of a generated layer. The argument must be a Tcl string and should be enclosed in double quotation marks (" "). Be aware that the delimiters you use to enclose the *comments_string* can have an impact on how the string can be used in a future analysis run.

- -keep_analyze_inputs

  An optional argument that specifies the input layers to DFM Analyze operations in the rule script are retained in memory after the script completes.

- -keep_measure_inputs

  An optional argument that specifies the input layers to DFM Measure operations in the rule script are retained in memory after the script completes.

- -keep_property_inputs

  An optional argument that specifies the input layers to DFM Property operations in the rule script are retained in memory after the script completes.

- -keep_all_inputs

  An optional argument that specifies the input layers to all DFM operations in the rule script are retained in memory after the script completes.

- -list_checks

  An optional argument used with the -dfm or -drc option that returns a list of rule checks executed by the dfm::new_layer command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -rdbs_as_files

  An optional argument that specifies the RDB options to DFM Analyze and DFM Measure, and DFM RDB operations, should write RDBs rather than creating layers. May not be specified with -rdbs_as_layers.

- -rdbs_as_layers

  An optional argument that specifies the RDB options to DFM Analyze and DFM Measure, DFM RDB operations, and layer operations processed in -drc mode should return layers rather than writing the layers to RDB files. When -keep_all_layers or -dfm is also specified, -rdbs_as_layers has no effect except to suppress warning messages about RDBs being saved as layers.

- -make_nodal

  An optional argument that specifies to assign node numbers to generated layers whenever connectivity can be passed to that layer.

- -overwritable

  An optional argument that specifies to create new layers that can be overwritten by future dfm::new_layer commands in the run. By default, new layers cannot be overwritten, and any attempts to create a layer with a name that already exists results in an error.

Return Values

For the -list_checks option, a Tcl list. Otherwise no values are returned.

Description

Executes an SVRF or TVF script that generates layer output. As layers are created by the operations in the script, they are added to the hierarchical database (HDB) in memory. Any layers that exist in the DFM database may be referenced in the rule script.

Because this command appears in a Tcl script, the SVRF or TVF code is only processed by the rule compiler after the dfm::new_layer command is executed. The Tcl pre-processor does not parse SVRF or TVF syntax; hence, it is best to test your SVRF or TVF code against the compiler using calibre -svrf or some other method prior to including such code in dfm::new_layer commands. This helps to avoid rule file compiler errors during a YieldServer run.

Depending on the options to dfm::new_layer, and on the way the layers are used in the rules, new layers created by the rule script operations are either deleted from the HDB in memory before dfm::new_layer completes or are maintained. The Arguments section discusses these circumstances in detail. Note that TMP<n> layers shown in the transcript and encrypted layers are never kept.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Layers generated from rule checks or layer derivations in a rule file script (those having "::" in them in a transcript when a layer operation is performed) must have their names quoted within an SVRF or TVF script. For example, the Calibre transcript may show this for a derived layer:

```
check.prop::INT_POLY = INT [poly] < 4.2
```

If the check.prop::INT layer is referenced in a **-svrf** or **-tvf** block, the layer name must be quoted. These layers might not be preserved in memory depending on which dfm::new_layer options are used.

Layer operations that require connectivity on input layers may only be used if a connectivity model exists in the database. The DFM Database CONNECT keyword ensures this in a calibre -dfm run. If connectivity is not present in the current DFM database, then Connect and Sconnect statements may be used in the ***svrf_cmds*** script. The connectivity statements in the dfm::new_layer rule script return an error if a connectivity model already exists. If a layer participates in a Connect statement executed by dfm::new_layer, all outstanding Tcl objects (such as Tcl iterators) in which the layer participates are invalidated. To preserve any Connect changes, you must run dfm::save_rev.

Layer operations in the rule script can access variables defined in previous dfm::new_layer commands in the same YieldServer session. You can also access rule file values from Variable statements in the original -dfm run or from previous dfm::new_layer commands in the same YieldServer session. You can define new Variable statements in the script to create new variables, but you cannot reset existing variables.

By default, an empty original layer referenced by dfm::new_layer causes a warning, and an empty layer is created. To change the severity to an error, you can use the dfm::set_new_layer_error_severity command.

Examples

Example 1

Assuming that A, B, and C are existing layers, this example executes the DFM Property operation and creates the new layer X, which is kept in memory after it is generated.

```
dfm::new_layer -svrf {X = DFM PROPERTY A B C [RATIO = AREA(B)/AREA(C)]}
```

Example 2

This shows how to output connectivity layers by net name to a layout database. The layout nets from which polygons are exported correspond to source nets. In this case, OASIS is used, but GDS is also an option with dfm::write_gds. Connectivity and device information must exist in the DFM database.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# source nets for export
set mynets {PWR GND}

# determine the source netlist name from the rule file and run LVS
# with that source against the DFM database layout
set source_path [dfm::get_svrf_data \
  [dfm::create_svrf_analyzer] -specification "source path"]
dfm::run_compare -source_netlist $source_path -auto

# find layout net names corresponding to the source names
foreach net $mynets {
  lappend my_layout_nets [dfm::get_layout_name $net -net]
}

# set up layer operations to export net shapes
set svrf ""
set out_layers [list]
foreach metal {metal1 metal2} {
  set metal_out ${metal}_out
  append svrf "$metal_out = NET $metal $my_layout_nets \n"
  lappend out_layers $metal_out
}
# create the layers
dfm::new_layer -svrf $svrf -keep_all_layers

# export net polygons to OASIS. begin with layer number 1.
set layer_info ""
set layer_nbr 1
foreach layer $out_layers {
  append layer_info "-layer_info \{$layer $layer_nbr 0\} "
  incr layer_nbr
}
eval "dfm::write_oas $layer_info -file out.oas"
```

Example 3

Assuming that M1 is an existing layer, this example does the following:

- Executes the analyze check.
- Keeps the RDB layer from the DFM Analyze operation in memory and saves it to the DFM database rather than writing it to an RDB (-dfm is used).
- Keeps layer MET1 in memory after the script completes (-keep_analyze_inputs option is used).
- If M1 is nodal, MET1 will also be nodal because the -make_nodal option is present.
- Attaches the comment “YieldServer generated analyze layer” to the ANALYZE RDB layer and MET1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
dfm::new_layer -dfm -keep_analyze_inputs -make_nodal -svrf {
    analyze {
        MET1 = COPY M1
        DFM ANALYZE MET1 [~~COUNT(MET1)] > 0 WINDOW 100
        RDB ONLY result.rdb
    }
    DFM SELECT CHECK "analyze"
} -comments {YieldServer generated analyze layer}
```

Example 4

```
# create a new layer using DRC mode and save the results to ys_drc_db
# INT_POLY is not generated in the DFM database because -drc is used
# without -rdbs_as_layers.
dfm::new_layer -drc -svrf {
   check.prop{
      INT_POLY = INT [POLY] < 4.2
      COPY INT_POLY
    }
  DRC SELECT CHECK check.prop
  DRC RESULTS DATABASE ys_drc_db
 }
```

Example 5

The -list_check option returns a list of rule checks executed by the dfm::new_layer command.

```
# the "checks" variable contains the "TEST_CHECK" rule check name
set checks [dfm::new_layer -list_checks -dfm -svrf {
                          DFM SELECT CHECK TEST_CHECK
                          TEST_CHECK {
                            y = DFM PROPERTY M1 [A = AREA(M1)]
                            COPY y
                          }
           }]
```

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::open_db

Opens a DFM database. Only one database can be open at the same time.

Usage

dfm::open_db *db_name* [-rev *revision_name*]

Arguments

- ***db_name***

  The pathname of the DFM database to be opened.

- -rev *revision_name*

  An optional argument set specifying the revision of the database to open. When not specified, Calibre YieldServer opens the default revision. If you have not explicitly defined the default revision using the dfm::set_default_rev command, the default revision is "0".

  > **Note**
  > Revision 0 is referred to as the "master" revision, and it is reserved as the initial version of a DFM database. At the end of any calibre -dfm batch run, the database created is assigned the revision number 0.

Return Values

None.

Examples

This example shows a common sequence of database modification commands.

```
> dfm::open_db dfmdb
Opening database "dfmdb", revision "master"
> dfm::create_rev new

Creating revision new

Saving revision new
...
# perform some changes
...
> dfm::save_rev
> dfm::set_default_rev [dfm::get_current_rev]
> dfm::close_db
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::open_rev

Opens a specified revision of the current database.

Usage

dfm::open_rev *revision*

Arguments

- ***revision***

  A required argument specifying the revision of the database to open. The ***revision*** may be a name or number of an existing revision.

Return Values

None.

Description

Before opening the specified revision, Calibre YieldServer closes any revision currently open, if it is unmodified. If the revision that is currently open has been modified, it must first be frozen using dfm::freeze_rev, saved using dfm::save_rev, or closed using dfm::close_db -force.

Revision 0 is referred to as the “master” revision, and it is reserved as the initial version of a DFM database.

Examples

Example 1

```
dfm::open_db dbname
# Opens the default revision of database dbname

dfm::open_rev 1
# Opens the revision 1, if it exists and closes the default revision
# that was opened by dfm::open_db.
```

Example 2

```
dfm::open_db dbname
# Opens dbname revision 0

dfm::create_rev
# Creates a new revision 1 off of revision 0

dfm::open_rev 1
# opens revision 1

dfm::add_annotation -db_revision -annotation rev -value 1.0

dfm::open_rev 2
Error 26: Need to save changes before executing command dfm::open_rev.
# need to save changed in rev 1 before opening rev 2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::perimeter

Returns the perimeter of the current polygon or cell.

Usage

dfm::perimeter ***iterator***

Arguments

- ***iterator***

  A required argument specifying a polygon iterator, such as from dfm::get_device_geometries, dfm::get_flat_geometries, dfm::get_geometries, or dfm::get_net_shapes, or a cell iterator from dfm::get_cells. The command reports the perimeter for the current object for this iterator. This command returns an error when passed an iterator for accessing objects other than polygons or cells.

Return Values

Floating-point number.

Description

For a polygon, this command calculates the perimeter. For a cell, this command calculates the perimeter of the bounding box of objects in the cell. Measurements are in user units.

Examples

```
# process M1 shapes
set geo_iter [dfm::get_geometries M1]
while { $geo_iter != ""} {
# compute perimeter of current geometry
    set perimeter [dfm::perimeter $geo_iter]
# process the perimeters
    ...

    dfm::inc geo_iter
}
```

Related Topics

Layout Data Query Commands

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::print_layers

Prints the internally-generated numbers and names of all layers in the database to STDOUT. Internally generated layer numbers are not necessarily the same as the layer numbers defined in rule file Layer statements.

Usage

dfm::print_layers [-op_text]

Arguments

- -op_text

  An optional argument that specifies to write the text of original rule file operation that generated the layer to STDOUT after each layer name.

Return Values

String. By default, the string contains an internally generated numeric layer ID followed by a layer name. If -op_text is specified, the layer operation that generated the layer is output in parentheses.

Examples

```
> dfm::print_layers -op_text
20 new_layer (diff AND pwell)
21 new_layer2 (COPY new_layer)
```

Related Topics

Layer Management Commands

Layout Data Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::read_netlist

Also used in the Query Server Tcl shell.

Returns a SPICE netlist object.

Usage

dfm::read_netlist {**-layout** | **-source**} [{-rules *filename*} | {-perc *database* [-top_cell *name*]}] [-netlist *filename*] [-netlist_transforms *transform_list*] [-hcell *filename*]

Arguments

- **-layout**

  An argument that specifies to read the Layout Path netlist. Either this argument or **-source** must be specified. This assumes the Layout System is SPICE.

- **-source**

  An argument that specifies to read the Source Path netlist. Either this argument or **-layout** must be specified.

- -rules *filename*

  An argument set that specifies the SPICE netlist path is defined in a rule file indicated by the *filename*. Either the Layout Path or Source Path netlist is read, depending on the **-layout** or **-source** option. This argument may not be used when a Mask SVDB Directory or DFM Database is loaded.

- -perc *database*

  An argument set that specifies the SPICE netlist is read from either a Mask SVDB Directory or DFM database generated by Calibre PERC. The *database* argument specifies the directory to access. This argument set may not be specified with -rules or when an SVDB or DFM database is already loaded.

- -top_cell *name*

  An optional argument set used with -perc that specifies a primary cell name. This argument set is only necessary when the results of more than one design exist in the *database*. The netlist corresponding to the *cell_name* is read from the *database*.

- -netlist *filename*

  An optional argument set that specifies the SPICE netlist to read. This argument set overrides any other setting when reading the netlist.

- -netlist_transforms *transform_list*

  An optional argument set that specifies the netlist transforms to apply to the netlist as it is read. The *transform_list* is a non-empty Tcl list consisting of these keywords:

  deep_shorts — Deep shorts are resolved. This is done by default.

  high_shorts — High shorts are resolved. This is done by default.

  trivial_pins — Trivial pins are removed. This is done by default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reduced — Device reduction is performed according to the rule file. This should be specified with the other transform options if you want a netlist similar to what LVS uses internally.

none — No transformations are performed. This keyword may not be specified with another.

- -hcell *filename*

  An optional argument set that specifies a file containing an hcell list. When specified, the -hcell option only takes effect when the -netlist_transforms reduced option is used.

Return Values

Netlist object.

Description

Returns a SPICE netlist object. The referenced netlist depends on which options are specified. If none of the optional argument sets are used, the netlist is read from the rule file referenced in the currently loaded DFM database or Mask SVDB Directory.

The commands that take a netlist object as input are dfm::get_cells, dfm::set_netlist_options, and dfm::write_spice_netlist.

The netlist object created by this command should be closed with dfm::close_netlist before another netlist object is created.

Consider the following source netlist:

```
.subckt cell1 1 2
r1 1 2
r2 1 2
.ends

.subckt cell2 1 2
x1 1 2 cell1
.ends

.subckt top
x1 1 2 cell2
x2 3 4 cell2
c1 1 3
c2 2 4
.ends
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A “dfm::read_netlist -source -rules rules” command produces this output, which is topologically equivalent to the source:

```
.SUBCKT cell1 1 2
Rr1 1 2
Rr2 1 2
.ENDS

.SUBCKT cell2 1 2
Xx1 1 2 cell1
.ENDS

.SUBCKT top
Xx1 1 2 cell2
Xx2 3 4 cell2
Cc1 1 3
Cc2 2 4
.ENDS
```

A “dfm::read_netlist -source -rules rules -netlist_transforms { reduced }” command produces the same output as the previous case, except the resistors in cell1 are reduced:

```
.SUBCKT cell1 1 2
Rr2 1 2
.ENDS
```

Assuming an *hcell_list* file containing “cell1 cell1”, a “dfm::read_netlist -source -rules rules -hcell hcell_list -netlist_transforms { reduced }” command causes cell2 to be flattened because it is not an hcell:

```
.SUBCKT cell1 1 2
Rr2 1 2
.ENDS

.SUBCKT top
Xx1/x1 1 2 cell1
Xx2/x1 3 4 cell1
Cc1 1 3
Cc2 2 4
.ENDS
```

Executing the same command but with an empty hcell list causes the entire design to be flattened:

```
.SUBCKT top
Rx1/x1/r2 1 2
Rx2/x1/r2 3 4
Cc1 1 3
Cc2 2 4
.ENDS
```

If this command is not used before dfm::write_spice_netlist, the output of dfm::write_spice_netlist is based upon the physical design in a currently loaded DFM database

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

or SVDB and is similar to an extracted netlist produced by calibre -spice or the Calibre Connectivity Interface (CCI).

This command must be used with all transformation options specified in the *transform_list* when producing alternative output for LVS Write Layout Netlist or LVS Write Source Netlist.

Examples

Example 1

This example shows a command sequence for writing a transformed source netlist in an LVS flow. No database is loaded when using this code sequence.

```
# read the source netlist from the rules file; use LVS transforms
set source_handle [dfm::read_netlist -source -rules rules \
  -netlist_transforms {deep_shorts high_shorts trivial_pins reduced}]
# write the transformed netlist
dfm::write_spice_netlist src_xform.sp -netlist_handle $source_handle
# close the netlist object
dfm::close_netlist
```

Example 2

This example shows a script for gathering netlist cells from a Calibre PERC SVDB.

```
# Load a netlist
set nl [dfm::read_netlist -perc svdb]

# Iterate over all cells in the netlist
puts ""
puts "PRINT ALL CELLS IN THE NETLIST"
set c [dfm::get_cells $nl]
while {$c ne ""} {
   puts [dfm::get_data $c -cell_name]
   dfm::inc c
}
```

Related Topics

Netlist Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::reset_timer

Also used in the Query Server Tcl shell.

Restarts a timer object.

Usage

dfm::reset_timer *timer_object*

Arguments

- ***timer_object***

  A required Tcl object created with dfm::create_timer. The timer object is re-initialized when dfm::reset_timer is executed.

Return Values

None.

Related Topics

Timer Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::reset_transform

Resets the transformation on an iterator. Subsequent attempts to obtain vertices or extents from the iterator return coordinates in the cell context.

Usage

dfm::reset_transform *iterator*

Arguments

- ***iterator***

  A required argument supplying geometry or placement iterator that has a transform applied to it by dfm::apply_transform.

Return Values

None.

Related Topics

Iterator Processing Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::run_compare

Executes an LVS-H comparison between a source netlist file and the layout represented by the DFM database.

Usage

dfm::run_compare [-source_netlist *filename*] [-source_primary *primary_cell*] [-hcell *hcell_filename*] [-auto] [-flatten] [-use_cmdline_options]

Arguments

- -source_netlist *filename*

  An optional argument set that specifies the source SPICE netlist. By default, the source netlist is taken from the rule file Source Path statement that was used when generating the DFM database.

- -source_primary *primary_cell*

  An optional argument set that specifies the primary cell name of the source. By default, it is taken from the Source Primary statement that was when generating the DFM database. This option does not apply when -flatten is used.

- -hcell *hcell_filename*

  An optional argument that specifies a file containing an hcell list. This is a similar behavior to the LVS-H -hcell command line option. This option does not apply when -flatten is used.

- -auto

  An optional argument that specifies to automatically use matched cell names between layout and source as hcells. This is a similar behavior to the LVS-H -automatch command line option. This option does not apply when -flatten is used.

- -flatten

  An optional argument that runs the comparison flat. This is a similar behavior to the LVS -flatten command line option. The top-level cell names in layout and source must match when this option is used.

- -use_cmdline_options

  An optional argument that specifies to use the -automatch, -flatten, or -hcell command line settings (if any) that were used to generate the DFM database. This option is intended for LVS and Calibre PERC LDL source-based flows.

Return Values

None.

Description

In order to run an LVS comparison using this command, a DFM database must be loaded and must have both connectivity and device layers present. Using the DFM Database DEVICES and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

CONNECT keywords in the rule file is a prerequisite. The settings from the rule file referenced in the DFM database are used unless overridden by command arguments.

Explicit specification of -auto, -hcell, or -flatten in this command overrides a null specification of the corresponding command line option referenced by -use_cmdline_options. A local -hcell specification may not be used if -use_cmdline_options is specified and -hcell was used to generate the DFM database.

This command does not work if you change the name of a cell using dfm::set_layout_netlist_options. This command creates a new revision if the current open revision is frozen.

If this command is successful, a cross-reference database (XDB) is built as part of the current DFM database revision. Note that deleting the connectivity model using dfm::disconnect also deletes the XDB.

Note

This command does not support the LVS Push Devices SEPARATE PROPERTIES (PDSP) flow.

To write the comparison report, use dfm::write_cmp_report.

Examples

This example runs an LVS comparison on a DFM database design and then exports layout metal polygons that correspond to PWR and GND nets in the source.

```
# source nets for export
set mynets {PWR GND}

# determine the source netlist name from the rule file and run LVS
# with that source against the DFM database layout
set source_path [dfm::get_svrf_data \
  [dfm::create_svrf_analyzer] -specification "source path"]
dfm::run_compare -source_netlist $source_path -auto

# find layout net names corresponding to the source names
foreach net $mynets {
    lappend my_layout_nets [dfm::get_layout_name $net -net]
}

# set up layer operations to export net shapes
set svrf ""
set out_layers [list]
foreach metal {metal1 metal2} {
    set metal_out ${metal}_out
    append svrf "$metal_out = NET $metal $my_layout_nets \n"
    lappend out_layers $metal_out
}
# create the layers
dfm::new_layer -svrf $svrf -keep_all_layers
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# export net polygons to OASIS. begin with layer number 1.
set layer_info ""
set layer_nbr 1
foreach layer $out_layers {
    append layer_info "-layer_info \{$layer $layer_nbr 0\} "
    incr gds_nbr
}

eval "dfm::write_oas $layer_info -file out.oas"
```

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::save_rev

Also used in the Query Server Tcl shell.

Saves the current database revision.

Usage

dfm::save_rev

Arguments

None.

Return Values

None.

Description

Saves the modifications since the last save of the database. This command saves the all of the new layers created in memory, their properties and annotations, and any database annotations.

The following table shows the revision states and save behaviors.

Table 3-22. dfm::save_rev

State	Command Action	Comment
Current open revision is frozen and has no changes.	Save not allowed.	
Current open revision is frozen and has unsaved changes.	Creates a new revision using the usual naming convention and saves it.	New version becomes the current open revision.
Current open revision is not frozen and has unsaved changes.	Saves in the same revision.	New version becomes the current open revision.
Current open revision is not frozen and has no changes.	No saved changes.	Warning 2: No database changes. Nothing to do.

To save the modifications as a new revision, you must precede dfm::save_rev by a dfm::create_rev command. If a frozen revision is opened and dfm::save_rev is executed before dfm::create_rev is executed, then Calibre YieldServer creates a new revision using the usual naming convention. Note that after being saved, the new revision is in the unfrozen state.

Examples

This example shows a typical sequence of database administrative commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> dfm::open_db dfmdb
Opening database "dfmdb", revision "master"
> dfm::create_rev new

Creating revision new

Saving revision new
...
# perform some changes
...
> dfm::save_rev
> dfm::set_default_rev [dfm::get_current_rev]
> dfm::close_db
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::set_default_rev

Defines the revision of the database to be opened by default.

Usage

dfm::set_default_rev *revision*

Arguments

- ***revision***

  A positive integer or name that sets the default revision of the database to open. The specified ***revision*** must be a frozen revision. See the "DFM Database Revision State Diagram."

Return Values

None.

Description

Marks the specified revision as the default revision to be opened the next time the command dfm::open_db is issued without the -rev option.

Examples

This example shows a typical sequence of database administrative commands.

```
> dfm::open_db dfmdb
Opening database "dfmdb", revision "master"
> dfm::create_rev new

Creating revision new

Saving revision new
...
# perform some changes
...
> dfm::save_rev
> dfm::set_default_rev [dfm::get_current_rev]
> dfm::close_db
```

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::set_layout_netlist_options

Also used in the Query Server Tcl Shell. This command configures dfm::write_spice_netlist output in a similar way as numerous Calibre Connectivity Interface (CCI) Customized Layout SPICE Netlist Commands are used to configure LAYOUT NETLIST WRITE output.

Customizes behavior of the dfm::write_spice_netlist command for writing a layout netlist.

Usage

dfm::set_layout_netlist_options ***options***

Arguments

The ***options*** are any of the following. At least one is required when the command is used:

- -annotated_devices {NO | YES}

  Specifies whether annotated devices are used for writing device properties. Default is NO. The YES option may not be used with -separated_properties YES.

- -cell_prefix *prefix*

  Specifies to add the *prefix* before all subcircuit names in the netlist. This applies to subcircuit calls and .SUBCIRCUIT statements.

- -no_cell_prefix

  Specifies to override the current -cell_prefix setting such that no prefix is used.

- -comment_properties {NO | YES}

  Specifies whether all non-standard (not standard in regular SPICE syntax) properties appear in comment-coded format. Default is NO.

- -device_location {VERTEX | CENTER}

  Specifies the reference point for device locations. The netlist may be generated with $X= $Y= comments to indicate either a vertex or the center of the device seed shape. By default, the layout netlist is written with vertex coordinates. If LVS Center Device Location YES is specified in the rule file, CENTER must be used.

- -device_lowercase {NO | YES}

  Specifies whether all device element names, model names, and pin names use lowercase letters, regardless of the LVS Downcase Device setting in the rule file. Default is NO.

- -device_prefix {YES | NO}

  Specifies whether a prefix is written before all device names. The prefix applies when -hierarchy ALL is not used. Default is YES.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -device_templates {<u>NO</u> | YES}

  Specifies whether *.DEVTMPLT data is written to the netlist. Default is NO. The YES option is useful in pushdown separated properties (PDSP) flows and follows the -pin_locations YES format.

- -empty_cells {<u>NO</u> | YES}

  Specifies whether to write calls to empty cells. The netlist may be generated with calls to cells that do not contain devices (for example, via cells). Default is NO.

- -filter *filter_object*

  Specifies to filter the netlist output based upon device types in a dfm::create_filter *filter_object*. Any transformations defined in the filter object are also applied to device coordinates, but device properties like length and width are unaffected by filter transformations.

- -hierarchy {<u>ALL</u> | FLAT | AGF | HCELL}

  Specifies the output netlist hierarchy. The keyword options are these:

  - ALL — No cells are expanded, which is the default.
  - FLAT — Netlist is flattened to the top level.
  - AGF — Netlist is consistent with the default output of the AGF WRITE command in Calibre Query Server.
  - HCELL — All non-hcells are expanded into parent cells, including unbalanced hcells that are expanded during comparison. This option is unaffected by -names SOURCE.

- -hierarchy_expand_cell *cell_name*

  Specifies whether the cell having the *cell_name* is expanded one level in the output netlist. The asterisk (*) wildcard may be used in the *cell_name* parameter to match zero or more characters.

- -hierarchy_prefix {<u>YES</u> | NO}

  Specifies whether to write the prefix for expanded names as specified by the -hierarchy option. Default is YES.

- -hier_separator "*separator_char*"

  Specifies the hierarchy separator character for the layout netlist. The *separator_char* must be a permitted character in SPICE. The default is "/".

- -hspice_cr {<u>NO</u> | YES}

  Specifies whether the model names in capacitor and resistor elements use the standard HSPICE model name field instead of the default comment-coded format. Default is NO.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -hspice_user {NO | YES}

  Specifies whether all model names for user-defined devices use model names as subcircuit reference names in addition to the default comment-coded format. Default is NO.

- -instance_prefix {YES | NO}

  Specifies whether a prefix is written before all instance names. The prefix applies when -hierarchy ALL is not used. Default is YES.

- -names {{LAYOUT | SOURCE | NONE} {[NETS INSTANCES] | [NETS] | [INSTANCES] | [ALL]}}

  Specifies to output certain names. LAYOUT selects layout names and is the default if -names is not used. SOURCE selects corresponding source names. NONE specifies not to output names. NETS selects only net names for output. INSTANCES selects only instance names for output. The default is both NETS and INSTANCES. ALL is the same as the default behavior when LAYOUT is specified. When SOURCE is specified with ALL, in addition to selecting corresponding source net and instance names, source cell (subcircuit) names are selected for output.

- -pin_locations {NO | YES}

  Specifies whether to provide pin locations for all devices. Pin location information includes a device template for each device type that shows the ordering of pins and a $PIN_XY comment after each device instance showing the coordinates of the pins. Pin locations are written only when the PINLOC keyword is used in the DFM Database specification statement. Default is NO.

- -port_pads {NO | YES}

  Specifies whether to write comments containing port object details in subcircuit definitions of the output netlist. Ports are declared in the rules using Port Layer Text or Port Layer Polygon. When YES is specified, comments like this are included in the netlist:

  ```
  * PORT <port_name> <net_name> <x> <y> <layer>
  ```

  The *port_name* is <UNNAMED> for Port Layer Polygon objects, otherwise it is a user-given name. The *net_name* is either a user-given name from a text object or an internally assigned numeric net ID. The *port_name* and *net_name* may differ, in which case the *net_name* is the one used in the netlist to indicate connections between objects. The coordinates are given in cell space, and the port's layer is also given. Default is NO.

  See "Port Table File Format" in the *Calibre Query Sever Manual* for related information.

- -primitive_device_subckts {YES | NO}

  Specifies whether the netlist generated by the dfm::write_spice_netlist command contains primitive device subcircuits. The netlist may be generated with or without .SUBCKT statements for user-defined primitive devices. Normally, primitive device subcircuits are included for all user-defined devices in the rule file. Default is YES.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -separated_properties {NO | YES}

  Specifies whether properties saved by the LVS Push Devices SEPARATE PROPERTIES YES statement are written to the output netlist. Default is NO. The YES option may not be used with -annotated_devices YES. The YES option is unsupported in YieldServer.

- -string_properties {YES | NO}

  Specifies whether string properties produced with the device property computation language are written. Default is YES.

- -seed_promoted_trivial_pins {NO | YES}

  Specifies whether trivial pins from cells having seed promotions (*.SEEDPROM in the extracted netlist) are written. Other trivial pins are not written. Default is NO.

- -trivial_pins {NO | YES}

  Specifies whether all trivial pins (pins not connected to devices) are output. Default is NO.

- -reset

  Resets layout netlist generation options to default values.

- -unique_names {NO | YES} [INSTANCES] [NETS]

  Specifies whether reduced (smashed) layout net or instance names that are mapped to source names are output as unique. The defaults are NO NETS and YES INSTANCES, which means mapped net names are not made unique but mapped instance names are. INSTANCES or NETS must be specified with either NO or YES. If NO or YES is specified by itself, it is as though both INSTANCES and NETS were specified.

Return Values

None.

Description

Specifies how various elements are handled during writing of an extracted netlist by dfm::write_spice_netlist. The dfm::read_netlist command has no interaction with dfm::set_layout_netlist_options. In YieldServer, a DFM database must be loaded before using this command. When used in the Query Server Tcl shell, this command writes a netlist based upon PHDB data.

The options of this command behave like Query Server CCI commands in the LAYOUT NETLIST family. The options have similar names to their CCI counterparts, but without using “layout_netlist” as part of the option name. For example, -cell_prefix corresponds to the LAYOUT NETLIST CELL PREFIX command in CCI.

The dfm::set_netlist_options command is related to dfm::set_layout_netlist_options but differs in these ways:

- dfm::set_netlist_options applies to either the layout or source netlist and interacts with dfm::read_netlist.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- dfm::set_netlist_options requires that a netlist object be specified instead of a DFM database or Mask SVDB Directory being loaded.

The -names option used with the SOURCE ALL keywords is useful in the Calibre PERC LDL source-based flow to get the source names. When using the source-based flow, it is highly desirable that LVS be clean. Using dfm::run_compare is a pre-requisite so the source-to-layout correspondences can be determined.

Examples

This example shows an interactive session that writes a layout netlist containing pin location comments for each device. This requires the rule file to specify the DFM Database PINLOC keyword (Mask SVDB Directory PINLOC if using Query Server flow).

```
> dfm::open_db dfmdb
Opening database "dfmdb", revision "master"
> dfm::set_layout_netlist_options -pin_locations YES
> dfm::write_spice_netlist dfm.layout.spi
Loading hierarchy and connectivity
Loading device info
```

Related Topics

dfm::list_layout_netlist_options

Netlist Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::set_netlist_options

Also used in the Query Server Tcl Shell.

Customizes behavior of the dfm::write_spice_netlist command regarding the output of comment-coded string properties.

Usage

dfm::set_netlist_options -netlist_handle *object* -comment_properties {YES | NO}

Arguments

- **-netlist_handle *object***

  A required argument set that specifies a netlist object created by dfm::read_netlist.

- **-comment_properties {YES | NO}**

  A required argument set that specifies whether the output netlist written by dfm::write_spice_netlist uses comment-coded string properties. The default is YES. Specifying no causes such property names not to be preceded by a dollar sign ($).

Return Values

None.

Description

Specifies how string properties are handled by dfm::write_spice_netlist.

The dfm::set_layout_netlist_options command is related but applies specifically to layout netlists generated from a DFM database or Mask SVDB Directory PHDB.

Examples

Assume the following source netlist:

```
.subckt debug N1 N2 N3 N4
Q1 N1 N2 N3 N4 npn2 p=1 string_prop="abc"
.ends
```

This sequence of commands:

```
set nl [ dfm::read_netlist -source -rules rules ]
dfm::write_spice_netlist out.spice -netlist_handle $nl
```

Produces this output:

```
.SUBCKT debug N1 N2 N3 N4
QQ1 N1 N2 N3 N4 NPN2 p=1 $string_prop="abc"
.ENDS
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Notice the string property is coded as a comment, which is the default behavior. Using this command in the flow:

```
dfm::set_netlist_options -netlist_handle $nl -comment_properties NO
```

causes string_prop not to be coded as a comment.

Related Topics

Netlist Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::set_new_layer_error_severity

Specifies an error exception severity for dfm::new_layer.

Usage

dfm::set_new_layer_exception_severity {**DEFAULT** | **MISSING_ORIGINAL_LAYER**}

Arguments

- **DEFAULT**

  Keyword that specifies to return the exception severity level to the original setting.

- **MISSING_ORIGINAL_LAYER**

  Keyword that specifies a missing original layer referenced in a dfm::new_layer command causes an error rather than a warning.

Return Values

None.

Description

This command performs a similar role for the dfm::new_layer command as the Layout Input Exception Severity statement in the Calibre database constructor module.

By default, a missing original layer referenced by dfm::new_layer raises a warning and an empty layer is created. When **MISSING_ORIGINAL_LAYER** is specified, this raises the exception level to an error, and the dfm::new_layer command aborts.

Examples

```
# raise the exception level for a missing layer to an error.
dfm::set_new_layer_error_severity MISSING_ORIGINAL_LAYER

# NON_EXISTING_ORIG is a non-existent layer.
# allow the run to continue in the event of an error.
set cval [catch { dfm::new_layer -svrf "x = COPY NON_EXISTING_ORIG"} emsg]

# give a message in the event of an exception.
if {$cval} {
    puts ""
    puts "PRODUCE ERROR FOR MISSING ORIGINAL LAYER"
    puts $emsg
    puts ""
}

# reset the severity to the default (warning) level.
dfm::set_new_layer_error_severity DEFAULT
...
```

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::split_unmerged

Splits an un-merged DFM Analyze detail layer into two new layers.

Usage

dfm::split_unmerged ***analyze_detail_layer***

Arguments

- ***analyze_detail_layer***

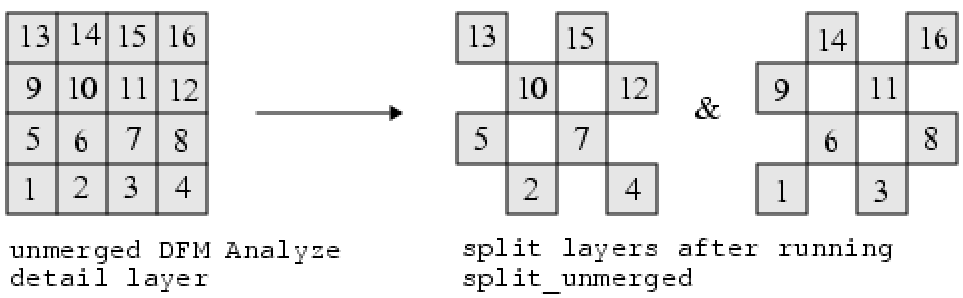
  A required argument that specifies the name of the detail layer generated by a DFM Analyze operation. The command returns an error if ***analyze_detail_layer*** is not a layer of this type.

Return Values

Tcl list.

Description

Splits the specified ***analyze_detail_layer*** into two layers resembling black and white squares on a checkerboard. A list of the generated layer names is returned. This command prevents Calibre from merging a DFM Analyze _detail layer. The windows in the ***analyze_detail_layer*** must not overlap.



Examples

```
# create a DFM Analyze detail layer and split it
dfm::new_layer -svrf {ANALYZE_LAYER = DFM ANALYZE METAL1 WINDOW 16 \
[AREA(METAL1)] >= 0 }
set split_layer [dfm::split_unmerged ANALYZE_LAYER_detail]
```

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::static_analyze_tvf

Retrieves information about Calibre PERC TVF Function block code.

Usage

dfm::static_analyze_tvf *svrf_analyzer tvf_function_name*
[-list [-ruleerrors] [-globalinfo]] [-graphfile *filename*]

Arguments

- ***svrf_analyzer***

  A required SVRF analyzer object generated by dfm::create_svrf_analyzer.

- ***tvf_function_name***

  A required name of a TVF Function block. See "TVF Function" in the *SVRF Manual* for details.

- -list

  An optional argument that specifies to output message data using a Tcl list of lists {*key* {*value_list* ...}}, where each *key* is a string indicating a class of output, and *value_list* is a sub-list associated with the *key*. There are one or more *value_list* arguments. Each *value_list* contains a message string. See Example 3.

- -ruleerrors

  An optional argument specified with -list that causes only rule check errors caused by global variables or invalid rule check functions to be output. The associated list key is ERRS_IN_RULECHECK.

- -globalinfo

  An optional argument specified with -list that causes only global variables to be listed, along with the rule checks that reference them. The associated list key is GLOBALS_IN_RULECHECK.

- -graphfile *filename*

  An optional argument set that specifies an output graph file. An image file can be generated by using the command "dot -Tpng *filename* > *filename*.png".[1]

Return Values

By default, strings of the following forms appear in sections named Rulecheck Errors, Syntax Errors, or File IO:

```
function_name (TVF FUNCTION): line_number: message_type message
```

1. See *www.graphviz.org* for information about dot.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

with these semantics for everything but (TVF FUNCTION), which is always present:

function_name — Name of the TVF Function block.

line_number — Line number in the scope of the TVF Function.

message_type — ERROR, WARNING, NOTE, or File. "File" is only given for File IO messages.

message — Informational statement about what is reported.

By default, strings of this form appear for the Global Usage sections:

```
::global_variable: rule_check
```

with these semantics:

global_variable — Name of a global variable.

rule_check — Name of a rule check.

When -list is used, the strings shown previously appear in list elements. Example 3 shows sample outputs.

Description

Returns information about code in a TVF Function block used in Calibre PERC. By default the output is in report format demarcated by section headers indicating classes of messages that follow them. The four classes are Rulecheck Errors, Syntax Errors, File IO, and Global Usage.

The first two classes relate to problems with rule check procs and Tcl syntax, respectively.

File IO messages indicate when file reading and writing occurs. This is generally problematic for performance and should be avoided. In multithreaded runs, this can be particularly problematic because file access can happen from disparate threads and in an unpredictable order.

Global Usage messages indicate when global variables are used in rule checks. These can be problematic when used in multithreaded runs because the order of setting, resetting, and accessing them is unpredictable in that context.

When -list is used, the outputs are in Tcl list form. Keys can be used to access the four classes of messages discussed previously. These keys are ERRS_IN_RULECHECK, SYNTAX_ERRS, FILE_IO, and GLOBALS_IN_RULECHECK, respectively. Example 2 shows sample usage code.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This shows a simple usage to output the default format report to the run transcript. The TVF Function in the *rules* file is tvf_lib.

```
# Report to transcript
set svrf_analyzer [dfm::create_svrf_analyzer rules]
# use default format
puts [dfm::static_analyze_tvf $svrf_analyzer "tvf_lib"]
```

Example 2

This example shows use of -list. The messages_${c} variables can then be processed further to output desired information.

```
# -list usage
# return the result lists for each key
proc get_msg_list {results key} {
  foreach entry $results {
    if {[lindex $entry 0] == $key} {
      return [lindex $entry 1]
    }
  }
  return {}
}

set svrf_analyzer [dfm::create_svrf_analyzer rules]
set keys "ERRS_IN_RULECHECK SYNTAX_ERRS FILE_IO GLOBALS_IN_RULECHECK"
set results [dfm::static_analyze_tvf $svrf_analyzer "tvf_lib" -list]
set c 1
foreach k "$keys" {
  set messages_${c} [get_msg_list $results "$k"]
  incr c
}
...
```

Example 3

Following are sample list outputs for the four keys.

```
{ERRS_IN_RULECHECK {{constraints (TVF FUNCTION): 1010: ERROR   rulecheck
command perc::check_data used in init proc void}}}

{SYNTAX_ERRS {{constraints (TVF FUNCTION): 4788: ERROR   Too many
arguments to 'return'} {constraints (TVF FUNCTION): 1223: NOTE    Can't
find ./.xchange_file.tcl}}}

{FILE_IO {{constraints (TVF FUNCTION): 1602:  File $constraint_file opened
for writing} {constraints (TVF FUNCTION): 2482:  File cg.rep opened for
writing}}}

{GLOBALS_IN_RULECHECK {{::devstats rule1} {::cellstats rule2} {::settings
rule3}}}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Rule File Query Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::transform_vertices

Applies a placement transformation to a list of vertices.

Usage

dfm::transform_vertices *"vertices_list"* -xform *xform* [-inverse]

Arguments

- ***"vertices_list"***

  A required argument that specifies vertices as a Tcl list. Vertices are specified as X and Y coordinate pairs in succession and in user units.

- **-xform *xform***

  A required argument that specifies a transformation to apply. The ***xform*** is an object generated by dfm::get_data -xform.

- -inverse

  An optional argument that applies the inverse transformation contained in the ***xform*** object.

Return Values

List.

Description

Applies a placement transformation to a list of vertices and returns the result as a list.

By default, the transformation is to the coordinate space of an ancestor cell placement as defined by the ***xform*** object. When -inverse is used, the inverse transform down the hierarchy to a descendant cell placement is applied.

This command does not check coordinate values for reasonableness under transformation. For example, if the ***xform*** object has a transformation to top-level space, and the ***vertices_list*** contains top-level coordinates, then using dfm::transform_vertices with that set of inputs will return coordinates outside of top-level space. Hence, you need to be aware of what hierarchy level the ***vertices_list*** coordinates come from versus the transformation in the ***xform*** object. The dfm::get_data -depth option can be helpful when navigating the hierarchy.

Examples

This example reports the coordinates of shapes in the DFM database in both local cell and primary cell coordinate spaces. It also reports instance paths for cell placements in which the shapes occur.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# cells of interest
set cell nand
set top [dfm::get_top_cell]
# mn_seed is a layer saved in the rules
# rule::<2> is a derived error layer from a rule check
set layer_list "mn_seed rule::<2>"
set list_idx [expr [llength $layer_list] -1]
while {$list_idx >= 0} {
  set layer [lindex $layer_list $list_idx]
  puts "  layer: $layer"
  set cell_placements [dfm::get_placements $top -of_cell $cell -flat]
  while {$cell_placements != ""} {
# if its a derived error layer, get the top-level shapes.
# otherwise, get the cell-level shapes.
    if {[string match {::<} $layer]} {
      set cell_layer_shapes [dfm::get_geometries $layer -cell $top]
    } else {
      set cell_layer_shapes [dfm::get_geometries $layer -cell $cell]
    }
# get the placement contexts and transforms
    set placement_path [dfm::get_data $cell_placements -path_name]
    set path_context [dfm::get_path_context $top -path $placement_path]
    set xform [dfm::get_data $path_context -xform]
    while {$cell_layer_shapes != ""} {
      set layer_loc [dfm::get_data $cell_layer_shapes -vertices]
      puts "  $cell ${placement_path}:"
# if a derived error layer, transform to local cell space.
# otherwise, transform to top-level space.
      if {[string match {::<} $layer]} {
        puts "    cell space: [dfm::transform_vertices $layer_loc \
                               -xform $xform -inverse]"
        puts "    top space:  $layer_loc"
      } else {
        puts "    cell space: $layer_loc"
        puts "    top space:  [dfm::transform_vertices $layer_loc \
                               -xform $xform]"
      }
      dfm::inc cell_layer_shapes
    }
    dfm::inc cell_placements
  }
  incr list_idx -1
}
exit
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The coordinate output in the transcript is per-layer, per-placement, per-shape. This output is for a derived error layer:

```
  layer: rule::<2>
Loading hierarchy and connectivity
Loading device info
Loading layer rule::<2> into the hierarchical database
  nand X1/X0:
    cell space: 5375 14000 6625 29000
    top space:  62085 14000 63335 29000
  nand X1/X0:
    cell space: 13375 14000 14625 29000
    top space:  70085 14000 71335 29000
```

Related Topics

dfm::apply_transform

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::unload_layer

Unloads the specified layer from memory. This command is used for memory optimization; the layer on disc is unchanged. You should ensure that the specified layer is saved to disc before unloading it.

Usage

dfm::unload_layer *layer_name*

Arguments

- ***layer_name***

  Required argument that specifies the name of the layer to unload.

Return Values

None.

Related Topics

Layer Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::update_rev_format

Creates a new revision by saving the current open (frozen) revision.

Usage

dfm::update_rev_format

Arguments

None.

Return Values

TCL_OK or TCL_ERROR.

Description

Creates a new revision by saving the current open revision, which must be a frozen revision. This commands loads the hierarchy and each layer, and re-saves them to disk. This can take a considerable amount of time for large databases.

Once the new revision is created, you can set it as the default revision using dfm::set_default_rev.

Related Topics

Database Administration Commands

DFM Database Revisions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::v_minmax

Returns minimum and maximum values of a vector property.

Usage

dfm::v_minmax ***iterator property_name***

Arguments

- ***iterator***

  A required argument that specifies a geometry iterator. The current shape in the iterator is used. An error occurs if the iterator is not a geometry iterator.

- ***property_name***

  A required argument that specifies the name of a numeric vector property. An error occurs if the property is not a vector property.

Return Values

Tcl list.

Description

Evaluates the specified vector property associated with the current shape in the ***iterator*** and returns the minimum and maximum values in the vector as a Tcl list in the form {min max}. The returned values are floating-point numbers.

Examples

```
# check if property1 is a vector property. if so, write the
# min and max values to the transcript
if {[dfm::get_data $iterator -is_vector_property property1]} {
    set result [dfm::v_minmax $iterator property1]
    puts "min is [lindex $result 0]"
    puts "max is [lindex $result 1]"
}
```

Related Topics

DFM Property Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::v_sumprod

Returns sum and product values of a vector property.

Usage

dfm::v_sumprod ***iterator property_name***

Arguments

- ***iterator***

  A required argument that specifies a geometry iterator. The current shape in the iterator is used. An error occurs if the iterator is not a geometry iterator.

- ***property_name***

  A required argument that specifies the name of a numeric vector property. An error occurs if the property is not a vector property.

Return Values

Tcl list.

Description

Evaluates the specified vector property associated with the current shape in the ***iterator*** and returns the sum and product of the values in the vector property as a Tcl list in the form {sum product}. The returned values are floating-point numbers.

Examples

```
# check if property1 is a vector property. if so, write the
# sum and product values to the transcript.
# the iterator is a geometry iterator.
if {[dfm::get_data $iterator -is_vector_property property1]} {
    set result [dfm::v_sumprod $iterator property1]
    puts "sum is [lindex $result 0]"
    puts "prod is [lindex $result 1]"
}
```

Related Topics

DFM Property Management Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_cmds

Writes out a history of all the commands that were successfully executed during the run.

Usage

dfm::write_cmds *filename*

Arguments

- ***filename***

  A required argument that specifies the name for the file to be created. If the file exists, it is overwritten.

Return Values

None.

Description

Writes out a history of all the commands that were successfully executed during the run. The output file is a syntactically correct Tcl script of commands. This is typically different from the transcript.

Examples

This saves all the successful commands from an interactive session to the file named *replay.tcl*.

```
> dfm::write_cmds replay.tcl
```

Related Topics

dfm::help

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_cmp_report

Writes an LVS comparison report to the specified file. This command is valid only if the dfm::run_compare command has been executed to create an XDB in the DFM database.

Usage

dfm::write_cmp_report ***filename***

Arguments

- ***filename***

  A required argument that specifies the name of the output LVS report.

Return Values

None.

Examples

```
# perform an LVS comparison and write the report
dfm::run_compare -source_netlist adder4.spi -source_primary adder4
dfm::write_cmp_report lvs.ys.report
```

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_gds

Writes specified layers to a GDS database file. The various options configure the structure of the database.

Usage

dfm::write_gds {{**-layer_info** '**{**' ***input_layer_name layer_number*** [*datatype*] [-flatten]
[[-aref {*cell width length* [*min_element_count*] [-substitute {*x1 y1... xn yn*}]}] |
[-autoref [*cell_name*]]] '**}**'} …} **-file *filename***
[-pseudo] [-compressed] [-noempty]
[-drc_magnify_result *value*]
[-drc_results_database_precision *value*]

Arguments

- **-layer_info** '**{**' ***input_layer_name layer_number*** …'**}**'

  A required argument set that specifies a layer to be output. The ***input_layer_name*** is the name of a layer in the DFM database. The ***layer_number*** must be a non-negative integer or numeric variable. These are used in the output file. These elements are specified as part of a Tcl list. The entire argument set from **-layer_info** through -autoref may be specified more than once.

- *datatype*

  An optional non-negative integer or variable corresponding to an output datatype. The default is 0.

- -flatten

  An optional argument that specifies the output layer is flattened. Specified as part of the **-layer_info** argument set.

- -aref {*cell width length* [*min_element_count*] [-substitute {*x1 y1...xn yn*}]

  An optional argument set that specifies rectangle arrays are stored as AREF structures. Specified as part of the **-layer_info** argument set. May not be specified with -autoref.

  The semantics of this option are essentially the same as the DRC Check Map AREF keyword. The *cell*, *width*, *length*, *min_element_count*, -substitute argument set must follow the -aref argument in the order shown. The parameters are defined as follows:

  cell — Specifies the name of the target cell in which the AREF structures are placed. The *cell* may be a singleton string variable.

  The cell name may not match any *placed* structure names in the input layout databases (or any pseudocell names). If this occurs, a warning is issued and output is suppressed.

  width — Positive floating-point number, numerical expression, or variable that specifies the width in user units of rectangles to be converted into an AREF structure.

  length — Positive floating-point number, numerical expression, or variable that specifies the length in user units of rectangles to be converted into an AREF structure.

min_element_count — Argument that specifies the minimum element count of output AREF structures. If the element count is greater than or equal to *min_element_count*, the structure is output as an AREF. Must be a positive integer if specified. Default is 16.

-substitute *x1 y1...xn yn* — Argument set that specifies a polygon with vertices having floating-point coordinates *x1 y1...xn yn* to constitute AREF structures rather than a rectangle. Values are in user units. A rectangle is the default. The polygon must have at least two coordinate pairs, may not intersect itself, and be orientable (the interior of the polygon must be consistently to the right or the left when traversing the perimeter). A two-point polygon is interpreted as a rectangle with sides parallel to the database axes.

- -autoref [*prefix*]

  An optional argument set that instructs Calibre to perform automatic rectangle compaction on the layer. The semantics of this option are essentially the same as the DRC Check Map AUTOREF keyword. Specified as part of the **-layer_info** argument set. May not be specified with -aref.

- **-file *filename***

  A required argument set that specifies the name of the output GDS database.

- -pseudo

  An optional argument outputs pseudocells. By default, pseudocell output is suppressed, and such cells are expanded into non-pseudocells.

- -compressed

  An optional argument indicates input layers were generated with the DFM Fill COMPRESS keyword.

- -noempty

  An optional argument that prevents empty cells from being output. By default, they are output.

- -drc_magnify_result *value*

  An optional argument set that magnifies the output similar to the DRC Magnify Results rule file statement. The *value* is a positive, floating-point number (or numeric expression) that is a dimensionless multiplication factor applied to vertex coordinates. By default, the magnification of the DFM database is used.

- -drc_results_database_precision *value*

  An optional argument set that specifies the output database precision similar to the DRC Results Database Precision rule file statement's behavior. The *value* is a positive, floating-point number (or numeric variable) that specifies the precision value. By default, the precision of the DFM database is used.

Return Values

None.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 2 under dfm::new_layer shows code that can be modified to use dfm::write_gds to output layers by net name.

Example 1

```
dfm::write_gds -layer_info {M1 500 0} -file m1.gds
# Writes the layer M1 in GDS format to the file m1.gds.
# 500 is the output layer number and 0 is the datatype.
```

Example 2

```
dfm::write_gds -layer_info {M1 500 0 -aref {cellA 0.01 0.01 \
-substitute {0 0 0.01 0.01}}} -file layer.gds
# Writes the layer M1 in GDS format to the file layer.gds.
# 500 is the output layer number and 0 is the datatype. Arrayed rectangles
# of width 0.01 and height 0.01 are reduced into AREF structures, which
# are constituted by a polygon with coordinates 0, 0, 0.01, 0.01.
```

Example 3

```
dfm::write_gds -layer_info {M1 500 0} -layer_info {M2 501 0} -file out.gds
# Writes layers M1 and M2 out in GDS format to the file out.gds
# 500 and 501 are the output layer numbers and 0 is the datatype for each.
```

Example 4

```
dfm::write_gds -layer_info {m1fill_compress 100 0} \
               -layer_info {m1fill_compress_h 100 1} \
               -compressed -file compressed_fill_ys.gds
# Input layers are DFM Fill COMPRESS layers.
```

Example 5

```
dfm::write_gds -layer_info {POLY 500 0 -flatten} \
               -layer_info {POLY 501 0} -noempty -file noempty_mix.gds
# Layer 500.0 is flattened; 501.0 is not. No empty cells are written.
```

Related Topics

Database Output Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_ixf

Also used in the Query Server Tcl shell.

Writes an instance cross-reference (IXF) file. Requires a Calibre Connectivity Interface (CCI) license.

Usage

dfm::write_ixf ***filename*** [-box_cells] [-source_omn] [-layout_omn]

Arguments

- ***filename***

  A required argument that specifies the name of the output file. If the output ***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_cells

  An optional argument that includes LVS Box cells in the output.

- -source_omn

  An optional argument that includes original model names for the source in the output.

- -layout_omn

  An optional argument that includes original model names for the layout in the output.

Return Values

None.

Description

The behavior and output of this command are similar to the INSTANCE XREF WRITE command of the Calibre Query Server. This command is valid only if the dfm::run_compare command has been executed to create the cross-reference database (XDB) in the DFM database.

Format of the output file is described under "Instance and Net Cross-Reference File Formats" in the *Calibre Query Server Manual*.

Examples

```
# run comparison, write report and cross-reference files
dfm::run_compare -source_netlist src.spi -source_primary top
dfm::write_cmp_report lvs.ys.report
dfm::write_ixf full_chip.ixf
dfm::write_nxf full_chip.nxf
```

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_lph

Also used in the Query Server Tcl shell.

Writes a layout placement hierarchy (LPH) file. Requires a Calibre Connectivity Interface (CCI) license.

Usage

dfm::write_lph ***filename*** [-omn]

Arguments

- ***filename***

  A required argument that specifies the name of the output file. If the output ***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 argument that specifies original model names are written. If this option is used, the Mask SVDB Directory OMN keyword must be specified in the rules.

Return Values

None.

Description

The behavior and output of this command is similar to the LAYOUT HIERARCHY WRITE command of the Calibre Query Server. Details of the output file format are provided under "Placement Hierarchy (LPH/SPH) File Format" in the *Calibre Query Server Manual*.

This command is valid in Calibre YieldServer only if the dfm::run_compare command has been executed to create the cross-reference database (XDB) in the DFM database.

Examples

```
# run lvs, write report and placement hierarchy files
dfm::run_compare -source_netlist src.spi -source_primary top
dfm::write_cmp_report lvs.ys.report
dfm::write_lph full_chip.lph
dfm::write_sph full_chip.sph
```

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_nxf

Also used in the Query Server Tcl shell.

Writes a net cross-reference (NXF) file. Requires a Calibre Connectivity Interface (CCI) license.

Usage

dfm::write_nxf ***filename*** [-box_cells] [-lnxf | -unmatched]

Arguments

- ***filename***

  A required argument that specifies the name of the output file. If the output ***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_cells

  An optional argument that includes LVS Box cells in the output.

- -lnxf

  An optional argument that includes layout 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. This output is similar to the LAYOUT NET XREF WRITE command output of the Calibre Query Server. May not be specified with -unmatched.

- -unmatched

  An optional argument that includes unmatched layout nets in the output. May not be specified with -lnxf.

Return Values

None.

Description

The default output of this command is similar to the NET XREF WRITE command of the Calibre Query Server. This command is valid only if the dfm::run_compare command has been executed to create the cross-reference database (XDB) in the DFM database.

Format of the output file is described under "Instance and Net Cross-Reference File Formats" in the *Calibre Query Server Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

```
# run lvs, write report and cross-reference files
dfm::run_compare -source_netlist src.spi -source_primary top
dfm::write_cmp_report lvs.ys.report
dfm::write_ixf full_chip.ixf
dfm::write_nxf full_chip.nxf
```

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_oas

Writes specified layers to an OASIS database file. The various options configure the structure of the database.

Usage

dfm::write_oas {{**-layer_info** '**{**' ***input_layer_name layer_number*** [*datatype*] [-flatten]
[-aref {*cell width length* [*min_element_count*] [-substitute {*x1 y1... xn yn*}]}]
[-write_properties] '**}**'} ...} **-file *filename***
[-flatten] [-write_properties] [-pseudo] [-compressed] [-noempty]

Arguments

- **-layer_info** '**{**' ***input_layer_name layer_number*** ... '**}**'

  A required argument set that specifies a layer to be output. The ***input_layer_name*** is the name of a layer in the DFM database. The ***layer_number*** must be a non-negative integer or numeric variable. These are used in the output file. These elements are specified as part of a Tcl list. The entire argument set from **-layer_info** through -write_properties may be specified more than once.

- *datatype*

  An optional non-negative integer or variable corresponding to an output datatype. The default is 0.

- -flatten

  An optional argument that specifies the output layer is flattened when the option is part of **-layer_info** argument set. If used outside of the **-layer_info** argument set, specifies the output database is flat.

- -aref {*cell width length* [*min_element_count*] [-substitute {*x1 y1... xn yn*}]

  Note

  This keyword set is not needed with OASIS output because OASIS placements are stored in arrays automatically.

  An optional argument set that specifies rectangle arrays are stored as AREF structures. Specified as part of the **-layer_info** argument set.

  The semantics of this option are essentially the same as the DRC Check Map AREF keyword. The *cell*, *width*, *length*, *min_element_count*, -substitute argument set must follow the -aref argument in the order shown. The parameters are defined as follows:

  cell — Specifies the name of the target cell in which the AREF structures are placed. The *cell* may be a singleton string variable.

  The cell name may not match any *placed* structure names in the input layout databases (or any pseudocell names). If this occurs, a warning is issued and output is suppressed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

width — Positive floating-point number, numerical expression, or variable that specifies the width in user units of rectangles to be converted into an AREF structure.

length — Positive floating-point number, numerical expression, or variable that specifies the length in user units of rectangles to be converted into an AREF structure.

min_element_count — Argument that specifies the minimum element count of output AREF structures. If the element count is greater than or equal to *min_element_count*, the structure is output as an AREF. Must be a positive integer if specified. Default is 16.

-substitute *x1 y1... xn yn* — Argument set that specifies a polygon with vertices having floating-point coordinates *x1 y1... xn yn* to constitute AREF structures rather than a rectangle. Values are in user units. A rectangle is the default. The polygon must have at least two coordinate pairs, be simple (it cannot be self-intersecting), and orientable (the interior of the polygon must be consistently to the right or the left when traversing the perimeter). A two-point polygon is interpreted as a rectangle with sides parallel to the database axes.

- **-file *filename***

  A required argument set that specifies the name of the output OASIS database.

- -write_properties

  An optional argument that specifies the output layer contains DFM properties of the input layer when the option is part of the **-layer_info** argument set. If used outside of the **-layer_info** argument set, specifies all output layers contain DFM properties of the input layers.

- -pseudo

  An optional argument outputs pseudocells. By default, pseudocell output is suppressed, and such cells are expanded into non-pseudocells.

- -compressed

  An optional argument indicates input layers were generated with the DFM Fill COMPRESS keyword.

- -noempty

  An optional argument that prevents empty cells from being output. By default, they are.

Return Values

None.

Examples

Example 2 under dfm::new_layer shows code to output OASIS layers by net name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 1

```
dfm::write_oas -layer_info {M1 500 0} -file m1.oas
# Writes the input layer M1 in OASIS format to the file m1.oas
# M1 is the output layer name, 500 is the layer number, and
# 0 is the datatype
```

Example 2

```
dfm::write_oas -layer_info {M1 500 0} -layer_info {M2 501 0} -file out.oas
# Writes input layers M1 and M2 in OASIS format to the file out.oas
# M1 and M2 are also output layer names.
# Output layers are 500.0 and 501.0.
```

Example 3

```
dfm::write_oas -layer_info {m1fill_compress 100 0} \
               -layer_info {m1fill_compress_h 100 1} \
               -compressed -file compressed_fill_ys.oas
# Input layers are DFM Fill COMPRESS layers.
```

Example 4

```
dfm::write_oas -layer_info {POLY 500 0 -flatten} \
               -layer_info {POLY 501 0} -noempty -file noempty_mix.oas
# Layer 500.0 is flattened; 501.0 is not. No empty cells are written.
```

Related Topics

Database Output Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_rdb

Also used in the Query Server Tcl shell.

Writes specified layers to an ASCII RDB file. The various options configure the structure of the database.

Usage

dfm::write_rdb -layer *layer_list* -file *filename* [*RDB_options*] [-append] [-cell_list *name*] [-suppress_optext] [-nodelims]

Arguments

- **-layer *layer_list***

  A required argument set that specifies layers written to the output database. The ***layer_list*** must be supplied as a Tcl list of layer names. Up to three may be specified.

  When multiple input layers are specified, the command clusters the secondary layers with the primary layer into a single rule check (the name of which matches the name of the primary layer, unless -check_name is specified). To write multiple layers to the same RDB, use a loop in conjunction with the -append option (see “Example 1” on page 288).

- **-file *filename***

  A required argument set that specifies the name of the output database.

- *RDB_options*

  An optional argument set controlling how the data in the DFM database is organized. These are similar to the DFM RDB keywords sharing the analogous names.

  -abut_also — Specifies that primary layer polygons abutting secondary layer polygons are output. This option only applies if there is more than one layer in the ***layer_list***.

  -all_cells — Specifies that one check is written for all cells. When -all_cells is specified, the DFM RDB matches the format of the default DRC Results Database. When -all_cells is not specified, a separate check is output for each cell.

  -no_cells — Specifies not to write cell headers in the output. This option emulates DRC Cell Name NO. You cannot use -no_cells with multiple input layers or the following options: -nodal, -cell_space, -all_cells, or -emulate_trans.

  -cell_space — Specifies to write the output coordinates in cell space. This option emulates DRC Cell Name YES CELL SPACE XFORM. You cannot use -cell_space with multiple input layers or the following options: -nodal, -no_cells, or -emulate_trans.

  -check_name *name* — Specifies that *name* be used instead of the layer name in the checks the output layers appear under. By default the output check names are of the form *layer*_N, where layer is the name of the layer and N is an integer. If -all_cells is used, then *name* is the check name with no integer.

  -comment “*comment_string*” — Specifies that a comment be added to the output for the layers. The *comment_string* must be quoted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-emulate_trans — Specifies to use the DFM Transition RDB format. When this option is used, there must be three layers in the ***layer_list***.

-filter *filter_object* — An optional argument set specifying a dfm::create_filter object containing transformation information. The transformations in the *filter_object* are applied to output coordinates. Inverse cell transformations stored in the database when -cell_space is used are affected by the filter object transformations.

-maximum *value* — Specifies a maximum number of objects from the layer to be written. The *value* is a non-negative integer. May not be specified with -maximum_all.

-maximum_all — Specifies that all objects from the layer are written. This is a default behavior. May not be specified with -maximum.

-nodal — Specifies to preserve node numbers.

-nodelims — Specifies vector properties written to the database have no < > delimiters.

-noempty — Specifies not to write empty cells.

-nopseudo — Specifies not to write pseudocells. This option is ignored for unmerged input layers.

-print — Specifies the output uses the Density PRINT option format.

-same_cell — Specifies that clustered objects are output on the same hierarchical level. This option only applies if there is more than one layer in the ***layer_list***.

- -append

  An optional argument used to append the output to the specified file, which is an RDB. The default behavior is to overwrite any existing contents.

- -cell_list *name*

  An optional argument set that specifies a cell list *name* defined with a Layout Cell List statement. This argument can be used to restrict the output to certain cells and is equivalent to using the RESULT CELLS option in the DFM RDB operation.

- -suppress_optext

  An optional argument that suppresses the printing of layer operation text as part of the RDB.

- -nodelims

  An optional argument that suppresses printing of delimiters for vector properties.

Return Values

None.

Description

Creates a DFM RDB (ASCII results database) containing the specified layers. For information on these alternative types of databases, refer to the *Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

"Example 2" on page 234 shows code that can be modified to use dfm::write_rdb to output layers by net name.

Example 1

This shows how to write multiple layers to the RDB.

```
# write multiple layers to the same DFM RDB file
# without the default clustering behavior.

set layer_list [list m1 m2 m3 m4 m5 m6 m7]
set i 1
foreach L $layer_list {
  if [expr {$i == 1}] {
# For the first layer, generate the RDB file.
    dfm::write_rdb -layer $L -file metals.rdb
  } else {
# For the rest of the layers, append the results to the same file
    dfm::write_rdb -layer $L -file metals.rdb -append
  }
  incr i
}
```

Example 2

This example shows corresponding outputs for three commands.

For this command:

```
dfm::write_rdb -layer metal2 -file out.rdb
```

This shows the header of *out.rdb* and the beginning of a sub-cell entry:

```
metal2_2
2 2 2 Jan 13 15:59:47 2015
CELL CellB 1 0 0 1 -29000 0
IL: metal2
p 1 4
31000 32000
35000 32000
35000 36000
31000 36000
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Adding the -cell_space option, the results are organized under a single check and the cell transformation information is given:

```
TOPCELL 1000
metal2
8 8 1 Jan 13 16:00:14 2015
IL: metal2
...

p 4 4
CN CellB c 1 0 0 1 29000 0 1
2000 32000
6000 32000
6000 36000
2000 36000
```

If the -all_cells option is used instead of -cell_space, there is this:

```
TOPCELL 1000
metal2
8 8 1 Jan 13 16:00:34 2015
IL: metal2
...

p 4 4
CN CellB 1 0 0 1 -29000 0 1
31000 32000
35000 32000
35000 36000
31000 36000
```

Related Topics

Database Output Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_reduction_data

Also used in the Query Server Tcl shell.

Writes a file containing information about transformation reductions occurring in LVS comparison. Requires a Calibre Connectivity Interface (CCI) license.

Usage

dfm::write_reduction_data ***filename*** {**-layout** | **-source** {**-net** | **-instance**}}
[-flat [-with_xref]] [-reason *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**

  Keyword that specifies the reduction data from the layout design are written. Either **-layout** or **-source** must be specified.

- **-source**

  Keyword that specifies the reduction data from the source design are written. Either **-layout** or **-source** must be specified.

- **-net**

  Keyword that specifies the reduction data from nets is written. Either **-net** or **-instance** must be specified.

- **-instance**

  Keyword that specifies the reduction data from instances is written. Either **-net** or **-instance** must be specified.

- -flat

  An optional keyword specifying the data is written in flat format vice 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 *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" in the *Calibre Query Server Manual*. Only one code may be specified per command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

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. The output of this command is similar to the REDUCTION DATA WRITE command in the standard Query Server.

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.

Details of the limitations regarding use of the -flat and -with_xref keywords are discussed under "Reduction Data File Command" in the *Calibre Query Server Manual*.

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_sph

Also used in the Query Server Tcl shell

Writes a source placement hierarchy (SPH) file. Requires a Calibre Connectivity Interface (CCI) license.

Usage

dfm::write_sph *filename* [-omn]

Arguments

- ***filename***

  A required argument that specifies the name of the output file. If the output ***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 argument that specifies original model names are written. If this option is used, the Mask SVDB Directory OMN keyword must be specified in the rules.

Return Values

None.

Description

The behavior and output of this command is similar to the SOURCE HIERARCHY WRITE command of the Calibre Query Server. Details of the output file format are provided under "Placement Hierarchy (LPH/SPH) File Format" in the *Calibre Query Server Manual*.

This command is valid in Calibre YieldServer only if the dfm::run_compare command has been executed to create the cross-reference database (XDB) in the DFM database.

Examples

```
# run lvs, write report and placement hierarchy files
dfm::run_compare -source_netlist src.spi -source_primary top
dfm::write_cmp_report lvs.ys.report
dfm::write_lph full_chip.lph
dfm::write_sph full_chip.sph
```

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::write_spice_netlist

Also used in the Query Server Tcl shell.

Writes a SPICE netlist to the specified file.

Usage

dfm::write_spice_netlist *filename* [-netlist_handle *object*]

Arguments

- ***filename***

  Required argument that specifies the name of the output netlist. If the output ***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.

- -netlist_handle *object*

  An optional argument set that specifies a netlist object created by dfm::read_netlist. This option must be specified if no DFM Database (YieldServer usage) or Mask SVDB Directory (Query Server usage) is currently loaded, if you want netlist transformations applied to the output, or if you want output based upon a source netlist.

Return Values

Hierarchical device count.

Description

Writes a SPICE netlist to ***filename***. The output of this command can be configured with the dfm::set_layout_netlist_options, dfm::set_netlist_options, or dfm::read_netlist commands.

The output produced by this command complies with the Calibre SPICE netlist format and is readable by Calibre nmLVS-H. Note that device properties are not written to the output unless Trace Property statements specify the properties.

If no DFM database or SVDB is currently loaded into the server application, then dfm::read_netlist must be used to create a netlist object, and the dfm::write_spice_netlist -netlist_handle option must be used.

Calibre YieldServer Usage with DFM Database Loaded

If both connectivity and devices are present in the DFM database and -netlist_handle is not used, then the output is similar to that produced by calibre -spice.

If the -netlist_handle option is not used and connectivity is not present in the DFM database, this command does not write a SPICE netlist. If connectivity is present and devices are not extracted (DFM Database DEVICES option was not used), this command writes out a SPICE

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

netlist for connectivity only when the -empty_cells and -trivial_pins options are set to YES in the dfm::set_layout_netlist_options command.

If the -netlist_handle option is used, the output is based upon the configuration of the dfm::read_netlist command that generated the *netlist_object*. The -netlist_handle option must be used if you want output based upon a source netlist.

Note

LVS Push Devices SEPARATE PROPERTIES YES and LVS Preserve Box Cells YES are not supported with dfm::write_spice_netlist in calibre -dfm or -ys.

Calibre Query Server Usage With SVDB Loaded

If the -netlist_handle option is not used, then the output is based upon the PHDB and is similar to that produced by calibre -spice or the Calibre Connectivity Interface LAYOUT NETLIST WRITE command.

The -netlist_handle option must be used if you want output based upon a source netlist or any time you want output with netlist transformations applied.

Examples

Example 1

If a DFM database is loaded in YieldServer, connectivity extraction and device information must be available in the database for this code to work. Pin location information is available in the output netlist so long as the DFM Database PINLOC keyword is used in the rules (Mask SVDB Directory PINLOC in Query Server flows).

```
> dfm::set_layout_netlist_options -pin_locations YES
> dfm::write_spice_netlist dfm.layout.spi
```

Example 2

This example shows a command sequence for writing a transformed source netlist. This command sequence can be used in either Query Server or YieldServer. A database is not loaded for this command sequence.

```
# read the source netlist from the rules file; use LVS transforms
set source_handle [dfm::read_netlist -source -rules rules \
  -netlist_transforms {deep_shorts high_shorts trivial_pins reduced}]
# write the transformed netlist
dfm::write_spice_netlist src_xform.sp -netlist_handle $source_handle
# close the netlist object
dfm::close_netlist
```

Related Topics

Netlist Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dfm::xref_xname

Also used in the Query Server Tcl shell.

Configures the subcircuit call format of cross-reference file commands. Requires a Calibre Connectivity Interface (CCI) license.

Usage

dfm::xref_xname {**-layout** | **-source**} {**-on** | **-off**} [-box_cells] [-device]

Arguments

- **-layout**

  Option that causes layout names to be configured. This option is used by default. When the command is used explicitly, either **-layout** or **-source** must be specified.

- **-source**

  Option that causes source names to be configured. This option is used by default. When the command is used explicitly, either **-layout** or **-source** must be specified.

- **-on**

  Option that causes the normal hierarchical pathname format used in LVS and in the Query Server to be output. This is the default.

- **-off**

  Option that causes subcircuit calls in paths to omit the leading X character. This does not include primitive subcircuit calls by default.

- -box_cells

  An optional keyword that causes only LVS Box cell primitive subcircuit calls to be processed by the command.

- -device

  An optional keyword that causes only primitive device subcircuit calls to be processed by the command.

Return Values

None.

Description

The behavior of this command is similar to the XREF XNAME command of the Calibre Query Server. This command configures the outputs of dfm::write_ixf, dfm::write_nxf, dfm::write_lph, and dfm::write_sph.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The default dfm::write_ixf 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 dfm::xref_xname -source -off -device is used, the output would be this:

```
0 X0/X0/X0 0 69/42/L0
```

Related Topics

LVS and CCI Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Calibre YieldServer Runtime Messages

Calibre YieldServer error and warning messages are issued at runtime to assist in debugging.

Table 3-23. Calibre YieldServer Error Messages

Error Code	Explanation
Error 1	Yield Server is not initialized.
Error 2	Need to load a DFM database first. Use open_db or open_rev.
Error 3	Reserved for internal use.
Error 4	Reserved for internal use.
Error 5	Invalid string representation of an object.
Error 6	Invalid operation on revision. Refer to "DFM Database Revisions" on page 43 for valid operations.
Error 7	Cannot modify a frozen revision.
Error 8	Invalid data (for example, an invalid Tcl iterator).
Error 9	A layer is not configured for adding properties.
Error 10	Copy database failed.
Error 11	Invalid SVRF in a dfm::new_layer command.
Error 12	The specified argument is invalid for the specified command.
Error 13	The specified argument is the wrong type for the specified command.
Error 14	Unable to open the specified file.
Error 15	The specified functionality is unsupported.
Error 16	A problem occurred while evaluating the specified expression.
Error 17	Trailing slash in the specified destination DFM database path is not allowed.
Error 18	Cannot find the specified data.
Error 19	Cannot access encrypted data.
Error 20	A command is missing a required argument.
Error 21	Database is already loaded.
Error 22	There is no command history to write out. Check your file permissions.
Error 23	An exception occurred in the DFM database. The exception is displayed.
Error 24	Reserved for internal use.
Error 25	Reserved for internal use.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-23. Calibre YieldServer Error Messages (cont.)

Error Code	Explanation
Error 26	Need to save changes before executing the specified command.
Error 27	Cannot determine the top level cell for the displayed design.
Error 28	Reserved for internal use.
Error 29	Cannot open the specified database.
Error 30	Wrong number of arguments for the specified command.
Error 31	Cannot delete the specified data from the currently open database.
Error 32	Reserved for internal use.
Error 33	Cannot get the path to the current working directory. A stale NFS handle often causes this error.
Error 34	Cannot find the specified layer.
Error 35	Cannot add the specified property.
Error 36	The displayed operation is invalid for the layer type.
Error 37	Unable to annotate the specified property.
Error 38	Unable to compute the specified value.
Error 39	Unable to delete the specified layer from the currently open database.
Error 40	Cannot execute non-layer operations in a dfm::new_layer command.
Error 41	Unsaved changes. To discard changes, use the displayed command.
Error 42	Unknown error has occurred.
Error 43	Cannot obtain lock on the displayed revision.
Error 44	Cannot find the displayed revision.
Error 45	Invalid argument set for the specified command.
Error 46	The displayed word is reserved and cannot be used.
Error 47	Cannot find the specified cell.
Error 48	Reserved for internal use.
Error 49	Unable to evaluate the displayed function.
Error 50	The displayed file cannot be found.
Error 51	Unable to read the specified file.
Error 52	Reached end of iterator.
Error 53	The specified layer is unsaved.
Error 54	The specified layer does not support random access.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-23. Calibre YieldServer Error Messages (cont.)

Error Code	Explanation
Error 55	The specified polygon number is invalid.
Error 56	Cannot find the specified property.
Error 57	Copy failed. Close the DFM database before copying.
Error 58	The displayed destination DFM database already exists.
Error 59	Cannot obtain lock on the displayed database.
Error 60	The data type for the displayed property is invalid.
Error 61	The specified node number is invalid.
Error 63	Cannot unload layer from memory.
Error 64	Property already exists; cannot create a new one.
Error 65	Cannot find check.
Error 67	Unable to load the specified layer because it is not present in the database.
Error 68	Unable to execute Connect statement due to bad SVRF syntax.
Error 69	Cannot use layer in Connect statement.
Error 70	RDB format not supported for read. Only the ALL CELLS format is supported.
Error 71	Top cell in RDB does not match with database top cell.
Error 72	RDB parsing error.
Error 73	Unable to read the file.
Error 74	Polygon vertex is not valid.
Error 75	Invalid polygon due to bad vertices.
Error 76	DFM database path is invalid.
Error 77	Cannot delete the DFM database master revision.
Error 78	Cannot delete the DFM database default revision.
Error 79	Duplicate vertices in the polygon.
Error 80	Unsupported operation on a read-only database.
Error 82	Input layer to command has a wrong layer configuration.
Error 83	Input to command refers to an unknown device type.
Error 84	Error during rule file compilation due to bad SVRF syntax.
Error 85	Failure acquiring licenses while executing the SVRF statement.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-23. Calibre YieldServer Error Messages (cont.)

Error Code	Explanation
Error 86	Cannot find the device referred in the command.
Error 87	All input layers do not have connectivity information.
Error 88	DFM Analyze on side layers does not support by cell, inside_of options.
Error 89	Cannot find the displayed TVF library.
Error 91	Printing a combined density side layer is not supported.
Error 208	Error in processing cross-reference data.
Error 277	The dfm::set_layout_netlist_options -trivial_pins and -seed_promoted_trivial_pins are incompatible with PERC LDL Preserve Trivial Pins.

Table 3-24. Calibre YieldServer Warning Messages

Warning Code	Explanation
Warning 1	Duplicate layers are present in the layer list. Marking iterator invalid.
Warning 2	No database changes. Nothing to do.
Warning 3	Mismatched type.
Warning 4	The displayed data will not be saved.
Warning 5	Reserved for internal use.
Warning 6	Reserved for internal use.
Warning 7	The displayed implicit layer will not be saved.
Warning 8	The displayed layer is already frozen. Make a copy of this layer to save any changes.
Warning 9	Invalid use.
Warning 10	Invalid access.
Warning 11	The displayed layer does not have connectivity information.
Warning 14	Ignoring stamping conflicts due to ambiguous clustering.
Warning 15	Configuring the layer for adding property. This means the layer is being merged.
Warning 16	Rule compilation error.
Warning 17	Devices not extracted.
Warning 18	Connectivity information not available.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-24. Calibre YieldServer Warning Messages (cont.)

Warning Code	Explanation
Warning 25	Cross-reference information is not in the database. LVS comparison must be run to generate the data.
Warning 37	LVS Annotate Devices is required in the rule file in order to use dfm::set_layout_netlist_options -annotated_devices YES.

DFM Database Error Messages

DFM database error messages are issued when there are problems with the database itself.

Table 3-25. DFM Database Error Messages

Error Code	Explanation
Error 8	Cannot open DFM database name file for write.
Error 9	Cannot write to DFM database name file.
Error 10	Cannot open DFM database for write.
Error 11	Failed to save rules to DFM database.
Error 12	Failed to save text layers to DFM database.
Error 13	Failed to save Connect layers to DFM database.
Error 14	Failed to save Device layers to DFM database.
Error 15	Failed to finish writing to DFM database.
Error 94	Bad argument format for chmod.
Error 95	Bad argument format for chmod. The invalid format is displayed.
Error 96	Value out of range for chmod.
Error 97	Cannot acquire the displayed lock.
Error 98	Cannot close the displayed directory.
Error 99	Cannot copy the displayed file.
Error 100	Cannot create the displayed DFM Database directory (the target directory for the DFM database is read-only).
Error 101	Cannot create the displayed default revision file.
Error 102	Cannot create the displayed destination file.
Error 103	Cannot create the displayed directory.
Error 104	Cannot create the displayed token.
Error 105	Cannot create the displayed hard link.
Error 106	Cannot create temporary file for lock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-25. DFM Database Error Messages (cont.)

Error Code	Explanation
Error 107	Cannot create the displayed reference file.
Error 108	Cannot create the displayed reference source file.
Error 109	Cannot create a revision in a read-only database.
Error 110	Cannot create security file. Cannot save DFM database.
Error 111	Cannot create the displayed symlink.
Error 112	Cannot delete a default revision.
Error 113	Cannot delete the displayed references directory.
Error 114	Cannot find revision to open.
Error 115	Cannot find text layer entry for the displayed layer.
Error 116	A frozen revision cannot be frozen again.
Error 117	Cannot freeze revision in a read-only database.
Error 118	Cannot acquire DB lock.
Error 119	Cannot get exclusive DB lock for the displayed DFM database.
Error 120	Cannot get exclusive lock for the displayed revision.
Error 121	The displayed revision directory does not exist.
Error 122	Cannot acquire revision lock.
Error 123	Cannot load layers before the hierarchical database is initialized.
Error 124	Cannot load rules.
Error 125	Cannot load the displayed rules.
Error 126	Cannot load rules from the displayed file.
Error 127	Cannot load rules from the displayed file.
Error 128	Cannot open the displayed database.
Error 129	Cannot open the displayed database: not 32-bit compatible. This error occurs when you generate a DFM database on a 64-bit machine and attempt to open it on a 32-bit machine.
Error 131	Cannot open the displayed directory. This error occurs when you try to create a DFM database using OVERWRITE, but the database name already exists as a file, not a directory.
Error 132	Cannot open the displayed source file.
Error 133	Cannot read the displayed default revision file.
Error 134	Cannot read symlink.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-25. DFM Database Error Messages (cont.)

Error Code	Explanation
Error 135	Cannot remove existing DFM Database directory. This error occurs when you are trying to create a DFM database using OVERWRITE, but the database name exists and is read-only.
Error 136	Cannot remove the displayed directory.
Error 137	Cannot remove the displayed file.
Error 138	Cannot remove the displayed revision directory.
Error 139	Cannot rename file.
Error 140	Cannot reset directory permissions.
Error 141	Cannot reset permissions on the displayed file.
Error 142	Cannot reset permissions.
Error 143	Cannot restore password protected rules file from the displayed token.
Error 144	Cannot stat file or directory.
Error 145	Close the current revision before deleting.
Error 146	Too much contention during creation of lock file.
Error 147	Don't know where to write the database.
Error 148	File not specified.
Error 151	The displayed layer is not in the layer TOC.
Error 152	The displayed lock file already exists.
Error 155	New revisions can be created only from frozen revisions.
Error 157	The displayed file is not a DFM database. This error occurs when you are trying to run dfm::chmod on something that is not a DFM database.
Error 158	Database was not created with a YieldServer license.
Error 159	Error reading the displayed file.
Error 160	The displayed reference source file missing.
Error 161	Revision header file is missing.
Error 162	Error executing SQL.
Error 163	Unknown file for layer.
Error 164	Unknown layer.
Error 165	Unknown layer ID.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-25. DFM Database Error Messages (cont.)

Error Code	Explanation
Error 166	Unknown text layer.
Error 167	Error writing the displayed file.
Error 173	Cannot acquire lock. This error occurs when more than one process is attempting to open the DFM Database for write.
Error 174	Cannot access existing file during linking.
Error 175	Cannot link.
Error 176	DFM Database directory already exists. This error occurs when you specify DFM Database *dfmdb* without OVERWRITE when *dfmdb* already exists.
Error 177	This error occurs when you are trying to create a DFM database using OVERWRITE, but the database name exists and is not a valid DFM database.
Error 178	Existing DFM database has more than one revision. This error occurs when you create a DFM database, add one or more revisions to it, then try to overwrite it from another -dfm run without specifying DFM Database OVERWRITE REVISIONS.
Error 229	DFM Database requires PINLOC keyword in order to use the YS option.
Error 242	The command or option is not valid in the current context.

DFM Executive Messages

DFM executive error messages are issued when there is a problem detected by the executive module. Often these are due to rule file configuration problems.

Table 3-26. DFM Executive Messages

Error Code	Explanation
Error 1	A DFM Database or DFM Database Directory specification is required with calibre -dfm.
Error 2	There are no DFM Select Check statements and no DEVICE statements.
Error 3	Device recognition cannot be executed when DRC Incremental Connect YES is specified.
Error 4	Cannot open Spice report file for output.
Error 5	Top cell is empty or contains no needed data.
Error 6	Cell coordinate data out of range.
Error 7	There is no DFM YS AUTOSTART statement.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-26. DFM Executive Messages (cont.)

Error Code	Explanation
Error 16	DFM database path cannot begin with "/" when DFM DATABASE DIRECTORY is specified.
Warning 90	There are no Device statements in the rule file; device extraction will not be performed.

DFM Create Layer Error Messages

The DFM Create Layer statement in the rule file can trigger runtime error messages as discussed here.

Table 3-27. DFM Create Layer Error Messages

Error Code	Explanation
Error 1	DFM Create Layer is supported only in YieldServer.
Error 2	DFM Create Layer ORDERED with geometries in unnamed cell and top cell; output layer will be empty.
Error 3	The displayed cell does not exist in the hierarchy.
Error 4	Nets specified for unmerged polygons in the displayed cell.
Error 5	DFM Create Layer NODAL executed with no connectivity.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Appendix A
Calibre YieldServer Example Scripts

These scripts generate iterators, step through iterators, obtain data from iterator objects, traverse hierarchy, and output information. The methods employed are applicable in solving other problems.

Example: Report Net Connections Down the Hierarchy

This script reports net connections of a specified net moving down the hierarchy from a specified cell.

A DFM database must exist and have connectivity information to use this script.

The *descend_net.ys* file contains the primary YieldServer script. This script takes two required inputs arguments: a cell name and a net name in that cell. Errors are given if the cell cannot be found or if the net argument is numeric. The out_stream argument is optional and defines the name of a report file for output. By default, output is to stdout.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Descends the hierarchy from the specified net.
set indent "            "
proc browse_hierarchy {cell_name net_number out_stream } {
  global indent
# Get the placements containing the net. Each placement will be descended.
  set place_iter [dfm::get_placements $cell_name -net $net_number]
  while { $place_iter != "" } {
# Descend the net and get the information from it, then call
# the procedure again on the descended net. This will continue
# descending and outputting the placement info until at the
# bottom of the hierarchy.
    set net_in_instance  [dfm::descend_net $place_iter]
    set node_in_instance [dfm::get_data $net_in_instance -node]
    set sub_cell_name    [dfm::get_data $net_in_instance -cell_name]
    set sub_cell_net     [dfm::get_net_name $node_in_instance \
                            -cell $sub_cell_name]
    set placement_name   [dfm::get_data $place_iter -placement_name]
    puts $out_stream "$indent |"
    puts $out_stream "$indent +--->$placement_name (cell: $sub_cell_name)\
      $sub_cell_net (node ID: $node_in_instance)"
    set indent "     $indent"
    browse_hierarchy $sub_cell_name $node_in_instance $out_stream
    dfm::inc place_iter;
  }
  set indent "            "
}
```

```
proc descend_net {args} {

# Set options and parse the arguments passed into the proc
  set options {
      { cell.arg  ""  "The leaf cell name to get the nets from" }
      { net.arg   ""  "Net number of the net to descend from" }
      { out_stream.arg  "stdout"  "Where to print output to, \
        default is stdout" }
  }

  array set params [::cmdline::getoptions args $options]

  if {[llength $args] != 0} {
    puts "Error: Invalid argument \'[lindex $args 0]\'"
  }

  if {$params(cell) eq ""} {
    puts "Error: -cell cell_name is a required argument"
    return
  }

  if {$params(net) eq ""} {
    puts "Error: -net net_name is a required argument"
    return
  }

# Numeric node IDs are not permitted.
  if [regexp {^[0-9]+$} $params(net)] {
    puts "Error: Net name cannot be numeric."
    return
  }

# Get the cell name and net name
  set cell_name $params(cell)
  set net_name $params(net)

# Create the net iterator for the cell. If that cannot occur, error out.
  if {[catch {dfm::get_nets $cell_name} net_iter]} {
    puts "Error: Unable to create net iterator for cell $cell_name :\
      $net_iter"
    return
  }

# Open file to output to if filename provided; use stdout if not.
  if {$params(out_stream) ne "stdout"} {
    if {[catch {open $params(out_stream) w} out_stream]} {
      puts "$params(out_stream) could not be opened to write to:\
        $out_stream"
      return
    }
  } else {
    set out_stream stdout
  }
# Search for the requested net in the cell.
  set cell_net_name [dfm::get_data $net_iter -net_name]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
while {$net_iter ne "" && $cell_net_name != $net_name} {
    dfm::inc net_iter
    if {$net_iter ne ""} {
      set cell_net_name [dfm::get_data $net_iter -net_name]
    }
  }

# If the net iterator has reached its end, the net is not present
# in the cell. Notify and return to the calling environment.
  if {$net_iter eq ""} {
    puts "NOTE: No net \"$net_name\" in $cell_name."
    return
  }

# Get the node ID of the net. This will be used to conduct the
# hierarchical search.
  set node_number [dfm::get_data $net_iter -node]

  puts $out_stream "Tracing net: $net_name (ID: $node_number) in cell \
    $cell_name:"

# Trace the net.
  browse_hierarchy $cell_name $node_number $out_stream

# Close output stream.
  if {$out_stream ne "stdout"} {
    close $out_stream
  }
  return
}
```

A *ys.script* file called by Calibre YieldServer executes the *descend_net.ys* script:

```
source descend_net.ys
descend_net -cell TOPCELL -net PWR
```

When this command is executed:

```
calibre -ys -dfmdb dfmdb -exec ys.script
```

this is an example of the output:

```
Tracing net: PWR (ID: 2) in cell TOPCELL:
             |
             +--->X1 (cell: CellB)  (node ID: 1)
                  |
                  +--->X0 (cell: inv) PWR (node ID: 2)
             |
             +--->X0 (cell: CellA)  (node ID: 3)
                  |
                  +--->X0 (cell: nand) PWR (node ID: 2)
```

Instance IDs, corresponding cell names, and numeric node IDs are always reported. User-given net names are also reported when they exist.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example: Report Net Instance Connections Up to a Context Cell

This script reports net connections up the hierarchy from a specified net instance to a context cell. A specified net instance path is referenced from the context cell.

A DFM database must exist and have connectivity information to use this script.

The *trace_net_up.ys* file contains the primary YieldServer script. This script takes two required input arguments: a context cell name and a net instance path. The net instance path head is located in the context cell. For example, X0/X1/Y is a net instance path. Instance X0 is located in a specified context cell, instance X1 is in instance X0, and net Y is in instance X1. Errors are given if the context cell cannot be found or if the net path does not exist in the cell. The -out_stream argument is optional and defines the name of a report file for output. By default, output is to stdout.

```
# This script ascends a net instance path in the specified
# context cell and traces the net's connections.

# This proc closes the output stream.
proc close_stream {} {
  upvar out_stream out_stream
  if {$out_stream ne "stdout"} {
  close $out_stream
  }
}

# This proc uses the path context to trace a net's connections
# to a specified context cell.
proc trace_net_up {args} {

# Set options and parse the arguments passed into the proc.
  set options {
    { path.arg             ""       "Net instance path." }
    { context_cell.arg     ""      "Cell in which the trace terminates." }
    { out_stream.arg    "stdout"   "Where to write output to. \
      Defaults to stdout." }
  }

  array set params [::cmdline::getoptions args $options]
```

```
# Check the validity of the arguments.
  if {[llength $args] != 0} {
    puts "Error: Invalid argument \'[lindex $args 0]\'"
  }

  if {$params(context_cell) eq ""} {
    puts "Error: -context_cell cell_name is a required argument."
    return
  }

  if {$params(path) eq ""} {
    puts "Error: -path net_path is a required argument."
    return
  }

# Set the net path and context cell according to input arguments.
  set net_path $params(path)
  set context_cell_name $params(context_cell)

# Use the net path to get a path context in relation to the cell.
  if {[catch {dfm::get_path_context $context_cell_name \
    -net $net_path} path_context]} {
    puts "Error:  Could not find cell $context_cell_name"
    return
  }

# Get the name of the cell the net resides in.
  if {[catch {dfm::get_data $path_context -leaf_cell_name} cell_name]} {
    puts "Error: Could not find path $net_path connected \
      to cell $context_cell_name"
    return
  }

 # Open file to output to if filename provided. Use stdout if not.
  if {$params(out_stream) ne "stdout"} {
    if {[catch {open $params(out_stream) w} out_stream]} {
      puts "Error: $params(out_stream) could not be opened to write to: \
        $out_stream"
      return
    }
  } else {
    set out_stream stdout
  }

# Get the node ID, net name (if any), name of the net's containing cell,
# and path name of the path context.
  set node_id [dfm::get_data $path_context -node]
  set net_name [dfm::get_net_name $node_id -cell $cell_name]
  set path_name [dfm::get_data $path_context -path_name]

puts ""
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Set the path reporting format for the net.
  if {$net_name ne ""} {
    set path_report "Net: $path_name/$net_name (node id: $node_id)"
  } else {
    set path_report "Net: $path_name/$node_id"
  }

# Output the net that is being examined and the cell it belongs to.
  puts $out_stream "$path_report (in cell: $cell_name)"

# Create a net path object and point it to the top level.
  set top_context [dfm::get_path_context $context_cell_name \
    -net $net_path]
  dfm::ascend_path_context $top_context -to_top
  set top_cell_name [dfm::get_data $top_context -leaf_cell_name]

# If the top of the path is the same as the net that contains the cell,
# report it. At this point, there is no more for the script to do.
  if {$cell_name eq $top_cell_name} {
    puts "$cell_name is the parent cell of the net."
    close_stream
    return
  }

# Loop through the path context iteratively.
# Find the parent cells and net instance paths along the way.
# Report net names if they are available.
# Show in branch diagram.
  set padding "            "
  while {$cell_name ne $top_cell_name} {
    dfm::ascend_path_context $path_context
    set cell_name [dfm::get_data $path_context -leaf_cell_name]
    set node_id [dfm::get_data $path_context -node]
    set net_name [dfm::get_net_name $node_id -cell $cell_name]
    set path_name [dfm::get_data $path_context -path_name]

# Set the reporting format.
    if {$path_name ne ""} {
      set full_path "$path_name/$node_id"
    } else {
      set full_path "$node_id"
    }
# If net names are available, write them.
    if {$net_name ne ""} {
      set full_path "$full_path ($net_name)"
    }

    puts $out_stream "$padding |"
    puts $out_stream "$padding +--->$full_path (cell: $cell_name)"
    set padding "     $padding"
  }
  puts ""
  close_stream
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A *ys.script* file called by Calibre YieldServer executes the *descend_net.ys* script:

```
source trace_net_up.ys
trace_net_up -path X60/A  -context_cell sat
trace_net_up -path X0/X0/13 -context_cell mx
trace_net_up -path X0/X0/VDD -context_cell mx
```

When this command is executed:

```
calibre -ys -dfmdb dfmdb -exec ys.script
```

this is an example of the output:

```
Tracing net: X60/A (ID: 2) in cell inrb8
             |
             +--->6 (SSDN) (cell: sat)

Tracing net: X0/X0/13 in cell aoi222
aoi222 is the parent cell of the net.

Tracing net: X0/X0/VDD (ID: 1) in cell aoi222
             |
             +--->X0/2 (cell: ICV_3)
                  |
                  +--->2 (cell: mx)
```

Net paths, node IDs and cells containing the nets are reported. User-given net names are also reported when they exist.

Example: Report Hierarchy

This script reports hierarchy statistics for cells and placements in the DFM database. This report can be useful when analyzing hierarchy produced by the Calibre hierarchical database constructor.

The first portion of the script lists all cells in the design, the number of hierarchical levels in each cell, and the number of times each cell is placed.

The second portion of the script prints a complete hierarchical tree of the layout design showing cell placements and the levels at which they are placed. As this output is verbose and can give large amounts of output, the header to this part of the report and the call to the write_tree proc are commented out.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#
# Calibre YieldServer script to report cell instantiations and hierarchy
# levels.
#
# Call from calibre -ys -exec option or DFM YS AUTOSTART. In the latter
# case, source it from a TVF FUNCTION block in the rules.
#
##########################################################################
# use this for pretty output
package require struct::matrix

# set this to 1 to turn on debugging trace
# set DEBUG 1;

############################## PROCESS DATABASE ##########################
# get names of all cells in the database
set cells [dfm::get_cells];

while {$cells != ""} {
  set cellName [dfm::get_data $cells -cell_name];
  lappend cellList $cellName;
  dfm::inc cells;
}

# get the total cell count
set cellCount [llength $cellList];

# test for hierarchy and exit if there is none
if { $cellCount < "2" } {
  puts "\n NOTE: Layout has no hierarchy. Nothing to do. \n";
  exit 0;
}
# put any debug diagnostics here
if { [info exists DEBUG] == 1 } {
    puts "\n---$cellCount layout cells: \n\n[regsub -all { } \
      [lsort $cellList] "\n"] \n";
};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##########################################################################
# PROC: hier_count; outputs calculates the number of hierarchical levels
# of a cell.
# cells  = child cells of a cell
# level  = hierarchical level, indexed from 0
##########################################################################
proc hier_count { cells level } {
# if this proc is called, there is an additional level of hierarchy
  incr level;
  set currentCells $cells;
  set newCells "";

# track cells at a given level
  foreach child $currentCells {
    append newCells "[dfm::list_children $child] ";
  }

  if { [lindex $newCells 0] != "" } {
    hier_count $newCells $level;
  } else {
    return $level;
  }
}

puts "\n\n            CELL HIERARCHY LEVELS AND INSTANTIATIONS"
puts "----------------------------------------------------------------\n";
# iterate over all cells.
set hierList "";

foreach cell $cellList {
  set plCount [dfm::get_placement_count $cell]
  set childCells [dfm::list_children $cell]
# if there are child cells, process to find the number of hierarchy
# levels, otherwise add to the list with hierarchy as <0>
  if { [llength $childCells] > 0 } {
    lappend hierList [list $cell "levels: [hier_count "$childCells" 0]"\
      "instantiations: $plCount"];
  } else {
    lappend hierList [list $cell "levels: 0" "instantiations: $plCount"];
  }
}

# output the cell list in ascii order
# create a 3-column matrix for printing
struct::matrix m0;
m0 add columns 3;
set sortList [lsort $hierList];
set sortListLength [llength $sortList];

for {set i 0} {$i < $sortListLength} {incr i} {
# if struct::matrix is not available, use next line for output instead
#  puts "---[lindex $sortList $i]";
  m0 add row "---[lindex $sortList $i]";
}
# output the table
m0 format 2chan;
puts "\n";
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##########################################################################
# PROC: write_tree; outputs hierarchy tree of design
# cell   = cell to begin dumping from, "" means use topcell
# indent = string to format the hierarchy
# level  = hierarchical level, indexed from 0
##########################################################################
proc write_tree { level {cell ""} {indent ""}  } {

  if { $cell == 0 } {
    puts "\n ERROR: Cannot determine top cell.";
    exit 1;
  }

# get placements in a cell
  set pl [dfm::get_placements $cell]
# write the cell placement in the tree
  puts "$indent <$level> $cell";
# adjust the indent level of the tree
  append indent "    ";
# increment the level
  incr level;
# recursively call this proc to descend all branches of the hierarchy
  while {$pl != ""} {
    set plName [dfm::get_data $pl -cell_name];
    write_tree $level $plName $indent;
    dfm::inc pl;
  }
}

# OUTPUT FROM THIS PART IS VERBOSE! UNCOMMENT TO CALL IT.
# puts "\n\n                        HIERARCHY TREE"
# puts "-----------------------------------------------------------------\n";

# call the write_tree proc. hierarchy level is 0 beginning with top cell
# write_tree 0 [dfm::get_top_cell]; # UNCOMMENT TO WRITE TREE;

puts "\n\n---Done.\n"
exit -force
}
```

Output from the first part of the report appears like this:

```
            CELL HIERARCHY LEVELS AND INSTANTIATIONS
-------------------------------------------------------------
Loading hierarchy and connectivity
---CellA   levels: 1 instantiations: 2
---CellB   levels: 1 instantiations: 1
---TOPCELL levels: 2 instantiations: 0
---inv     levels: 0 instantiations: 1
---nand    levels: 0 instantiations: 2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When enabled, the tree report appears like this (output can be extensive):

```
                        HIERARCHY TREE
---------------------------------------------------------------
 <0> TOPCELL
     <1> CellA
         <2> nand
     <1> CellA
         <2> nand
     <1> CellB
         <2> inv
```

Example: Generating a Regression Layout from a DFM Database

Calibre YieldServer scripts are frequently useful in regression flows. One such application is to generate an OASIS layout from a DFM Database. OASIS layouts have the convenience of storing layer names intrinsically in the database. Such layer names can correspond to rule checks. By using that capability, rule check output can be saved under its own layer name to a physical design for regression testing.

```
# Calibre YieldServer script for generating OASIS layout data from a
# DFM Database.
# Upstream of this script, a DFM Database file must be generated
# containing layout shapes.
# The output is an OASIS file with layers generated from rule checks in
# the DFM Database.
# Edge and error cluster data are expanded to polygons.

# path of the output file.
set out_layout "out.oas"

# check names in the database.
set checks [dfm::list_checks]

# layer info string for final layout dump command.
set linfo ""

# SVRF commands to convert edge/error output to polygon.
set svrf ""

# initial layer number for output results.
set layer_nbr 1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# generate SVRF code for writing layers.
foreach check [lsort $checks] {
  puts "  CHECK: $check"
# get related check layer
  append svrf "// check $check \n"
  set i 0
  foreach layer [dfm::list_layers -no_ids -check $check] {
    puts "\tLAYER: $layer"
    incr i
# ignore window output from density, dfm analyze etc.
    if [regexp {_detail$} $layer ] {
      continue
    }

# get layer type:
# 1: polygon
# 2: edge layer
# 3: error cluster
    set type [dfm::get_data $layer -layer_type]
    set outlayer0 $layer
    set outlayer1 $layer
    if {$type == 3} {
# converting error clusters to edges
      set outlayer0 ${check}::<$i>_edge
      append svrf "'$outlayer0' =  DFM COPY '$layer' EDGE \n"
    }
    if {($type == 3) || ($type == 2)} {
# expanding edges
      set outlayer1 ${check}::<$i>_eedge
      append svrf "'$outlayer1' =  EXPAND EDGE '$outlayer0' BY \
                                (4/\$PRECISION) \n"
    }
# attach dummy property to force layer creation even with
# concurrent alias layers.
# drv.tcl script relies on layer name format: check::<nbr>_p
    set outlayer ${check}::<$i>_p
    append svrf "'$outlayer' =  DFM PROPERTY '$outlayer1' \
                 \[check = '$check'\] \n"
    append linfo " -layer_info \{\"$outlayer\" $layer_nbr 0 \}"
  }
  incr layer_nbr
}

# if there were any non-polygon error layers, execute related svrf.
if [llength $linfo] {
  dfm::new_layer -svrf $svrf -keep_all_layers
}

# dump the output layout
eval "dfm::write_oas $linfo -file \{$out_layout\}"
exit -force
```

This script can be called from within Calibre YieldServer using the -exec command line option. A DFM Database with layout data should be loaded at the same time. The output is based upon your design layout. The output design, *out.oas*, has layer names of this form: <check_name>::<n>_p, where <check_name> is both the name of a rule check and the layer it

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

generates, and <n> is an integer. These layers correspond to check names that were active in the rule file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Index

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.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.