QuickCap® Auxiliary Package User Guide and Technical Reference

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

11. QTF Language

A. Environment Variables

Preface

This preface includes the following sections:

About This User Guide

This user guide describes tools supplied as part of the auxiliary package available with the QuickCap tool.

Audience

This user guide is for engineers who use the QuickCap and gds2cap tool to create complex systems on a chip. The readers of this user guide must be technically oriented and have some familiarity with QuickCap products.

Related Publications

For additional information about the QuickCap tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to see the documentation for the following related Synopsys products:

- QuickCap®
- gds2cap
- StarRC™

Release Notes

Information about new features, changes, enhancements, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *QuickCap Release Notes* on the SolvNet site.

To see the *QuickCap Release Notes*,

1. Go to the SolvNet Download Center located at the following address:
   https://solvnet.synopsys.com/DownloadCenter
2. Select QuickCap, and then select a release in the list that appears.

Licensing

You can use QuickCap Auxiliary Package using a single node license or a floating node license.

Single-Node License

If you are using QuickCap Auxiliary Package on a single node license, start the license manager (supplied by Synopsys). Set the appropriate FLEXlm environment variable:

```
lmgrd -c path
setenv LM_LICENSE_FILE=path
setenv QUICKCAP_LICENSE_FILE=path
```

Use **QUICKCAP_LICENSE_FILE** instead of **LM_LICENSE_FILE** that ensures you do not adversely affect any application licensed with FLEXlm but using a different license file.

For information about environment variable, see Appendix A, “Environment Variables”.

Floating-Node License

If you are using QuickCap Auxiliary Package on a floating-node license, complete either of these tasks:

- Make the license file available to all nodes in the network that need it by placing it or a copy on as many file systems as necessary, and set **LM_LICENSE_FILE** or **QUICKCAP_LICENSE_FILE** appropriately
- Set either **LM_LICENSE_FILE**or **QUICKCAP_LICENSE_FILE** to [***port***]***@host***, where ***port*** and ***host*** are from the SERVER line in the license file. The ***port*** value need not be specified if the SERVER line uses a default port.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access SolvNet, go to the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to SolvNet at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at https://www.synopsys.com/support/global-support-centers.html
  - Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at https://www.synopsys.com/support/global-support-centers.html

1. Introduction

The auxiliary package includes tools that support QuickCap NX, the Synopsys advanced 3-D Monte Carlo capacitance extractor product. This guide describes the following tools:

- cap+spef
- cap2sigma
- cap2sum
- floatCap
- gds2density
- gds2tag
- gds2text
- qtfx
- SvS

2. The *cap+spef Tool*

Overview

Format 1

cap+spef [**-license**[**s**]] [**-v**[**ersion**]] [**-verbose**] ***rootName***

Format2

cap+spef [**-license**[**s**]] [**-v**[**ersion**]] [**-verbose**] ***numericFile spefFile***

The cap+spef tool scales capacitances in a Standard Parasitic Exchange Format (SPEF) file so the total capacitance on each net agrees with results input from a .numeric file (generated using the QuickCap **-numeric** option). This provides first-order correction to RC models of nets defined in the SPEF file. For nets modeled by a single lumped capacitance, this effectively replaces capacitance values with QuickCap results.

The **-license** option causes cap+spef to the print license information. The **-version** option causes cap+spef to print the full path name of the executable and the build date, and then exit.

Operation

The format:

cap+spef [**-license**[**s**]] [**-v**[**erbose**]] ***rootName***

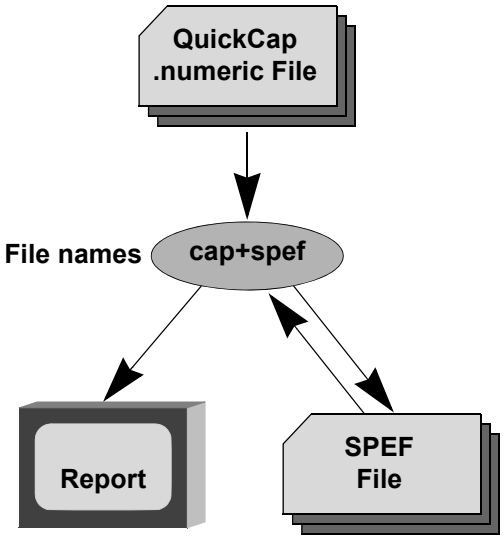
is equivalent to:

cap+spef [**-license**[**s**]] [**-v**[**erbose**]] ***rootName*.numeric *rootName*.spef**

It creates a new SPEF file with the same name, but with capacitances scaled so that the total capacitance on each net agrees with QuickCap results. Coupling-capacitance component values in the SPEF file are also scaled when the .numeric file includes the same components generated by QuickCap. See Figure 2-1. Nets for which no QuickCap results have been produced are unaffected.

The cap+spef tool backs up the SPEF file by appending a tilde (~) to the file name.

Figure 2-1: cap+spef Flow



Sample Run

The following sample run modifies c.spef according to QuickCap results in c.numeric.

```
>cap+spef c
*********************************************************************
Copyright (C) 2003-2013 Synopsys
cap+spef version 1.2.##-s (compiled time)
*********************************************************************

1234 QuickCap NETS REFERENCED IN 'c.numeric'

100 EXTRACTED QuickCap NETS IN 'c.numeric'

PROCESSING spef DATA IN c.spef

SCALING CAPACITANCE TO MATCH QuickCap RESULTS FOR 'N174397'
                  Coupled RC network was 0.8% low...
                                    ground: 5.8% low*
                                    N41520: 3.8% low
                                    N43722: 1.5% high
                                    N27177: 2.7% high
<...lines deleted from output...>
                                    N21635: 14% high
                                     N8078: 18% high
                                   N508489: 4.9% low
  *ground includes 1219 QuickCap coupling components
                                                (no SPEF component)
<the -v option would list the 1219 coupled net names here>
  and 44 SPEF coupling components (no Quickcap component)
<the -v option would list the 44 coupled net names here>
spef CAPACITANCE VALUES MODIFIED ON 1 NET

THE spef FILE DID NOT CONTAIN THE FOLLOWING EXTRACTED QuickCap NETS:
   N104311_1
   N105591
   N105600
   N10582
<...lines deleted from output...>
   N218494_1
   N219085
   N221340
   N222598
```

```
ORIGINAL spef FILE RENAMED TO 'c.spef~'

TERMINATED NORMALLY:  Mon Dec 16 13:11:15 2013
```

The cap+spef report lists the discrepancy between the original SPEF values and the QuickCap results. In this case, the original total capacitance is low by about 0.8%; although, some coupling components have a larger error. With the **-v[erbose]** option, cap+spef lists net names for any coupling components found in the QuickCap .numeric file that are not in the SPEF file, as well as for any coupling components found in the .SPEF file that are not in the .numeric file.

3. The cap2sigma Tool

Overview

Format

cap2sigma [**-license**[**s**]] [**-v**[**ersion**]] [***options***] ***file1 file2***

The cap2sigma tool searches for statistically significant differences between two QuickCap .numeric output files.

When the **-version** option is specified, cap+spef prints the full path name of the executable and the build date and exits.

The **-license** option causes cap2sigma to print the license information. The **-version** option causes cap2sigma to print the full path name of the executable and the build date, and then exit.

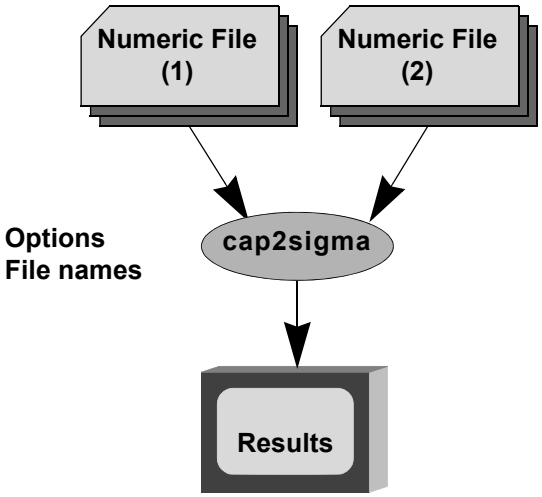
Operation

For each pair of capacitance values found in the two .numeric files (generated using the QuickCap **-numeric** option or generated by cap2sum), cap2sigma tallies the difference (a statistical estimate of the value 0), normalized to the normal (statistical) error of the difference. This value, called *sigma*, is a measure of the discrepancy between the two values. See Figure 3-1 on page 3-2.

If you specify the **-list** command-line option, all sigmas are listed. The cap2sigma tool also shows the statistics and a histogram of the sigmas, which should be consistent with a normal distribution (a bell curve) that has an average of 0 and a standard deviation of 1. The **-crit *value*** option causes cap2sigma to exit with a status of 2 when the worst absolute sigma is larger than ***value***. A variant, **-crit *pct***, generates a critical value that yields a *false negative* (exceeding the critical value when there is no statistically significant error) a given fraction of the time, ***pct***.

If no data is found, cap2sigma exits with a status of 1.

Figure 3-1: cap2sigma Flow Diagram



Sigma is the difference between two values normalized to the statistical error of the difference. Following are the two results:

$$a_1 \pm \sigma_1 \text{ and } a_2 \pm \sigma_2$$

sigma is calculated by:

$$\frac{a_1 - a_2}{\sqrt{\sigma_1^2 + \sigma_2^2}}$$

For example, the results of 90±10 and 100±10 have a difference of 10±14.1, or a sigma of 0.7. These are not in disagreement. Results of 90±2 and 100±2 have a difference of 10±2.82, or a sigma of 3.5. By itself, this is statistically significant. However, if this is the worst sigma of 1,000 trials, it is *not* significant.

Options

The following options are available for the cap2sigma tool:

-all
Computes sigmas for all values in either .numeric file. When a capacitance component is defined in one file but not in the other, the missing component value is treated as 0 +/- 0. In the absence of the **-all** option, cap2sigma does not report a sigma for such a case.

-crit *value*
-crit *pct*
Causes cap2sigma to exit with a status of 2 when the largest absolute sigma exceeds ***value***. The normal exit status is 0.

In the second form, when the argument ends with a percent sign (%), cap2sigma selects a critical ***value*** that has a chance ***pct*** of yielding a false negative. (For a false negative, the critical value is exceeded even though there is no bias.) The selected critical value is a function of the number of elements compared. For example, with **-crit 5%**, approximately one in twenty runs exits with a status of 2 even though the inputs are statistically equivalent, assuming the statistical error has a normal (gaussian) distribution.

The summary generated by cap2sigma includes both the limiting sigma ***value***, and the corresponding probability of a false negative ***pct***. The histogram includes a dotted line marking ***value*** (see Example 3-2).

-list
Includes in the output a list of sigmas sorted by absolute value. See the sample shown on page 3-4.

-nEff
When calculating statistics of sigma (average and standard deviation), counts a coupling-capacitance component as a fraction of a net according to the fraction of the capacitance (coupling capacitance divided by total capacitance). This is useful when different numeric files include different approximations to small coupling capacitance terms, such as in partitioned QuicKCap run.

-skipGround
Does not compare ground caps. Use the **-skipGround** option to check cell-based analysis and to check partition runs when looking at coupling capacitance.

In cell-based analysis, the QuickCap deck includes statements **xCell ...** [**n=*n***] or **yCell ...** [**n=*n***]. The **n** value indicates the number of cells for QuickCap to maintain individual coupling-capacitance statistics. Increasing **n,** increases the coupling capacitance, and decreases the amount of ground capacitance. To compare QuickCap runs with different

values of **n**, use the cap2sigma option **-skipGround**, and avoid using the **-all** option, as it causes cap2sigma to compare coupling capacitances in one numeric file to zero, when they are not in the other numeric file.

Sample Runs

The following cap2sigma run compares run1a.numeric and run1b.numeric. Because of the **-list** option, cap2sigma lists individual sigmas, sorted by magnitude. Then, it summarizes the result (average, standard deviation, and worst case) followed by a histogram of the absolute values (see Example 3-1). This run terminates with an exit status of 0.

Example 3-1:cap2sigma Output With -list

```
>cap2sigma -list run1a.numeric run1b.numeric
*****************************************************************
Copyright (C) 2003-2013 Synopsys
cap2sigma version 1.2.##-buildPlatform (compiled time)
*****************************************************************
COMPARISON OF NUMERICAL DATA:
  run1a.numeric
  run1b.numeric

SIGMAS:
   Net.3   Net.3: +0.050
   Net.6   Net.6: -0.131
   Net.2   Net.6: +0.176
   Net.3   Net.4: -0.178
  GROUND   Net.6: +0.278
   Net.4   Net.5: +0.306
   Net.5   Net.5: -0.370
  GROUND   Net.5: +0.518
   Net.1   Net.4: -0.551
  GROUND   Net.3: +0.566
   Net.1   Net.5: -0.650
   Net.1   Net.6: +0.667
  GROUND   Net.4: -0.794
   Net.2   Net.3: -0.817
   Net.2   Net.4: -0.861
   Net.4   Net.4: -0.894
   Net.1   Net.3: -0.903
   Net.2   Net.2: +1.071
  GROUND   Net.2: +1.083
   Net.3   Net.6: +1.319
```

```
   Net.4   Net.6: +1.485
   Net.1   Net.1: +1.547
   Net.1   Net.2: -1.585
   Net.3   Net.5: +1.590
   Net.2   Net.5: +1.852
   Net.5   Net.6: -2.250
  GROUND   Net.1: +2.607

SUMMARY OF 27 SIGMAS
    Average sigma: 0.19 +/- 0.215
 Std Dev of sigma: 1.12
      Worst sigma: 2.61

HISTOGRAM OF abs(sigma):
  6 |*   *
  5 |  *
  4 |        *
  3 |      *
  2 |
  1 |          * * *
    0-----1-----2-----3
>echo $status
0
>
```

The **-crit** option (see “Options” on page 3-3) is useful for scripting. It generates an exit status of 2 when the worst sigma is larger than the critical value, ***value***. Due to the statistical nature of the results, for a given number of data points, there is a chance, ***pct***, of a false negative, where the worst sigma exceeds the critical value even though the capacitance results have no bias. Because the worst sigma is 2.607, the following run (with **-crit 2.5**) generates an exit status of 2. The probability of a false negative is 29%, as reported by cap2sigma. Thus, for 27 data points **-crit 2.5** is equivalent to **-crit 29%**.

Example 3-2:cap2sigma Output With -crit

```
>cap2sigma -crit 2.5 test.numeric gold.numeric
****************************************************************
Copyright (C) 2003-2013 Synopsys
cap2sigma version 1.2.##-buildPlatform (compiled time)
****************************************************************

COMPARISON OF NUMERICAL DATA:
  run1a.numeric
  run1b.numeric
```

```
SUMMARY OF 27 SIGMAS
    Average sigma: 0.190 +/- 0.215
 Std Dev of sigma: 1.119
      Worst sigma: 2.607 (1 VALUE EXCEEDS -crit LIMIT)
            -crit: 2.500 (limit) or 29% (chance of a false negative)

HISTOGRAM OF abs(sigma):
  6 |*   *          .
  5 |  *            .
  4 |        *      .
  3 |      *        .
  2 |               .
  1 |          * * *
    0-----1-----2-----3
>echo $status
2
>
```

Note that the cap2sigma exit status with **-crit 3** is 0. When selecting a critical value, keep in mind that even without any other source of error, about 1 in 20 QuickCap results differ by more than two sigma due to statistical error, about 1 in 350 differ by more than three sigma, and about 1 in 15,000 differ by more than four sigma. When comparing 27 results, in Example 3-2, **-crit 3.5**, more reasonable, is equivalent to **-crit 1.2%**, triggering an exit status of 2 about 1 time in 80.

4. The cap2sum Tool

Overview

Format 1

cap2sum [**-license**[**s**]] [**-v**[**ersion**]] [***options***] ***file1*** [***file2*** [***file3*** …]]

Format 2:

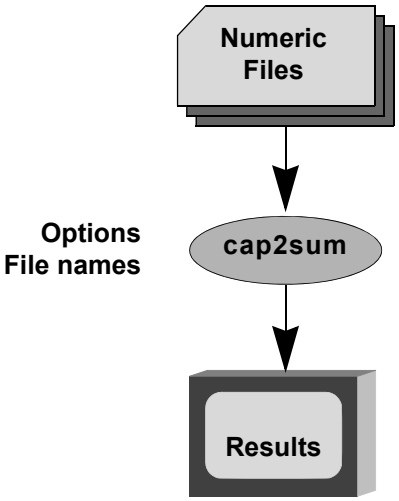
cap2sum [**-license**[**s**]] [**-v**[**ersion**]] [***options***] **+** ***file1*** [**+|-** ***file2*** [**+|-** ***file3*** …]]

The **-license** option causes cap2sum to print the license information. The **-version** option causes cap2sum to print the full path name of the executable and the build date, and then exit.

Depending on whether you specify **-avg**, the cap2sum tool sums or averages a number of independent .numeric output files (generated using the QuickCap **-numeric** option or generated by cap2sum). With the **-enumerate** option, results from the input files are enumerated to concatenate all results in a single file. With the **-spice** option, cap2sum generates a netlist file. The -**spice** option cannot be used to incrementally update an existing netlist. Without the **-spice** option, a .numeric file is generated.

In the second format, when **+** is specified as the first command-line argument after any option, each file name must be preceded by a plus or minus sign (**+** or **-**). The cap2sum tool then performs the specified operation when combining files. This format is not compatible with the **-avg** command-line options. This format can be used, for example, to calculate the device capacitance based on a numeric file containing the total capacitance (parasitic and device) and a numeric file containing the parasitic capacitance.

Figure 4-1: cap2sum Flow Diagram



Options

-asymmetric
Maintains matrix asymmetry when adding numeric files containing asymmetric results. By default, after adding up asymmetric results, cap2sum then averages the symmetric elements.

-avg
Calculates the average of the results. Averaging is useful for combining independent QuickCap results of the same capacitance problem.

-enumerate
Enumerates net names according to which file they are in. The **-enumerate** option is useful for merging results from different QuickCap runs that use the same name for extracted nets. The **-enumerate** option cannot be used with **-avg**.

-spice
Generates output in a netlist format instead of a numeric file.

Averaging Results

Averaging (using the **-avg** option) is useful for combining independent QuickCap results of the same capacitance problem.

Because any weighted average can be used, cap2sum uses the weighted average that minimizes the statistical error. Consider several QuickCap results from the same problem (each run with the **-0** QuickCap option):

$$a_1 \pm \sigma_1,\ a_2 \pm \sigma_2,\ \ldots,\ a_n \pm \sigma_n$$

The optimally weighted average is as follows:

$$\frac{1/\sigma_1^2 + a_2/\sigma_2^2 + \ldots + a_n/\sigma_n^2}{1/\sigma_1^2 + 1/\sigma_2^2 + \ldots + 1/\sigma_n^2} \pm \sqrt{\frac{1}{1/\sigma_1^2 + 1/\sigma_2^2 + \ldots + 1/\sigma_n^2}}$$

This result is applicable only when the various results are independent: no result is derived from another, and the results do not include correlated error (bias).

5. The floatCap Tool

Overview

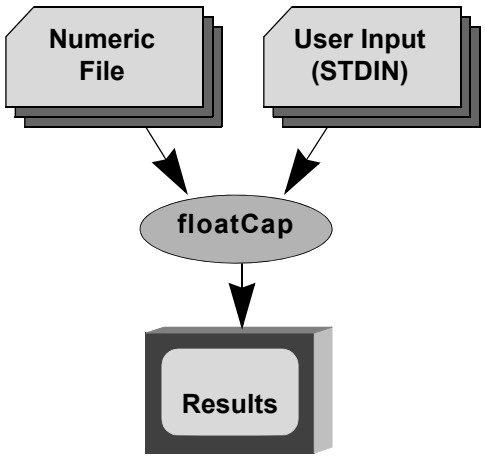
Format

floatCap [**-license**[**s**]] [**-v**[**ersion**]] ***file***

The **-license** option causes floatCap to print the license information. The **-version** option causes floatCap to print the full path name of the executable and the build date and then exit.

The floatCap tool modifies a capacitance matrix from a numeric file (generated using the QuickCap **-numeric** option or generated by cap2sum) by floating user-specified nets. (The charge on a floating net is independent of the voltages on other nets.) You can use the floatCap tool to confirm that QuickCap floating-net parameters do not introduce too much error.

Figure 5-1: floatCap Flow Diagram



Operation

The floatCap tool reads the capacitance data in the .numeric file and prints the capacitance matrix to the terminal. It then prompts you to enter the names of nets to float until an empty line is returned. Each time you specify a net to float, floatCap calculates the new capacitance matrix for the specified net, in addition to nets previously specified as floating.

Consider capacitance-matrix results in a form similar to that generated in a QuickCap summary file:

$$
\begin{array}{llll}
C_{11} & C_{12} & \ldots & C_{1F} \\
C_{21} & C_{22} & \ldots & C_{2F} \\
\ldots & \ldots & \ldots & \ldots \\
C_{F1} & C_{F2} & \ldots & C_{FF}
\end{array}
$$

$$
\begin{array}{llll}
C_{01} & C_{02} & \ldots & C_{0F}
\end{array}
$$

If net F is floating, these results can be simplified to:

$$
\begin{array}{lll}
(C_{11} - C_{F1}C_{1F} / \ C_{FF}) & (C_{12} + C_{1F}C_{F2} / \ C_{FF}) & \ldots \\
(C_{21} + C_{2F}C_{F1} / \ C_{FF}) & (C_{22} - C_{F2}C_{2F} / \ C_{FF}) & \ldots \\
\ldots & \ldots & \ldots
\end{array}
$$

$$
\begin{array}{lll}
(C_{01} + C_{0F}C_{F1} / \ C_{FF}) & (C_{02} + C_{0F}C_{F2} / \ C_{FF}) & \ldots
\end{array}
$$

Self-capacitance decreases and coupling capacitance increases in accordance with coupling to the floating net and the total capacitance of the floating net.

Option

The floatCap tool recognizes the following option:

-numeric
: Displays the capacitance matrix in numeric format, which can be recognized by the cap+spef, cap2sigma, and cap2sum tools.

Sample Run

In a sample QuickCap deck, testFloat.numeric, net F is between N1 and N2 and is symmetric with a QuickCap floating net that is between N2 and N3. QuickCap is run for a minute with **-matrix** and **-numeric**.

A subsequent floatCap run, shown here, floats F mathematically.

```
>floatCap testFloat.numeric
*****************************************************************
Copyright (C) 2003-2013 Synopsys
floatCap version 1.2.##-l<buildPlatform>(compiled <time>)
*****************************************************************

                        N1               F              N2              N3
         N1 99.18aF(0.46%) 18.38aF (1.4%)  1.58aF (5.6%) 0.3419aF (6.6%)
          F 18.38aF (1.4%) 103.1aF (0.5%) 18.02aF (1.6%) 0.7633aF (4.4%)
         N2  1.58aF (5.6%) 18.02aF (1.6%)   101aF(0.59%) 4.593aF (1.8%)
         N3 0.3419aF (6.6%) 0.7633aF (4.4%) 4.593aF (1.8%) 96.01aF(0.23%)

     GROUND 78.58aF(0.66%) 66.82aF(0.89%) 76.33aF(0.87%) 90.29aF(0.26%)
----------------------------
Net to float? F
                        N1              N2              N3
         N1 95.91aF(0.49%) 4.792aF   (3%) 0.478aF   (5%)
         N2 4.792aF   (3%) 97.84aF(0.62%) 4.726aF (1.7%)
         N3 0.478aF   (5%) 4.726aF (1.7%)    96aF(0.23%)

     GROUND 90.48aF(0.68%) 88.01aF(0.85%) 90.78aF(0.26%)
----------------------------
Net to float?
```

With F floating, the capacitance values of N1 and N3 should be the same except for:

- Statistical differences
- Approximations introduced by QuickCap to model the floating net

Mathematically floating F, in the previous sample run, shows that in this case the QuickCap results on a floating net are indistinguishable from extracting the net and simplifying the matrix.

6. The gds2density Tool

Overview

Format 1:

gds2density [**-license**[**s**]][**-v**[**ersion**]] [***options***] ***gdsFileName strName data***

gds2density [**-license**[**s**]][**-v**[**ersion**]] [***options***] ***txtFile data***

Format 2:

gds2density -redo ***fileName***

According to data on the command line, the gds2density tool either generates:

- a single density map of one or more specified layers in the form of an xy table of values over a uniform grid

or

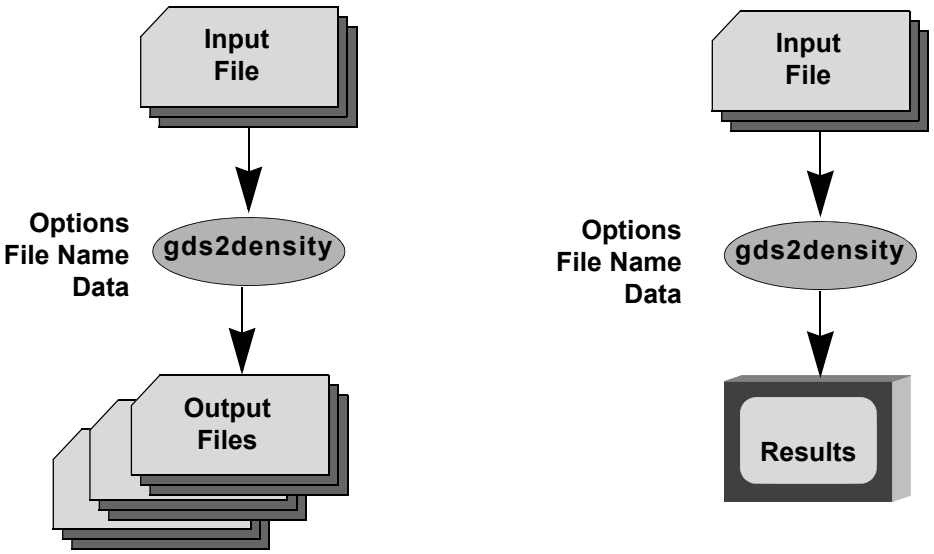
- density maps to separate files, each map based on a single layer.

The format of the gds2density output is recognized by gds2cap as a table. Unless you specify the **-binary**, **-pt** or **-rect** option on the command line, the table data is tab delimited and is compatible with spreadsheet programs.

The **-license** option causes gds2density to the print license information. The **-version** option causes gds2density to print the full path name of the executable and the build date and then exit.

Figure 6-1 on page 6-2 shows the two possible flow diagrams. When ***data*** includes a character string that allows gds2density to generate a layer-name dependent file name, gds2density generates an output file for each layer. Otherwise, gds2density outputs a single density map to the standard output.

Figure 6-1: gds2density Flow Diagram, Depending on Data



The gds2density tool can process a .txt file that is generated by gds2cap or manually, or a GDSII file, according to the command line. See “Input File” on page 6-5. The method gds2density uses to generate the grid and the table data is described in “Procedure” on page 6-3. You specify the grid size and calculation window using the options described in “Options” on page 6-7. You use command-line data to specify the layers to be analyzed and any fringe (a description of the region around the layout data), as described in “Data” on page 6-9.

The gds2density tool does not remove overlap. To avoid double counting, you can use gds2density on a .txt file generated by gds2cap. The gds2cap tool generates .txt data with overlap removed when the gds2cap technology file includes layer definitions with the **txtFile** property.

The gds2density output includes, as a comment, the command line that was used to invoke gds2density. This allows a run to be executed again. For example:

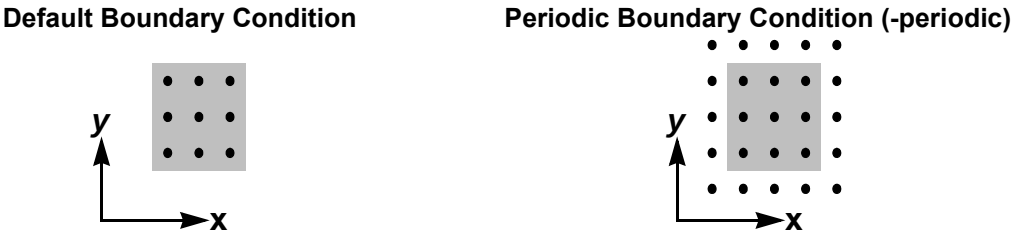
```
>gds2density -redo myFile.M2.density >myFile.M2.DENSITY
```

Be careful not to redirect STDOUT to the same file named as the argument to **-redo**, because this might erase the file before gds2density determines the original gds2density command.

Procedure

Specify the type of input file to use with the **-gds** or **-txt** option (see "Input File" on page 6-5). The grid size is determined by the **-grid** option (see "Options" on page 6-7), which specifies a maximum grid size. Without **-grid**, you must specify **-window**, which is used for the maximum grid size. (The smallest window is used if you specify multiple windows.) In the absence of padding (**-pad** or **-padTopRight**) the grid spacing is decreased independently in *x* and in *y*, as necessary, so that the layout edges (plus any fringe) are half a grid spacing outside the first and last grid points (default boundary conditions). For periodic boundary conditions (**-periodic**), two extra rows and columns are added so that layout edges (plus any fringe) are halfway between the first two and the last two grid points. See Figure 6-2. If **-pad** or **-padTopRight** is specified, gds2density preserves the user-defined grid size and either centered on the layout (**-pad**) or placed so the bottom-right corner of the layout is half a grid spacing outside the first grid point (default boundary conditions) or halfway between the first two grid points (**-periodic**).

Figure 6-2: Grid Points and Layout (Shaded) for Default and Periodic Density Calculations



The *bounds* of the layout is the smallest rectangle that includes all data in the input file, independent of the layers that are processed. Any fringe is specified by data on the command line (see "Data" on page 6-9).

A calculation region is associated with each grid point. The density associated with a grid point is the total polygonal area of GDSII objects within the associated calculation region divided by the area of the region. Calculation regions associated with different grid points can overlap. For example, density based on a 50µm window can be calculated on a 10µm grid to ensure smooth results.

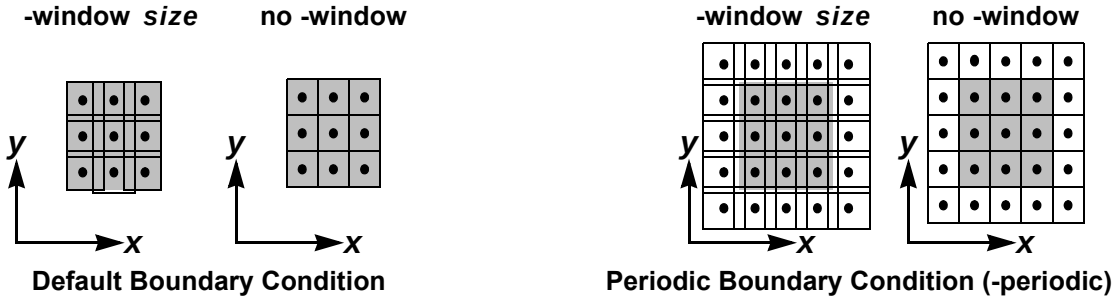
GDSII or .txt objects are included in the calculation only if they are on a layer specified on the command line (see "Data" on page 6-9). For a .txt input file, however, if the command line specifies the format of the output file ([***prefix***]**%s**[***suffix***]), gds2density generates a separate output file for each layer named on the command line. If [***prefix***]**%s**[***suffix***] is specified and *no* layers are named on the command line, gds2density generates a separate output file for each layer in the .txt file.

The gds2density tool does *not* check for polygon overlap. Although the density *should* be between 0 and 1, it can exceed 1 if the region contains overlapping objects. In general, GDSII data includes overlap. A .txt file generated by gds2cap does not have overlap.

By default, gds2density calculates the density considering the area outside the layout (plus any fringe) to be empty. The **-periodic** option, however, invokes *periodic* boundary conditions. For periodic boundary conditions, gds2density calculates the density considering the area outside the layout (plus any fringe) to be a translated copy of the layout plus fringe. To represent the periodic nature of the density map, gds2density tool adds a first and last row and column of grid points. The density value at the first grid point is the same as the density value at the next-to-last grid point, and the value at the last grid point is the same as the value at the second grid point If padding (**-pad** or **-padTopRight**) and using default boundary conditions (not periodic), the calculation window is clipped to the boundary of the layout. Calculation regions associated with different grid points can overlap.

As illustrated in Figure 6-3, when you specify **-window**, the calculation region associated with any grid point is a square of the specified size, or multiple squares of the specified sizes when the argument to the **-window** option defines multiple windows. If you do not specify **-window**, the calculation region associated with a grid point is the region nearer to that grid point than to any other. A rectangle with edges halfway to adjacent grid points.

Figure 6-3: Density Calculation Regions



If padding (**-pad** or **-padTopRight**) and using default boundary conditions (not periodic), the calculation windows of outside grid points is clipped to the boundary of the layout. The window associated with an interior grid point is not clipped. This can lead to erroneous density values when the window size (**-window**) is more than twice the grid size (**-grid**).

The gds2density tool might generate erroneous density values when all of the following conditions are met: default boundary conditions (not **-periodic**), padding (**-pad** or **-padTopRight**), and the window (**-window**) is more than twice the grid size (**-grid**).

Input File

The gds2density tool can process a .txt file, a gzipped .txt file, or a GDSII file, according to arguments on the command line. QuickCap generates the .txt file. The format of the command line and data on the command line is different for the .txt and GDSII input files.

Determining .txt or GDSII Input

To determine whether the input file is .txt or GDSII format, gds2density performs the following tests in the order listed:

1. Checks the **-gds** or **-txt** option, which indicates that the input file is GDSII or .txt data, respectively.
2. Checks the suffix of the file name.
   A. If the file name ends with .txt or .gz, the file is considered .txt data or as gzipped .txt data.
   B. If the file name ends with .gds (with up to two additional characters), the file is considered GDSII data.
3. Attempts different suffixes.
   A. If a file named ***root***.gds is found, that file is processed as GDSII data.
   B. If a file named ***root***.txt or ***root***.txt.gz is found, that file is processed as .txt data. or as gzipped .txt data.

If none of these tests to determine the data format is successful, gds2density terminates with an error.

Command-Line Syntax and Data for GDSII Input

Format: **gds2density** [**options**] ***gdsFileName structName data***

To process a GDSII file, the gds2density tool requires that the structure to be analyzed is named explicitly, and requires that each layer to be considered is specified by a layer ID. For information about formats for layer IDs and fringe data, see “Data” on page 6-9.

For GDSII input, the table is always output to standard output, which should be redirected to a file. For example:

```
>gds2density -grid 100 myFile.gds TOP 20:1 21 >myFile.M2.density
```

This finds the summed density of layers 20 (data type 1) and 21 (any data type) over a grid with a spacing of up to 100 µm. No polygon overlap is removed.

Command-Line Syntax and Data for .txt Input

Format: **gds2density** [**options**] ***txtFile data***

To process a .txt file, the gds2density tool requires that layers in the data section of the command line are specified by the same names used in the .txt file. When the command line includes data for formatting the name of an output file, output is directed to one or more files, rather than to the standard output. See the description of [***prefix***]**%s**[***suffix***] on page 6-10. With [***prefix***]**%s**[***suffix***] included as data, there is no need to specify layer names. In this case, an output file is generated for each layer found in the .txt file.

Encrypted .txt Data

The gds2density tool reads encrypted .txt files (containing **#hidden**). Such files are generated by gds2cap when xy data is to be hidden from the user. The gds2density tool processes encrypted data only if all lengths defined on the command line are round values: all grid and window sizes (defined by the **-grid** and **-window** options) must be multiples of 1 (µm), and all fringes (defined by ***fringe@density*** command-line data) must be multiples of 0.5 (µm). This restriction limits the amount of positional information you can extract from the density data.

The following environment variable supports character-based decryption, which might be available from some vendors:

setenv QUICKCAP_VENDOR_CHAR_CRYPT *DLL(s)*
: Specifies full path names of DLLs (dynamically linked libraries) for including character-based decryption. Multiple DLLs can be separated by commas or spaces. The DLLs for character-based decryption are described in the vendor-supported encryption and decryption section in the *gds2cap User Guide and Technical Reference.*

Enhanced .txt Data

The gds2density tool allows a text-file command not generated or recognized by gds2cap.

#include *fileName*
: Includes the named file. This is useful for combining text files that involve different windows.

Options

The gds2density tool recognizes the following options.

-binary
Outputs the table data in binary format. This option results in more compact data for faster processing. The **-binary** option is advised for large maps (100 by 100 tables, for example). However, the binary output cannot be easily imported into a spreadsheet program.

-gds
Indicates that an input file is in GDSII format, regardless of the suffix of the input-file name.

-grid *size*
Specifies the maximum grid spacing (in µm). The grid size is decreased, as necessary, (independently in *x* and *y*) to be uniform across the layout and so that each layout edge (plus any fringe) is halfway between the first two and last two grid points. To process an encrypted .txt input file (containing **#hidden**), the grid size must be an integer.

When you specify **-window**, the value at a grid point is the density within a square region. If you do not specify **-window**, the value at a grid point is the density over the rectangular region that goes halfway to adjacent grid points.

When you do not specify **-grid**, you must specify **-window**. In this case, the grid size is the window size (the smallest window if multiple windows are specified) or smaller.

-group *size*
Specifies the maximum size (in µm) for a hierarchical calculation. The density and bounding box is found for structures that are smaller than ***size*** in both *x* and *y*. Then, on encountering a structure reference, gds2density models the instance as a single rectangle with uniform density, rather than flattening it. The **-group** option is ignored for .txt input files.

-pad
Expands the bounds of the layout data symmetrically such that the grid size in *x* and *y* exactly matches the values specified by **-grid**.

-padTopRight
Expands the bounds of the layout data along the top and right edges such that the grid size in *x* and *y* exactly matches the values specified by **-grid** while preserving the bottom-left coordinates (minimum x and y). The **-padTopRight** option takes precedence over **-pad** when both the options are specified.

-periodic
Wraps any calculation window that extends beyond the edge of the layout (plus any fringe) to the opposite edge. This is equivalent to calculating the density in a periodic environment.

The gds2density *default* behavior (without **-periodic**) clips calculation windows to the edge of the layout plus any fringe. For example, a window 50μm wide at a grid point that is 20μm from the right side of the layout extends 20μm to the right rather than 25μm.

-pt

Generates a point-based list rather than a numeric-table format. Each line in a point-based list consists of the x and y values specifying a grid point, and a third value specifying the density.

xGrid yGrid density

The **-binary** option has no effect when **-pt** is specified.

-rect

Generates a rectangle-based list rather than a numeric-table format. Each line in a rectangle-based list consists of the x and y values of the lower left and upper right corners of a rectangle centered on a grid point, and a fifth value specifying the density.

xLowerLeft yLowerLeft xUpperRight yUpperRight density

The **-binary** option has no effect when **-rect** is specified.

-section *name*

Specifies the name of the section of a text input file to read. This option allows different sections of a single input file to be processed by separate gds2density runs. A section in the text file begins with a line of the form ***section *name***.

-txt

Indicates that an input file is in.txt format, regardless of the suffix of the input-file name.

-window *size0*[,*wgt0*[,*size1*[,*wtg1*[,..]]]]

Specifies the window sizes (in μm) for the calculation domain associated with each grid point. Each weight is a value (such as 0.4) or a percentage (such as 40%). The density value at a grid point is a weighted average of the density within each window. All window sizes must be at least as large as the grid spacing. To process an encrypted .txt input file (containing **#hidden**), all windows sizes must be integers.

To specify multiple windows, the argument list consists of window sizes alternating with weights (up to 1 or 100%), separated by commas, and without embedded spaces. When the argument list ends with a window size, that window size receives a weight so that the total weight is 1 (100%). In this case, the sum of the specified weights must be less than 1 (100%). When the argument list ends with a weight, the sum of all weights must be 1 (100%).

When you do not specify **-window**, the value at a grid point is the density over the area halfway to adjacent grid points. The height and width of the calculation window are not necessarily the same as each other in this case, because the grid is adjusted to exactly cover the layout (plus any fringe) independently in *x* and *y*.

When you do not specify **-grid**, you must specify **-window**. In this case, the grid size is the smallest window size or smaller.

-

This option, which must be the *last* option specified, allows the file name to begin with a dash (-).

Data

Format: **gds2density** [**-license**[**s**]][***options***] ***fileName*** [***structName***] ***data***

Data after the structure name specifies the layer or layers to be included in the density calculation, and any fringe condition. Unless the input is a .txt file and [***prefix***]**%s**[***suffix***] is specified, you must specify at least one layer. Each item of data is interpreted according to its format.

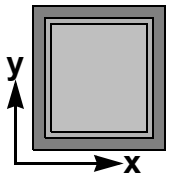
fringe@density

Just outside the bounding box of the data (and any previously defined fringe), add a fringe of width ***fringe*** (in µm) and density ***density***,which can either be a numeric factor between 0 and 1, or a percentage between 0% and 100%. This is useful for modeling scribe lines (between chips on the wafer). Because the layout plus any fringe is considered periodic, only half of a scribe line should be described. You can define multiple fringes. The order in which you define fringes is important because each specified fringe is outside any previously specified fringes.

For the tool to process an encrypted .txt input file (containing **#hidden**), each fringe must be a multiple of 0.5 (µm), and the total fringe (twice the sum of the fringes) must be an integer multiple of the grid size and all window sizes. Figure 6-4 shows two fringes.

In the absence of scribe lines, to avoid any effects of periodicity, a fringe of at least half the window size can be specified with zero density. For **-window 50**, for example, a fringe of **25@0** suffices.

Figure 6-4: Layout (Center) and Two Fringes



[***prefix***]**%s**[***suffix***] *.txt input file*

For each layer specified on the command line (or for each layer in the .txt input file), create an output file named according to the layer name, with an optional prefix and suffix. This option is recognized only for .txt input files.

layerID *GDSII input file*

Any GDSII elements with the specified layer ID (integer) are included in the calculation, whether or not any data type is associated with the element.

layerID:- *GDSII input file*

Any GDSII elements with the specified layer ID (integer) but without any data type are included in the calculation. Note that “no data type” is *not* equivalent to “data type 0”.

layerID:***dt*** *GDSII input file*

Any GDSII elements with both the specified layer ID (integer) and the specified data type (integer) are included in the calculation.

layerID:***dt0-dt1*** *GDSII input file*

Any GDSII elements with both the specified layer ID (integer) and a data type in the specified range (integers) are included in the calculation.

layerName *.txt input file*

Any .txt elements on the specified layer ID (integer) are included in the calculation. When [***prefix***]**%s**[***suffix***] is specified, gds2density generates an output file for each layer according to the layer name, with an optional prefix and suffix and containing density data based on that layer. Otherwise, a single output file is generated (on STDOUT) with density calculations based on all layers.

7. The gds2tag Tool

Overview

Format

gds2tag [**-license**[**s**]][[***options***] [**-v**[**ersion**]] ***fileIn*** [***fileOut***]]

The gds2tag tool translates a GDSII layout (file) to a TAG-formatted file, or retranslates a TAG-formatted file. The TAG format is a Synopsys-proprietary format that can be input more efficiently than a GDSII format file. The gds2cap tool can read a TAG-formatted file faster than a GDSII file, especially in a windowed (**-window**) run. The command line specifies the file name, an optional structure name, and any options. The gds2tag tool infers an input file type based on its contents.

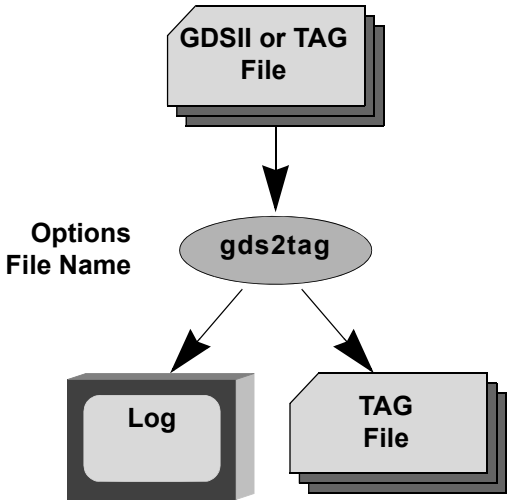
The **-license** option causes gds2tag to print the license information. The **-version** option causes gds2tag to print the full path name of the executable and the build date and then exit.

When an output file is not named, gds2tag derives the name by replacing a .gds suffix or a .agf suffix (annotated GDSII) by .tag. The .gds suffix can include up to two additional characters. When the input file name has no recognized suffix, gds2tag appends .tag to the entire input file name.

When an output file already exists, gds2tag renames it by appending a tilde (**~**) to the name. If the input file and the output file have the same name (when tagging a file with a .tag suffix, for example), gds2tag uses the renamed input file. For example, for **gds2tag test.tag**, gds2tag renames the input file to test.tag~ and creates a new tag file named test.tag.

You can filter layers by using the **-keep** or **-skip** option.

Figure 7-1: gds2tag Flow Diagram



Options

-keep *layerID(s)*
Retains only the data that contains the specified layer IDs. You can delimit multiple layer IDs by using commas (without spaces). This option is not compatible with the **-skip** option

Note: The gds2tag tool can only filter data by layer ID, and not by the data type.

-skip *layerID(s)*
Skips data that contains the specified layer IDs. You can delimit multiple layer IDs by using commas (without spaces). This option is incompatible with the **-keep** option.

Note: The gds2tag tool can only filter by layer ID, not by data type.

-v[**erbose**]
Generates additional output.

Sample Run

The following sample run translates a GDSII file test.gds to a TAG-formatted file, test.tag.

```
>gds2tag test.gds
**********************************************************************
Copyright (C) 2003-2013 Synopsys
gds2tag version 1.2.##-buildPlatform (compiled time)
**********************************************************************

INPUT FILE (gds): test.gds

  GDS+001: OPENING INPUT FILE: "test.gds"
  GDS+001: CATALOGING gds STRUCTURES IN "test.gds"...
  GDS+001: CALCULATING ALL STRUCTURE BOUNDS...
  GDS+001: CONVERTED test.gds TO tag FILE test.tag

TERMINATED NORMALLY:  Mon Dec 16 13:36:47 2013
```

8. The gds2text Tool

Overview

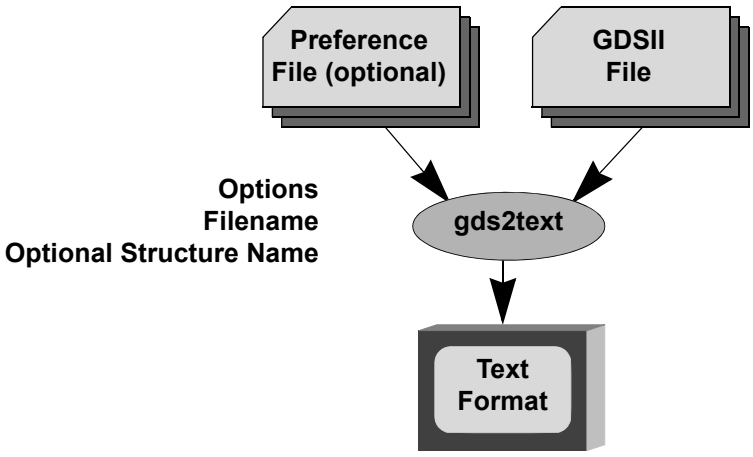
Format

gds2text [**-license**[**s**]] [**-v**[**ersion**]] [***options***] ***file*** [***structure***]

The gds2text tool formats GDSII (binary) data as text and is useful for checking the contents of a GDSII file. The command line specifies the file name, an optional structure name, and any options.

The **-license** option causes gds2text to print the license information. The **-version** option causes gds2text to print the full path name of the executable and the build date and then exit.

Figure 8-1: gds2text Flow Diagram



Procedure

Format: **gds2text** [**-license**[**s**]] [***options***] ***file*** [***structure***]

The gds2text tool interprets GDSII data according to the command-line input and data in the preference file, if any. When you specify ***structure***, gds2text prints data only from that GDSII structure to standard output. If no structure is named, gds2text prints the entire .gds file. The preference file can be used to set the default format and to define options that affect formatting.

After processing the license (if specified) but before processing any arguments, gds2text processes a default preference file, if found. The preference file processed is the first one found named +gds2textPrefs or .gds2textPrefs when searching the current directory, then the user's home directory, and finally the directory specified by the QUICKCAP_PATH environment variable, if defined.

After any default preference file is processed, gds2text processes any remaining command-line options in the following order.

1. The **-d**[**esc**] option describes the data being printed as flag, short, long, float, string, or byte.
2. The **-f**[**ile**] ***fName*** option specifies a preference file to be processed. You can specify multiple preference files using multiple **-f** options.
3. The **-l**[**ong**] option causes gds2text to print one record per line. If you do not specify this option, gds2text prints one primary record per line, along with any secondary records.
4. The **-p**[**os**] option causes gds2text to prefix each line with the file position.
5. The **-v**[**erbose**] option causes gds2text to print preferences.

Description of GDSII Data

The GDSII data, also called a stream, consists of a series of records. Each record begins with eight bytes that indicate the record length:

Bytes 1–4: The record length in bytes

Bytes 5–6: The record type

Bytes 7–8: The type of data

The gds2text tool processes each record in a GDSII file. Each standard record type is associated with a specific data type, although this is not a prerequisite for the data format. For each known record type (with the known data type), gds2text prints a corresponding name. For example, 0x0002 (bytes 5–8) is a header record. For information about record types, see "Recognized Record Types" on page 8-7.

Any data is printed according to the type of data.

0x00: Any data that is in the record is printed as an array of bytes.

0x01: Bit arrays (one byte each).

0x02: A two-byte signed integer (short integer) or an array of short integers.

0x03: A four-byte signed integer (long integer) or an array of long integers.

0x05: An eight-byte real value or an array of real values.

0x06: A string (text).

Other data types are printed as individual bytes.

The gds2text tool classifies the record types as primary or secondary. Each primary record type is associated with a record type that is the last record containing any related data. Typically, a primary record type is either standalone (it is its own last record), or it encompasses data up to the ENDDEL record. Primary and secondary records affect formatting. Normally, a primary record and all records up to the last one are printed on a line. If gds2text is run with the **-long** option, a primary record and all records up to the last one are printed together, with each record separated from the next record by a blank line.

The gds2text tool associates a print status with each record type: *blank*, *ignore*, or *print*. By default, all records are printed, except that ENDDEL is blank (generates an end-of-line, or if **-long** is specified, generates a line containing dashes).

Preference-File Declarations

The preference file, if defined, is the first one named +gds2textPrefs or .gds2textPrefs found when first searching the current directory; then the user home directory; and finally the directory specified by the QUICKCAP_PATH environment variable, if defined.

The preference file can be used to specify gds2text default behavior and create user-defined options. Any data in the preference file before the first ***-userOption*** declaration (see the following definition) sets the default behavior of gds2text. A semicolon begins a comment that extends to the end of the line.

-[***userOption***][**,**] [***description***]
Specifies that declarations on following lines (until the next definition) are invoked if you specify ***-userOption*** on the command line. If gds2text is run without arguments or without a GDSII file, any options and descriptions are listed, even when ***description*** is specified and ***userOption*** is not.

hex4[**:**] ***PrintStatus name*** [**secondary**]
hex4[**:**] ***PrintStatus name*** **until** ***recordName***
Specifies the name and format of a record that begins with four bytes specified by ***hex4***.

The ***hex4*** value must be of the format **0x*ttdd*** or **0X*ttdd***, where ***ttdd*** consists of four hexadecimal digits (0–9, A–Z, a–z). The first two digits, ***tt***, determine the record type. The next two digits, ***dd***, specify the type of data. (See “Description of GDSII Data” on page 8-2.)

PrintStatus is either **blank**, **ignore**, or **print**.

The ***name*** value is the name that gds2text uses to represent the record.

If **until** is specified, ***hex4*** is considered to be a primary record, ending with the specified ***recordName***, usually either ENDDEL (when secondary records are included) or the primary record itself ***name***.

If **until** is not specified, ***hex4*** is considered to be a secondary record.

blank *RecordOrType*[[**,**] ***RecordOrType*** [**…**]]
Specifies that the named records or types generate an end-of-line or (if **-long** is specified on the command line) a blank line. You can specify multiple records or record types on a line, optionally separated by commas.

RecordOrType can be the name of a record or a type of record. A record name can be followed by **secondary** (to make it a secondary record) or **until *RecordName*** (to make it a primary record). The following are recognized record types:

all records: Select all records.

blank records: Select blank records.

changed records: Select all records modified since the last **list** declaration.

ignored records: Select ignore records.

primary records: Select primary records.

printed records: Select print records.

unchanged records: Select all records not modified since the last **list** declaration.

cr_print *RecordOrType*[[**,**] ***RecordOrType*** [**…**]]
Prints a blank line followed by the named records or types. You can specify multiple records or record types on a line, optionally separated by commas.

ignore *RecordOrType*[[**,**] ***RecordOrType*** [**…**]]
Ignores the named records or types. You can specify multiple records or record types on a line, optionally separated by commas. See **blank** on page 8-4.

list *RecordOrType*[[**,**] ***RecordOrType*** [**…**]]
Prints the named records to SDTERR in a format equivalent to defining the command (see ***hex4*** on page 8-3). All records listed are marked as unchanged. ***RecordOrType*** is the same as that described previously, except that **secondary** and **until** are not recognized.

print *RecordOrType*[[,] ***RecordOrType*** [**...**]]
Prints the named records or types. You can specify multiple records or record types on a line, optionally separated by commas.

Sample Preference File

The following is an example of a preference file, +gds2textPrefs.

```
;COMMON STANDALONE STUFF
-basic, Print primary records
ignore all records
print primary records

-labels, Print labels (and structure names)
ignore primary records,MAG,ANGLE,TEXTTYPE,PRESENTATION,STRANS,XY
print TEXT,STRNAME

-refs, Print array and structure references, names
ignore primary records,MAG,ANGLE,TEXTTYPE,PRESENTATION,STRANS,XY
print AREF,SREF,STRNAME

-elements, Print physical elements (and structure names)
ignore primary records,MAG,ANGLE,TEXTTYPE,PRESENTATION,STRANS,XY
print BOX,PATH,BOUNDARY,STRNAME

;-just IS MEANT TO BE USED WITH OTHER STUFF
-just, Ignore primary records
ignore primary records

- To be used with -just...
-TEXT, Print labels
print TEXT

-REFS, Print references
print SREF,AREF

-ELEMENTS, Print elements
print BOUNDARY, PATH, BOX

-STR, Print structure names
print STRNAME
```

```
- Listing elements
-list, List all elements
list all records

-changed, List changed elements
list changed records
```

Sample Run Without Arguments

The gds2text tool, run without arguments and using the preference file +gds2textPrefs shown in “Sample Preference File” on page 8-5, generates the following output.

```
*****************************************************************
Copyright (C) 2003-2013 Synopsys
gds2text version 1.2.##-buildPlatform (compiled time)
*****************************************************************

This program generates a text description of a GDSII file.

FORMAT:
  gds2text [-license] [<options>] <gdsIIFile> [<gdsIIStr>]

OPTIONS:
  -d[esc]         Describe data
  -f[ile] <fName> Process preference file
  -l[ong]         Print one record per line
  -p[os]          Prefix each line with position in file
  -v[erbose]      Print preferences

OPTIONS DEFINED IN PREFERENCE FILE +gds2textPrefs:
  -basic          Print primary records
  -labels         Print labels (and structure names)
  -refs           Print array and structure references, names
  -elements       Print physical elements (and structure names)
  -just           Ignore primary records
 To be used with -just...
  -TEXT           Print labels
  -REFS           Print references
  -ELEMENTS       Print elements
  -STR            Print structure names
```

```
Listing elements
 -list           List all elements
 -changed        List changed elements
```

Recognized Record Types

The following list shows recognized record types. You can generate it using the preference file shown in “Sample Preference File” on page 8-5 and executing the **gds2text -list** command.

```
0x0002:  print HEADER until HEADER
0x0102:  print BGNLIB until BGNLIB
0x0206:  print LIBNAME until LIBNAME
0x0305:  print UNITS until UNITS
0x0400:  print ENDLIB until ENDLIB
0x0502:  print BGNSTR until BGNSTR
0x0606:  print STRNAME until STRNAME
0x0700:  print ENDSTR until ENDSTR
0x0800:  print BOUNDARY until ENDDEL
0x0900:  print PATH until ENDDEL
0x0A00:  print SREF until ENDDEL
0x0B00:  print AREF until ENDDEL
0x0C00:  print TEXT until ENDDEL
0x0D02:  print LAYER
0x0E02:  print DATATYPE
0x0F03:  print WIDTH
0x1003:  print XY
0x1100:  blank ENDDEL until ENDDEL
0x1206:  print SNAME
0x1302:  print COLROW
0x1400:  print TEXTNODE
0x1500:  print NODE
0x1602:  print TEXTTYPE
0x1701:  print PRESENTATION
0x0000:  print SPACING
0x1906:  print STRING
0x1A01:  print STRANS
0x1B05:  print MAG
0x1C05:  print ANGLE
0x0000:  print UINTEGER
0x0000:  print USTRING
0x1F06:  print REFLIBS
0x2006:  print FONTS
```

```
0x2102:  print PATHTYPE
0x2202:  print GENERATIONS until GENERATIONS
0x2306:  print ATTRTABLE
0x2406:  print STYPTABLE
0x2502:  print STRTYPE
0x2601:  print ELFFLAGS
0x2703:  print ELKEY
0x0000:  print LINKTYPE
0x0000:  print LINKKEYS
0x2A02:  print NODETYPE
0x2B02:  print PROPATTR
0x2C06:  print PROPVALUE
0x2D00:  print BOX
0x2E02:  print BOXTYPE
0x2F03:  print PLEX
0x3003:  print BGNEXTN
0x3103:  print ENDEXTN
0x3202:  print TAPENUM
0x3302:  print TAPECODE
0x3401:  print STRCLASS
0x3503:  print RESERVED
0x3602:  print FORMAT
0x3706:  print MASK
0x3800:  print ENDMASKS
0x3902:  print LIBDIRSIZE
0x3A06:  print SRFNAME
0x3B02:  print LIBSECUR
0x3C00:  print BORDER
0x3D00:  print SOFTFENCE
0x3E00:  print HARDFENCE
0x3F00:  print SOFTWIRE
0x4000:  print HARDWIRE
0x4100:  print PATHPORT
0x4200:  print NODEPORT
0x4300:  print UCONSTRAINT
0x4400:  print SPACER_ERR
0x4500:  print CONTACT
```

9. The qtfx Tool

Overview

Format

qtfx [**-license**[**s**]] [**-v**[**ersion**]] [***-inFmt***] ***inputFile*** [***outputOptions***] [***-outFmt***] [***outputFile***]

The qtfx tool (QTF translator) translates from one of several technology-file formats into another format. It is designed primarily for conversion into the QTF format, (See “QTF Language” on page 11-1.)

The **-license** option causes qtfx to print the license information. The **-version** option causes qtfx to print the full path name of the executable and the build date, and then exit.

Program Operation

When an input file is specified, qtfx writes translated data to ***outputFile***, or to *stdout*. Any comments and error messages are written to *stderr*. If the output file is named on the command line, any existing file is preserved by appending a tilde (**~**) to its file name. Translation from QTF format to QTF format replaces any **width** tables by **etch** tables, generates any **scaleR** tables, generates tables from any **qtfDeriveTable** formulas, simplifies tables, includes and default values, and reformats the data.

If run without any options, qtfx prints a description of its format and available options, as follows.

```
*****************************************************************
Copyright (C) 2003-2013 Synopsys
```

```
qtfx version version-buildPlatform (compiled time)
****************************************************************

This program translates technology-file formats
  (by default from 'tech' to 'qtf')

FORMAT:
  qtfx [-license[s]] [-<inFmt>] <inFile> [<outputOptions>] [-<outFmt>] [<outFile>]
When <outFile> is not specified, output is to stdout
Default formats are inferred from the suffix of the file names

RECOGNIZED INPUT FORMATS:
  tech
  qtf
  ircx
  ircxDepletedWell
  ircxNoMap
  ircxEM
  itf

RECOGNIZED OUTPUT OPTIONS:
  -adjustStack            Apply constant adjustDepth values
  -encrypt [weak|strong]  Encrypt data (default: weak]
  -ITFepsVsS [eps|epsXY]  Form of ITF space-dependent dielectrics (default: eps)
  -noFlow                 Do not calculate cases applicable to various flow levels
  -removeIgnoredData      Remove data specified in 'qtfIgnoreData' block
  -stdTypes               Categorize layers according to standard layer types
  -tableAliases [off|on|same]  Use table aliases (default: same existing aliases)

RECOGNIZED OUTPUT FORMATS:
  tech
  qtf
  em
  txt (test structures)
```

The **-license** option prints a description of the license information.

Run Status

In a standard run, qtfx translates the input file from the input format to an output format. The translated output is either written to a file named on the command line, or to *stdout*. Any comments and error messages are written to *stderr*. The qtfx tool exits with one of the following status codes:

- **0**: *No error*: qtfx successfully translated the input file.
- **1**: *License error*: qtfx was unable to establish that it was licensed.
- **2**: *Command-line error*: The command line contained an unrecognized option or the wrong number of arguments.

- **3**: *File-open error*: qtfx was either unable to open the specified input file for reading, backup the specified output file, or open the output file for writing.
- **4**: *Input error*: Input-data errors prevented qtfx from translating the input file.
- **5**: *Content error*: qtfx translated the input file but found some problems, marked by question marks (**?**) in the translated output.
- **6**: *Memory-allocation error*: qtfx ran out of memory. This error probably indicates a program error.
- Other values indicate a program error.

Recognized Technology Formats

The default input format is **tech** (gds2cap language, containing no QTF commands). The default output format is QTF. With no output file specified, qtfx outputs to *stdout*.

The qtfx tool recognizes the following values for ***-inputFmt*** and ***-outputFmt***.

-ircx *Input format*

Uses iRCX format, for input only. The iRCX format is owned by TSMC, Ltd. The qtfx tool reports to *stderr* about information it does not use, either because it does not recognize an IRCX keyword, or because it recognizes an iRCX keyword but does not need the associated information. If ***inputFmt*** is not specified and the input file name ends with *.ircx*[*NoMap*], qtfx considers the file to be iRCX formatted.

The output variants **-ircxDepletedWell** and **-ircxNoMap** affect the gds2cap-technology data generated in the QTF file. The **-ircxEM** output option translates only the electromigration data.

-ircxDepletedWell *iRCX output variants*
-ircxNoMap

Specifies an output variant when translating from iRCX to QTF (**-ircx**). By default, qtfx interprets the layer mapping section of the iRCX file and generates corresponding gds2cap technology-file data in a **qtfVerbatim tech** block. The resulting QTF file can be used directly as a gds2cap technology file. By default, the regions under the gate, source, and drain regions of a MOSFET are considered to be conductive.

For **-ircxDepletedWell**, qtfx still generates gds2cap technology-file data, but uses a depleted-well model in which the regions under the gate, source, and drain regions of a MOSFET are considered to be depleted silicon (few mobile charge carriers), essentially dielectric material with a dielectric constant of 12.9.

For **-ircxNoMap**, qtfx ignores the mapping information and generates no technology-file data.

Note: When translating from iRCX and generating gds2cap technology data, qtfx incorporates an internal library of device ID data and corresponding *Cdp* data. (See the description of the **CdpPerLength** layer property in the *gds2cap User Guide and Technical Reference*.) TSMC does not support such data in the iRCX format. When the internal library does not include this information for a given process node (and revision), qtfx generates a warning. The generated technology-file information can still be used, but it includes no Cdp-related information.

-ircxEM Output format

Interprets the electromigration data in the iRCX input file (**-ircx**) and generates associated tables in a list format. The stack data is used only to establish the range of the drawn width (**Wdr**) to use for converting iRCX polynomial expressions into tables. This output is not part of the QTF language and is intended for processing by electromigration-related tools.

-itf *Input format*

Uses ITF format, for input only. The ITF format is owned by Synopsys, Inc. The qtfx tool reports to *stderr* about information it does not use, either because it does not recognize an ITF keyword, or because it recognizes an ITF keyword but does not need the associated information. If ***inputFmt*** is not specified and the input file name ends with *.itf*, qtfx considers the input file to be ITF formatted.

-removeIgnoredData *Option after input file*

Removes any data specified in the input file (QTF) in an ignore-data section. (See “QTF Ignored Data” on page 11-35 for a description of ignored data). The qtfx tool also flags data in the ignore-data section with *(removed)*, and appends *(modified)* to the **technology** string, if defined.

-qtf (Default output format) *Input and output formats*

Uses QTF format, for input or output. An input QTF file can only include QTF declarations. See “QTF Language” on page 11-1.

-tech (Default input format) *Input and output formats*

Uses gds2cap format, for input or output. An input gds2cap technology file *cannot* include any QTF data. As an output format, qtfx generates basic stack-related declarations, but no information related to etch, without nominal heights, or stack variation (**adjustDepth**). If ***inputFmt*** or ***outputFmt*** is not specified and the input or output file name ends with *.tech*, qtfx considers the file to be a standard gds2cap technology file. See “POLY Contacts” on page 9-8.

-txt *Output format*

Generates a test layout consisting of the four regions, shown in Figure 9-1 on page 9-5. Each region contains a uniform density of wires of minimum or maximum width (*wMin* or *wMax*) and minimum or maximum spacing (*sMin* or *sMax*). These test patterns can be used to confirm that the QTF data is correct. Test patterns only include interconnects that do not overlap vertically with any other interconnects. If ***outputFmt*** is not specified and the output

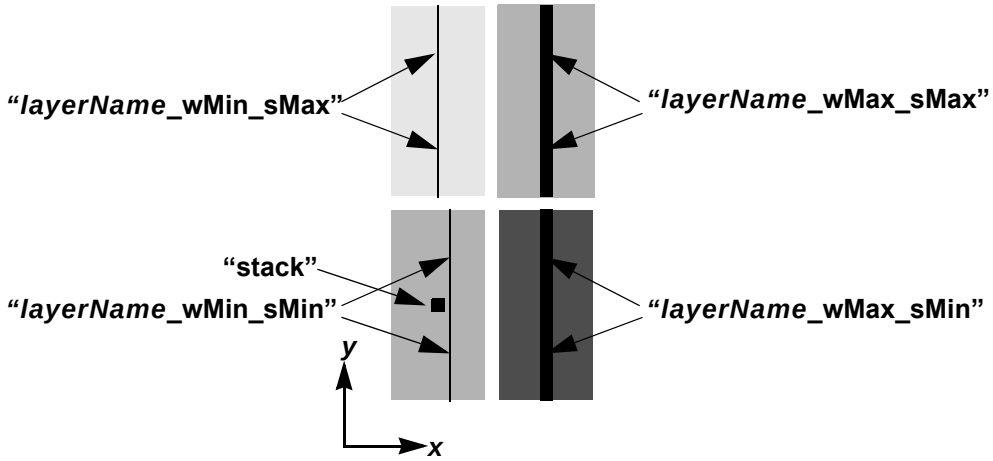
file name ends with *.txt*, qtfx generates the test patterns. To prevent gds2cap from enumerating the individual lines in the line arrays, include **global suffix="!"** in the technology file.

When no density window or grid is specified, or when the density window is 50 μm and the density grid is 25 μm, each region is 125 μm (x) by 225 μm (y). Otherwise, qtfx selects a region width and height so that the edges are at least one window width away from a 100 μm line (y) in the center.

Stack Structure: The lower-left region includes a stack with vias, named **stack**. This structure can be examined by running gds2cap, followed by QuickCap (**-x**).

Resistance Structures: In the lower-left region, the third line to the right of the stack on each layer can be used to check resistance (minW, minS). In the other regions, the line in the center of the stack can be used to check resistance. Each of these lines has two labels 100 μm apart. The labels are of the form ***layerName_w_s***, where ***w*** is either *wMin* or *wMax*, and ***s*** is either *sMin* or *sMax*. To check the resistance, ensure that the technology file includes the declaration **labelAsPin allLabels**, and run **gds2cap -rc 1 *techFile* test**. The resistance values in test.spice represent the interconnect resistance of 100 μm lines. The stack resistors should match the resistance of the via layers.

Figure 9-1: Test-pattern names in the four test regions. Each region is approximately uniform with lines of width *wMin* or *wMax*, and spacing *sMin* or *sMax*. The *wMin*/*sMin* region also includes a stack with vias.



Recognized Output Options

The following output options can be specified on the command line after the input-file name to customize aspects of the generated output.

-adjustStack
Applies any uniform **adjustDepth** value (**thkT**) to the stack. This changes all **z0** and **z1** values at or above **z1** of the uniform **adjustDepth** layer by a constant based on the **adjustDepth** layer values: [**thkT**-(**z1**-**z0**)]. Values below **z0** of the uniform **adjustDepth** layer are not affected. A value *z* between **z0** and **z1** of the uniform **adjustDepth** layer is changed by the value [(*z*-**z0**)/(**z1**-**z0**)]*[**thkT**-(**z1**-**z0**)]. Without **-adjustStack**, the qtfx tool considers a uniform **adjustDepth** value to be an error.

-encrypt [**weak**]
-encrypt strong
Generates encrypted QTF data. The qtfx tool does not encrypt header comments, **qtfIgnore** data (intended for user modification), or **qtfVerbatim tech** data (intended for gds2cap). When weak encryption is specified, qtfx encrypts generated QTF data, but other tools such as RGEN, QCP, gds2cap, or QuickCap do not hide any information they generate. Support for strong encryption depends on the tool used.

The qtfx tool can read an iRCX or ITF file that contains a **#hidden** command. In the absence of the **-encrypt** option, qtfx encrypts such a file as if the **-encrypt weak** option were specified.

-noFlow
Prevents qtfx from generating flow data in a QTF output file. The **-noFlow** option must be specified on the command line after the input-file name. Flow data, recently added to the QTF language, allows gds2cap and QuickCap to perform technology modeling for more efficient operation with little impact on accuracy. See Flow Data Models on page 11-43. When the input file is a QTF file, **-noFlow** passes any flow data from the input file to the output file.

-ITFepsVsS [**eps**|**epsXY**] (Default: **eps**)
When translating from ITF, the **-ITFepsVsS** option specifies whether to translate ITF ER_VS_SI_SPACING commands as physical dielectrics (**eps**, the default) or nonphysical dielectrics (**epsXY**).

-stdTypes
Assigns standard layer types to each interconnect and via layer based on their relationship to each other, guided by the layer names. The type appears in the conductor or via stack of the QTF file under the header **desc**. See **desc** on page 11-14 for a list of recognized standard types. Without **-stdTypes**, qtfx preserves any **desc** values that it finds in the input file.

-tableAliases off|on|same (Default: **same**)

Specifies implementation of the **qtfTable alias** command. The **-tableAliases** option must be specified on the command line between the input and output file names. For information on table aliases, see the **alias** command on page 11-48.

The arguments of the **-tableAliases** option are

- **same** (default): qtfx only uses the table aliases found in the input file.
- **off**: qtfx converts any aliased tables to **qtfTable** structures.
- **on**: qtfx represents all identical tables as a single **qtfTable** structure with table aliases.

High-Level Interpretation

The qtfx translator performs some high-level interpretation from ITF and iRCX formats, recognizing the following common naming conventions for layers related to standard integrated-circuit technologies.

Field Oxide

A dielectric layer name beginning with *FOX* or *STI* represents field oxide.

Gate Oxide

A dielectric layer name beginning with *GOX* represents gate oxide. For the purpose of generating the correct surface profile, qtfx treats gate oxide as a zero-thickness dielectric on top of FOX, conformal over OD (or RX) with the specified thickness. The qtfx translator changes the height of GPOLY (gate poly) if necessary, to keep it above the gate oxide.

Liner and Spacer

Dielectric layer names beginning with *SPACER* and *LINER* represent inside and outside conformal layers over POLY. The qtfx translator ensures that these are defined in the correct order relative to each other and relative to ILD. It also uses these thicknesses to decrease QuickCap range from its default value of **scale** when such a change is deemed to improve extraction speed.

ILD

A dielectric layer name beginning with *ILD* represents the dielectric above POLY and any liner or spacer.

Active Region

An interconnect layer name beginning with *OD* or *RX* represents the active layer. This includes MOS source drain and channel regions, and well contacts. The qtfx translator considers the gate oxide to be a conformal layer that only exists over the active region.

POLY

Interconnect layer names beginning with *POLY*, *GPOLY*, or *FPOLY* are considered to be POLY. The qtfx translator considers that any via contacts GPOLY.

POLY Contacts

Via layer names beginning with *POLYCONT*, *PPOLYCONT*, or *NPOLYCONT* are considered to be POLY contacts. The qtfx translator considers that any such layers contact FPOLY and GPOLY if both are defined.

10. SvS

Overview

Format 1

SvS [**-license**[**s**]] [**-v**[**ersion**]] [***options***] [***root***[**.spice**] [***refNetlist*** [***subCkt***]]]

Format 2:

SvS -redo ***fileName***

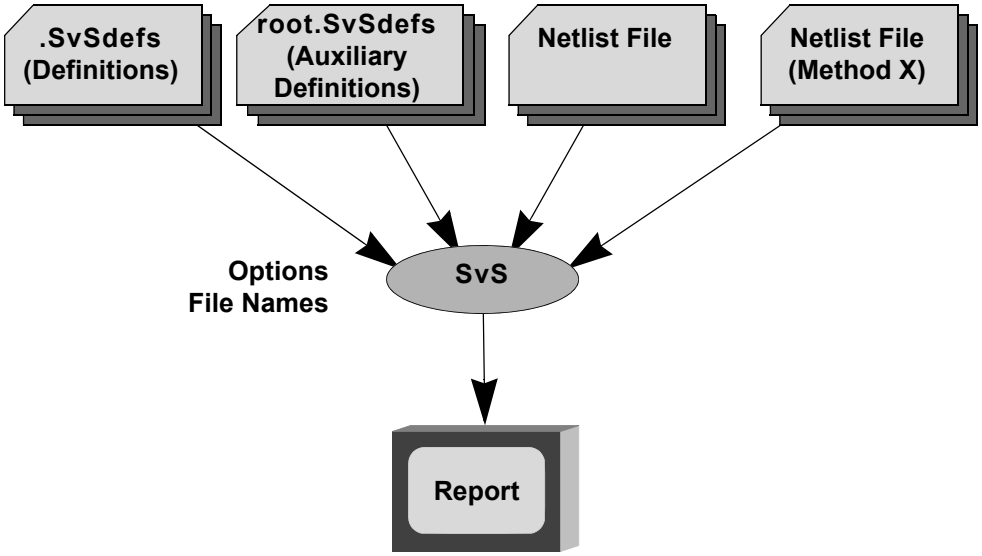
The SvS command compares the netlist file with a reference deck and can produce any of several output formats so that net names generated by gds2cap are consistent with those of the reference deck. (SvS stands for Schematic versus Schematic.) You can run SvS in a loop to process data generated by a series of gds2cap export runs.

The **-license** option causes SvS to print the license information. The **-version** option causes SvS to print the full path name of the executable and the build date, and then exit.

SvS provides a mechanism for establishing net names when any of the nets in the GDSII file are unlabeled. SvS is also useful for verifying that connectivity established by gds2cap is consistent with a reference netlist.

Figure 10-1 on page 10-2 shows the basic SvS flow. Based on definitions in +SvSdefs (or .SvSdefs) and, optionally, in an auxiliary definition file, root.SvSdefs, SvS compares a netlist such as one generated by gds2cap to a reference netlist. SvS generates a report on the screen, describing the correspondence between the two netlists. Some options cause SvS to generate or modify other types of files.

Figure 10-1: SvS Basic Flow Diagram



Procedure

Format: **SvS** [**-license**[**s**]] [***options***] [***root***[**.spice**] [***refNetlist*** [***subCkt***]]]

or

SvS -redo ***fileName***

The SvS command reads a definition file and the two netlists to be compared.

The definition file defines formats of netlisted devices used for netlist comparisons and other general features of the netlist. If not specified on the command line with the **-defs** option, the definition file is +SvSdefs or .SvSdefs.

An auxiliary definition file, named root.SvSdefs, is also read if it exists. This is generally maintained by SvS for a hierarchical netlist.

The reference netlist, if not specified, is named root.ref. The netlists are compared on the basis of topology. If they match, SvS can produce any of several result files taking into consideration the mapping of net and the device names between netlist files.

Program operation is controlled by command-line options (see “Options” on page 10-6). Netlist statements are interpreted according to definition-file declarations (see “Definition-File Commands” on page 10-8).

Use the **-redo** option to execute SvS using a command line found in the second line of ***file***, typically a file previously generated by SvS.

Definitions

SvS first inputs definitions from a definition file. The default name is +SvSdefs or, if such a file does not exist, .SvSdefs. You can specify a different definition-file name as an extra argument on the command line before the root name. Definitions are also input from an auxiliary definition file named root.SvSdefs, if such a file exists. Definition-file declarations describe the netlist statements, including device terminals that can be interchanged, accuracy required for parameters to be considered equivalent, how to recognize and merge parallel devices, how to recognize short circuits, and equivalent device-model names.

Netlist Input

The gds2cap and reference netlists become inputs to SvS.

During an SvS export run (see the **-export** option definition on page 10-6), SvS processes a single subcircuit from the gds2cap netlist for which no template is declared in a definition file.

During an SvS run that is *not* an exported run, only data outside of any subcircuit definition is processed or only data inside a subcircuit definition is processed, depending on whether a subcircuit name is specified as the last argument on the command line.

Netlist Correspondence

There are three types of netlist correspondence, as described in the following sections.

Topology-Based Correspondence

Correspondence between the two netlists is based primarily on topology. Devices are differentiated, initially, only by device type (such as resistor, capacitor, or MOSFET) and device name (such as MN or MP). Nets are distinguished by the device terminals they include. For example, if only one net in each netlist is connected to 15 source terminals of MOSFETs with model MN, the two are nets correlated.

The *correct* correspondence can be ambiguous. In an eight-bit latch, for example, all eight bits might be exactly identical from a topological viewpoint. This results in eight sets of nets in the same *automorphic class*. Without any additional information, SvS arbitrarily selects one of the possible correspondence mappings.

Name-Based Correspondence

Correspondence between nets (nodes) can also be established from net names, depending on the argument of the **-names** option (see page 10-6). By default, this occurs before any topology-based correspondence is established. A name comparison can be bypassed, however; or it can occur after an initial topological comparison, after devices are distinguished based on parameter values, or after automorphic reduction (selecting an arbitrary distinct correspondence within an automorphic class).

Parameter-Based Correspondence

Correspondence between devices can also be established based on device parameters such as L and W of a MOSFET, depending on the argument of the **-parms** option (see page 10-7). By default, no such correspondence is made. However, correspondence can be established from parameter values, according to parameter definitions in the definition file that include resolution properties.

Results

This section describes the results of an SvS run.

File Names and Result Files

When SvS processes a subcircuit (because the command line lists the subcircuit or includes the **-export** option), it derives a prefix, strRoot, described later in this section, based on the name of the associated GDSII structure (if SvS can find it) or based on the subcircuit name. To find the associated GDSII structure, SvS searches the gds2cap auxiliary technology file root.tech.aux (if the file exists) to find a **structure** declaration with an **alias** property matching the subcircuit name. SvS expects the first line of a **structure** declaration to be of the following form:

structure ***strName*** **[,] alias [=]** ***subcktName***

This format is consistent with the technology-file format of the **structure** declaration, but stricter than actually required by gds2cap. The gds2cap tool permits line breaks. Also, gds2cap allows other properties to precede any **alias** declaration. Any **structure** declaration generated by gds2cap during an **-export** run, however, is in the form expected by SvS.

The strRoot prefix consists of root (from the command line) suffixed by a dot (.) and either the GDSII structure (if it can establish the structure name) or the subcircuit name. When SvS does not process a subcircuit, strRoot is the same as root.

Summary Output

Results are summarized on the terminal (standard output). Additional information is included if you specify the **-verbose** option. You can specify the **-Devices** and **-Nodes** options to print the detailed correspondence between devices and between nodes (nets).

The **-Output** option generates an output file, strRoot.out, that includes detailed device and node information, independent of the **-Devices** and **-Nodes** options.

Result Files

You can generate result files for a variety of purposes, depending on the options you use.

Except for the output file ***strRoot.out*** (generated when you use **-Output**), result files are normally generated only if 100% correspondence has been established and the names of some corresponding nets are different. The **-force** option forces generation of result files, even if full correspondence has not been established or all net names match.

-labels
Translates a strRoot.pos file to a strRoot.labels file with net and device names matching those in the reference file. This file can be used by gds2cap to generate data with net names matching reference-file names. The file names and formats allow a subsequent gds2cap export run to automatically read the labels file associated with the structure being exported.

-quickcap
Creates a rename file (strRoot.rename), translates a bounds file (strRoot.bounds) if it exists, and creates or updates a header file (strRoot.cap.hdr). The original bounds file is saved in a strRoot.bounds file. Any original header file is saved in strRoot.cap.hdr~.

-spice
Updates the netlist file, root.spice, so that net and device names match those in the reference netlist. The original file is saved in a root.spice~ file. Device names cannot be mapped for parallel devices and short circuits (processed as such, according to declarations in the definition file).

Exporting Hierarchy

Like scripted gds2cap export runs, SvS export runs can be scripted. The following executable file, loop, or a variant, can be used for repeatedly executing a command line until it returns without a zero status.

```
#/bin/csh
while (1)
$*
if($status) break
end
```

SvS terminates with an exit status of 0 unless it fails to find 100% correspondence or if (on an export run) you define templates for all subcircuits in the gds2cap netlist.

Using the loop file defined previously, a command such as the following one exports subcircuit templates from the gds2cap netlist, lib.spice, until there are no structures left to export or until less than 100% correlation is found.

```
>loop SvS -export -quickcap -spice lib lib.ref
```

The reference netlist, here, is lib.ref. Each SvS run exports a new template to lib.SvSdefs. If any pair of corresponding nets have different net names, QuickCap and netlist files are generated and updated. You can specify the **-force** option to generate or update these files even if all names match.

Options

The following options are available for the SvS command:

-body
Compares only the body of the subcircuit, ignoring the .SUBCKT line. This is useful for establishing net names when GDSII data does not include labels for the subcircuit pins. The **-body** option requires that the subcircuit be named on the command line.

-d[**efs**] ***defsFile***
Specifies a definition file other than the default one (.SvSdefs or +SvSdefs).

-e[**xport**]
Generates and updates the root.SvSdefs file. Compares one subcircuit in a hierarchical netlist. The subcircuit is one found in the gds2cap netlist and *not* already defined as a subcircuit template in any definition file. On completion, exports a template corresponding to the subcircuit to the auxiliary definition file, root.SvS.defs. The original auxiliary file, if any, is saved in root.SvS.defs~.

-f[**orce**]
Implements generation of output files, even with less than 100% correlation or when no names have changed.

-names 0...4 (Default: **1**)
Selects the manner in which net names influence the correspondence of nets. If a net in one netlist has the same name as a net in the other file but SvS has already established that the nodes are not equivalent (for example, they are connected to a different number of NMOS source/diffusion terminals), SvS does not force the nodes to be equivalent based on names.

0 – Do not base netlist correspondence on names. Net names are never considered. This allows, for example, VDD to be correlated with the reference name VDD1, while correlating VDD1 with the reference name VDD2.

1 (default) – Start by correlating nets with same names.

2 – Correlate net names after parameter-based correlation of devices.

3 – Correlate net names after topological equivalence is established.

4 – Correlate net names after automorphic reduction (resolving ambiguous correlation).

-labels

Translates strRoot.pos to strRoot.labels, with net and device names matching those in the reference file. This can be used by a subsequent gds2cap run to generate data with net names matching reference-file names. The strRoot prefix includes the structure name for an export run, but is root, otherwise.

-parms 0..2 (Default: **2**)

Selects the level of comparison between devices with the same model name.

0 – Parameter values are not considered when checking whether devices are the same. Any discrepancies are noted in the output, however.

1 – Devices are not considered equivalent if the parameters are too different (compared with the tolerance). The comparison is not as strict as **-parms 2**.

2 (default) – Devices are not considered equivalent if the parameters are too different (compared with the tolerance). The comparison is stricter than **-parms 1**.

-quickcap

Creates a rename file (strRoot.rename), translates a bounds file (strRoot.bounds) if it exists, and creates or updates a header file (strRoot.cap.hdr). Use the **-netlist** option with **-quickcap** if QuickCap is to update the netlist. The strRoot prefix includes the structure name for an export run, but is root, otherwise.

The rename file contains QuickCap **rename** declarations, which specify an old net name and a new net name. When QuickCap reads a **rename** declaration, it renames a net having the old name with the new name.

If a bounds file strRoot.bounds exists, QuickCap **bounds** declarations in the bounds file are updated to reference names in the reference netlist. The original bounds file is saved in strRoot.bounds~.

If the header file already exists, the original is saved in strRoot.cap.hdr~. SvS generates a new header and removes old ***#include*** declarations and inserts new ones. If a bounds file exists, it is included *after* including the rename file. SvS also appends QuickCap **#include** declarations to include the QuickCap deck (strRoot.cap) and the rename file. The rename file is included before the bounds file to ensure that QuickCap **extract** declarations in the header

file reference new net names. This allows QuickCap to select the correct bounds of extracted nets. The rename file is also included after the QuickCap deck because of **rename** declarations that might be in the QuickCap deck itself.

-spice
Updates the netlist file, root.spice, so that net and device names match those in the reference netlist. The original netlist is saved in root.spice~.

-v[**erbose**]
Generates additional output to the terminal.

-D[**evices**]
Lists device correspondence between the gds2cap and reference netlists.

-N[**odes**]
Lists node correspondence between the gds2cap and reference netlists.

-O[**utput**]
Generates complete results (including device and node correspondence) in strRoot.out.

-
Specifies that there are no more options on the command line. This allows the next argument to begin with "-".

Definition-File Commands

Commands in the definition file include customizations (declarations beginning with **#**), subcircuit templates (which begin with the previously defined instance character or the previously defined subcircuit keyword), and device formats (which begin with an alphabetic character a–z or A–Z other than the instance character). Other lines are ignored.

#addNodeAlias ***nodeName nodeName(s)***
Establishes equivalent node names. This declaration is not robust. Use it only if each set of aliased node names in the source (.spice) file maps to the same set in the reference file.

#**caseFolding** [***element***] **on** (Default for devices, models, parameters, subcircuits)
#caseFolding [***element***] **off** (Default for nodes)
Specifies whether names are case dependent. The default is consistent with defaults for gds2cap and QuickCap. If not specified, ***element*** defaults to **nodes**. Although gds2cap creates a consistent **casefolding** declaration for QuickCap, it is not passed to SvS. Therefore, if the gds2cap technology file includes **caseFolding on**, the SvS definition file must include **#caseFolding on**.

The ***element*** value can be one of the following types:

devices – A device ID is the set of characters immediately following the first character on a line. Note that the first character is never case-sensitive.

models – A model is the name (optional) following the list of nodes of a device.

nodes – By default, node names are case-sensitive.

parameters – Parameters of a device follow the model (if specified) or the node list. Parameters of an instance follow the name of the subcircuit.

subcircuits – A subcircuit name is the first name in a subcircuit declaration or the name following the list of nodes of an instance.

#instance ***instanceCharacter*** (Default: **X**)
Specifies the netlist character (not case-sensitive) used to reference a subcircuit. An ***instanceCharacter*** is the first character on a line in the netlist that instantiates a subcircuit. Device formats in a definition file must not begin with ***instanceCharacter***. The default is **X**.

#modelAlias ***modelName modelName(s)***
Specifies equivalent model names. Model names can be separated by commas (,).

#multiplierParameter ***name***
Specifies the name of the device parameter that is used to represent a number of equivalent devices in parallel. For devices in parallel, parameter values are modified according to the parallel behavior defined with merge options (see ***mergeOption*** on page 10-12).

#subcircuit ***subcktKeyword***[,] ***endKeyword*** (Default: **.SUBCKT**, **.ENDS**)
Specifies netlist keywords delimiting a subcircuit. The format of a subcircuit declaration is assumed to be as follows: ***subcktKeyword subcktName pinList***. The default is not established until *after* the auxiliary technology file, if any, is read. To define subcircuits in the auxiliary technology file with the keyword **.SUBCKT**, it must first be defined in a **#subcircuit** declaration. The default is **.SUBCKT**, **.ENDS**.

elementCharacter [***pinList***] [***modelName***] [***parameterList***]
Defines the format of a netlist element such as a capacitor, resistor, or MOSFET. The element definition begins with a case-insensitive *element character*, that is, any character A–Z *except* the instance character, as defined by the **#instance** declaration. Only one definition is permitted per element character.

***pinList*:** Lists pin names, which are used to establish the number and equivalence of pins. You must give pins that are interchangeable, such as the terminals of a resistor or capacitor, the same name. Note that SvS does not handle devices such as mutual inductors or dependent sources when they reference other devices.

***modelName*:** Lists model names in the netlist. If specified, the model name appears between the pin list and the parameter list. You can use an asterisk or plus sign (* or +) in place of a model name.

***parameterList*:** Lists optional parameters for the element or instance. A parameter consists of a parameter name (optionally with a default value), an accuracy specification, a parallel-implementation formula, and a short-circuit condition.
To include a parameter list, make sure that either the first parameter name is followed immediately by an equal sign (**=**) or an opening parenthesis (**(**), or is preceded by an asterisk or plus sign (***** or **+**) or by a model name. Parameters are described in “Parameter Specification in the Definition File” on page 10-11.

instanceCharacter pinList subckt [*****] [***parameterList***]Instance definition
subcktKeyword subckt [***pinList***] [*****] [***parameterList***]

Defines the format of a netlist instance, such as a reference to a NAND gate. The instance declaration begins with the *instance character* or with the *subcircuit keyword*, which must be previously defined by the **#instance** or **#subcircuit** declaration, respectively. The two forms of declaration differ only by whether the subcircuit name is defined before or after the pin list. Any number of instance definitions are permitted if they have different ***subckt*** names.

***pinList*:** Lists of pin names used to establish the number and equivalence of pins. Pins that are interchangeable, such as the source and drain of a MOSFET, should be given the same name.

***parameterList*:** Lists optional parameters for the element or instance. A parameter consists of a parameter name (optionally with a default value), an accuracy specification, a parallel-implementation formula, and a short-circuit condition.
To include a parameter list, make sure that either the first parameter name is followed immediately by an equal sign (**=**) or an opening parenthesis (**(**), or is preceded by an asterisk or plus sign (***** or **+**) or by a model name. Parameters are described in “Parameter Specification in the Definition File” on page 10-11.

Parameter Specification in the Definition File

Netlist elements and instances can include parameters that can be:

- Used to distinguish elements or instances
- Modified to account for parallel devices
- Used to identify short-circuit groups (net names that are equivalent because they are shorted together)

A parameter is specified for an element or instance in a parameter list at the end of the definition of the element or instance:

elementCharacter [***pinList***] [***modelName***] [***parameterList***]
instanceCharacter pinList subckt [*] [***parameterList***]
subcktKeyword subckt [***pinList***] [*] [***parameterList***]

The parameter list can contain one or more parameters, optionally separated by commas. Each parameter can include a default value and a tolerance with up to two options:

parmName[=***default***][(***tolerance***[,***option***[,***option***]])]

The value ***default*** is used for elements and instances in the netlist that do not define the parameter.

Up to two options can be specified to define how to merge parallel devices (***mergeOption***) and how to recognize a short circuit (***shortOption***). A ***tolerance*** is required to specify any options.

tolerance

The tolerance is used to decide whether devices in the two netlists can be distinguished based on parameter values. The **-parms** argument, described on page 10-7, determines whether parameter differences are used to differentiate devices when establishing the correlation between netlists, or whether parameter differences are just noted in the results.

The tolerance can be relative (ends with %) or absolute (optionally ends with a character specifying units). A character specifying units is one of the following (case-insensitive): **k** (10^3), **m** (10^{-3}), **u** (10^{-6}), **p** (10^{-9}), **u** (10^{-12}), **f** (10^{-15}), or **a**(10^{-18}). Subsequent characters, if present, are simply ignored.

Parameters in the netlist file can be of the form ***parmName=value*** or simply ***value*** (in which case the value is attributed to the first parameter described in the definition file).

Devices are considered exactly equivalent only if the parameter values are within the specified tolerance. (Depending on the **-parms** argument, a topological match can be achieved with devices with unmatched parameters, in which case, discrepancies are noted.) If no tolerance is specified, the parameter value does not influence equivalence.

mergeOption

SvS recognizes and merges any parallel combinations of a device or instance in a netlist only if a merge option is specified for at least one parameter in the device or instance definition. Possible merge options are **add** (or **+**), **parallel** (or **||**), and **preserve** (or **=**). Two elements are in parallel when both are connected to the same nets (taking into account permutations allowed due to interchangeable pins), both have the same model or subcircuit name, and both have the same value for any **preserve** parameters. SvS also applies the merge operation for devices with a multiplier parameter (see **#multiplierParameter** on page 10-9).

add (or **+**): For devices in parallel, adds the parameters. Parameter values *a* and *b* result in a merged value *a*+*b*. This is suitable for capacitance and for the width parameter of MOSFETs.

parallel (or **||**): For devices in parallel, combines the parameters in parallel. Parameter values *a* and *b* result in a merged value *a***b*/(*a*+*b)*. This is suitable for resistance and inductance.

preserve (or **=**): Functions as the default merge option when any parameter in a parameter list includes a merge option. Devices are considered in parallel only if they have the same parameter value (within the specified tolerance). This is suitable for voltage sources and for the length parameter of MOSFETs. (If no parameter in a device or instance declaration includes a merge option, SvS does not reduce a parallel configuration of those devices.)

shortOption

SvS recognizes as a short circuit any device with a parameter value of 0 for all parameters including the option **short** (or **x**), if at least one parameter includes the **short** option. This is suitable for voltage sources, resistors, and inductors. SvS short-circuits all terminals together, so this is not suitable for the width parameter of a MOSFET. A short circuit essentially becomes part of a virtual device containing all nodes short-circuited together. This allows SvS to recognize equivalence between two netlists, for example, even when one netlist shorts A to B and B to C, whereas the other shorts A to B and A to C.

Limitations

SvS, although useful for establishing net names, has the following limitations.

- When netlists have large homomorphic sets (sets of devices and nodes in one netlist that can be arbitrarily mapped to devices and nodes in another), runtime can become large. This can occur, for example, in memory arrays where word or bitlines are not named and, therefore, cannot be uniquely matched.
- When netlists do not match, topologically, SvS might not be useful for establishing why they do not match. You generally need to compare the netlists manually to find out why they do not match. This can sometimes be accomplished by counting the number of occurrences of various types of elements and of various net names such as Vdd or Vss.

Sample Definition Files

The format of a definition file is described in “Definition-File Commands” on page 10-8. The definition files described in the following sections demonstrate basic attributes and more advanced attributes.

Basic Definition File

The following basic definition file defines resistors, inductors, capacitors, and MOS transistors.

Example 10-1:Basic Definition File

```
#caseFolding on
R term term * R(10%)
L term term * L(10%)
C
M SD G SD B * W(0.005um) L(0.005um)
```

The **#casefolding** declaration applies to net names.

Resistor and inductor terminals are interchangeable (both are named term). A pair of resistance or inductance values from the two netlists needs to agree within ±10% to be considered equivalent.

Capacitors are defined but are not used to establish topological agreement. This allows different parasitic capacitors to be defined in the two files. Without this declaration, SvS operates the same way, but generates a warning the first time it encounters a capacitor.

In MOS transistors, the source and drain terminals are interchangeable (both are named SD), and gate widths and lengths must match within ±0.005µm.

Advanced Definition File

The following definition file includes more advanced features: information for hierarchical analysis, parallel devices, short circuits, and a multiplier parameter.

Example 10-2:Advanced Definition File

```
#casefolding nodes=off models=on
#instance X
#multiplierParameter M
R term term * R(10%,||,x)
L term term * L(10%,||,x)
C
V term term * V(10%,=,x)
M SD G SD B * W(0.005um,+) L(0.005um)
#modelAlias NM NM_A NM_B
#modelAlias PM PM_A PM_B
```

The **#casefolding** declaration allows case folding for model names, but not for nodes.

Declaration of an instance character allows definitions of subcircuit instances. An auxiliary definition file, strRoot.SvSdefs, which can be generated by an SvS export run (see **-export**, page 10-6), for example, might contain the following:

```
X Z A B AND2
X Z A B C AND3
```

Although SvS does not generate templates with interchangeable pins, the file can be edited. For example, if the inputs to AND2 are equivalent, as well as the inputs to AND3, the following would be more appropriate:

```
X Z A A AND2
X Z A A A AND3
```

For resistors (R) and inductors(L): because **||** (equivalent to **parallel**) is specified, SvS merges parallel resistors and inductors, combining the resistor or inductor value in parallel. Because **x** (equivalent to **short**) is specified, any resistor or inductor with a value of 0 is considered a short circuit.

For voltage sources (V): because **=** (equivalent to **preserve**) is specified, parallel voltage sources with the same value are merged. Also, because of **x**, a voltage source of value 0 is considered a short circuit.

For MOSFETS (M): because **+** (equivalent to **add**) is specified, parallel MOSFETs with the same length are merged, adding the widths.

#modelAlias specifies that model names NM, NM_A, and NM_B are equivalent. Similarly PM, PM_A, and PM_B are equivalent.

SvS Examples

The following examples demonstrate various ways to ensure that the netlist updated by QuickCap is consistent with a reference netlist. These examples use a gds2cap technology file, **tech**; a GDSII file, layout; and a reference netlist, layout.ref.

Example 10-3:Updating the gds2cap Netlist

In the following example, SvS updates the gds2cap netlist and generates QuickCap header and rename files.

```
>gds2cap -spice tech layout
>SvS -spice -quickcap layout layout.ref
>quickcap options layout.cap.hdr
```

The SvS-generated QuickCap rename file contains QuickCap **rename** declarations. The SvS-generated QuickCap header file has QuickCap **#include** declarations to include the primary QuickCap deck and the rename file. QuickCap strips off the .hdr suffix automatically, so that the root name QuickCap uses is the same as if it operated directly on the primary QuickCap deck. If QuickCap options include **-spice,** QuickCap inserts capacitance results into the netlist.

Example 10-4:Converting a position File to a labels File

In the following example, SvS converts a position file to a labels file to be used by a gds2cap **-rc** run. The same flow could be used for a gds2cap **-rlc** run.

```
>gds2cap -spice -pos tech layout
>SvS -labels layout layout.ref
>gds2cap -rc tech layout
>quickcap options layout.cap
```

Because of the **-pos** option, the initial gds2cap **-spice** run generates a position file. SvS converts the position file to a labels file (**-labels**). (SvS is not able to establish correspondence between a **gds2cap -rc** netlist and the reference netlist because of the RC networks.) After the SvS run, the **gds2cap -rc** run automatically uses the default labels file to label nets. Net names in the resulting QuickCap deck and in the RC netlist are consistent with the names in the reference netlist. As in Example 10-3, if QuickCap options include **-spice**, QuickCap inserts capacitance results into the netlist.

Note: If the technology file includes an **importNoLabels** declaration, the **gds2cap -rc** command in Example 10-4 should be replaced by the following:

```
>gds2cap -rc -labels layout.labels tech layout
```

Example 10-5:Using Export Runs

The nonhierarchical netlist case in Example 10-3 can be adapted for export runs. In the following example, the reference netlist is also used by gds2cap to identify levels of the hierarchy and associated subcircuits and pins. The file loop is listed in “Exporting Hierarchy” on page 10-5. For each subcircuit, SvS updates the gds2cap netlist and generates a QuickCap header and rename files.

```
>loop gds2cap -spice -export layout.ref tech layout
>loop SvS -export -spice -quickcap -force layout layout.ref
>foreach FILE (layout.*.cap.hdr)
? quickcap options $FILE
? end
```

Running loop with gds2cap generates a hierarchical netlist containing subcircuits for GDSII structures corresponding to subcircuit declarations in the reference netlist. Running loop with SvS compares each subcircuit. SvS keeps track of which subcircuits have already been compared by maintaining an auxiliary netlist, tech.SvSdefs, which contains templates for subcircuits already exported. SvS generates or updates QuickCap-related files, even though all names are already correct (using the **-force** option). This ensures that each principal QuickCap deck has an associated header file. The **foreach** shell command runs QuickCap on each header file. If QuickCap options include **-spice**, each QuickCap run inserts capacitance results into the appropriate netlist subcircuit.

Example 10-6:Hierarchical RC Analysis with Full Net-name Correlation

Hierarchical RC analysis with full net-name correlation requires gds2cap and SvS runs to generate labels files consistent with the reference netlist.

```
>loop gds2cap -spice -pos -export layout.ref tech layout
>loop SvS -export -labels layout layout.ref
>rm layout.tech.aux
>loop gds2cap -rc -export layout.ref tech layout
>foreach FILE (layout.*.cap)
? quickcap options $FILE
? end
```

As in the nonhierarchical RC analysis case (Example 10-4), SvS needs to use without a RC netlist (gds2cap with **-spice** rather than **-rc**). The auxiliary technology file, layout.tech.aux, needs to be removed before the **gds2cap -rc** run so that the structures can be reexported. Note that the RC gds2cap run does not include a **-labels** option. On a gds2cap run without **-labels**, a file named

strRoot.labels is automatically processed as a labels file unless the technology file includes the declaration **importNoLabels**. Example 10-5 explains the operation of the loop script and the **foreach** shell command.

11. QTF Language

Except for a few layer-independent properties such as **densityWindow**, the QTF file uses a tabulated format to define layer properties. These consist of conductor stacks to define lateral conducting layers and vias, dielectric stacks to define planar and conformal dielectrics, and adjust-depth stacks to specify how stack heights vary from their nominal value across the layout. Some layer properties are names of other layers. Others are numeric, either a number or the name of a table of data, defined elsewhere in the file.

The QTF language can be generated by qtfx or manually. It can be read by qtfx (for translation) or by gds2cap (to integrate the physical properties with the functional layers). The QTF language defines the *physical* layers (such as METAL1) rather than the *functional* layers (such as MET1, RES1, and FLOAT1). The gds2cap tool integrates the description of the physical layers into its description of the functional layers.

General Format

Except for derived tables (see “Derived Tables” on page 11-50), QTF data is in blocks, each beginning with a line of the form **qtf*...*** [***name***] and ending with a line of the form **qtfEnd*...***. The QTF language includes the following blocks:

qtfEncrypted [**by vendor(s)**]/**qtfEndEncrypted**
Defines QTF encrypted data. See “QTF Bulk Encryption” on page 11-4.

qtfParameters/**qtfEndParameters**
Defines QTF parameters. See “QTF Parameters” on page 11-6.

qtfVerbatim tech|rgen|QuickCap/**qtfEndVerbatim**
Defines verbatim statements that other programs can access. See “Verbatim” on page 11-11.

qtfConductorStack/**qtfEndConductorStack**
Defines the conductor stack (interconnects, vias, and device-level conductors). See “Conductor Layers” on page 11-22.

qtfResistanceCorners/**qtfEndResistanceCorners**
Defines resistance-corner properties. See “Resistance Corners” on page 11-26.

qtfDielectricStack/**qtfEndDielectricStack**
Defines planar and conformal dielectrics. See “Dielectric Layers” on page 11-26.

qtfAdjustDepth/**qtfEndAdjustDepth**
Defines stack variation. See “Stack Variation” on page 11-29.

qtfCdpData/**qtfEndCdpData**
Specifies the amount of device capacitance that is in the parasitic domain. See “Cdp Data” on page 11-30.

qtfStubData/**qtfEndStubData**
Specifies the *stub data* used to generate stubs of material for capacitance extraction, such as raised parts of a diffusion region. See “Stub and Sublayer Data” on page 11-32. You need to specify the stub data rather than the sublayer data (**qtfSublayerData**) if the stub material must not be subjected to **ignoreCoupling** or **groundCoupling** commands that affect the parent layer.

qtfSublayerData/**qtfEndSublayerData**
Specifies the *sublayer data* used to generate sublayers of material for capacitance extraction, such as raised parts of a diffusion region. See “Stub and Sublayer Data” on page 11-32. You need to specify the sublayer data rather than the stub data (**qtfStubData**) if the sublayer material must be subjected to **ignoreCoupling** or **groundCoupling** commands that affect the parent layer.

qtfFlowData/qtfEndFlowData
Specifies the estimated error introduced by implementing certain approximations. See "Flow Data" on page 11-33.

qtfIgnoreData/qtfEndIgnoreData
Describes data to be ignored, avoiding the need to edit the QTF data itself. See "QTF Ignored Data" on page 11-35.

Similar to gds2cap, comments begin with a semicolon (**;**), and can be an entire line or on a line after data.

Data associated with a property (for example, with the **rshDr** property of METAL1) can be a *value* (number), a *table reference* (string, described in "Table References" on page 11-42), a *layer reference* (string), or *nothing* (indicated by two or more dashes).

The QTF language does not recognize any unit suffix. All lengths, heights, and widths are in microns. Resistivity is in ohm-microns. The only exception is the weight associated with a **densityWindow** parameter. The weight can be suffixed by **%**, in which case the value is interpreted as a percentage rather than a fraction.

Vendor-Based Encryption

The following environment variable supports character-based decryption, which might be available from some vendors. Encryption of QTF files is described in the next two sections.

setenv QUICKCAP_VENDOR_CHAR_CRYPT *DLL(s)*
Specifies full path names of DLLs for including character-based decryption. Multiple DLLs can be separated by commas or spaces. The DLLs for character-based decryption are described in the vendor-supported encryption and decryption section in the *gds2cap User Guide and Technical Reference*.

The qtfx tool encrypts data for the following cases:

- Generating an encrypted QTF file (specifying with the **-encrypt** command-line option, or translating a QTF file containing an **encrypt** QTF parameter)
- Translating **#qtfHide**…**#qtfEndHide** blocks.

The qtfx tool decrypts encrypted data when translating a QTF file containing encrypted data associated with the **qtfEncrypted** or **#qtfHidden** command. Decrypted data is reencrypted when it is translated.

QTF Bulk Encryption

The bulk of the QTF data can be encrypted in response to the **-encrypt** command-line option, or when translating a QTF file containing the **encrypt** parameter. Encrypted QTF data is represented between **qtfEncrypted** and **qtfEndEncrypted** commands. The qtfx tool generates these keywords and the encrypted data in response to the **-encrypt** flag, or if translating a QTF file containing the **encrypt** parameter. When any QTF data is encrypted, the only QTF data permitted outside of the **qtfEncrypted**/**qtfEndEncrypted** section are comments, **qtfIgnoreData** sections, and **qtfVerbatim tech** data.

This approach only encrypts the file, and does not prevent the user from referencing layer names in a technology file outside the scope of QTF data. The **qtfVerbatim tech** section, which is also covered by bulk encryption, can contain deprecated **hideDielectrics**, **hideLayer**, **hideParm**, **hideXY,** and **hideZ** commands.

A better approach is to hide specific data, as described in "QTF Data-Specific Encryption."

qtfEncrypted [**by vendor(s)**] Encryption command

An encrypted QTF file includes an optional list of one or more vendors. The qtfx tool implements character-based decryption based on associated DLLs defined by the **QUICKCAP_VENDOR_CHAR_CRYPT** environment variable. Each vendor must match a vendor key supported by one of the DLLs. The DLLs for character-based decryption are described in the vendor-supported encryption and decryption section in the *gds2cap User Guide and Technical Reference*.

Because qtfx generates the **qtfEncrypted** command in response to the **-encrypt** command-line option or in response to the **encrypt** QTF parameter, the only way to implement vendor-supported encryption is to include one or more **#qtfHide**…**#qtfEndHide** blocks that list the vendors. It is not necessary that these blocks include any data to be hidden. See **#qtfHide** on page 11-5.

QTF Data-Specific Encryption

Specific tables and layers can be hidden by adding **#qtfHide** and **#qtfEndHide** to a QTF file, and then using qtfx to convert the data to a new QTF file.

#qtfHidden [**by vendor(s)**]

Specifies the beginning of a block of hidden data. The qtfx tool generates **#qtfHidden** blocks when it translates a QTF file containing **#qtfHide**…**#qtfEndHide** blocks.

When the **#qtfHidden** command includes a list of one or more vendors, the qtfx tool then implements character-based decryption based on associated DLLs defined by the **QUICKCAP_VENDOR_CHAR_CRYPT** environment variable. Each vendor must match a

vendor key supported by one of the DLLs. The DLLs for character-based decryption are described in the vendor-supported encryption and decryption section in the *gds2cap User Guide and Technical Reference*.

#qtfHide [**by vendor(s)**]
#qtfEndHide

Specifies a block of QTF data to be hidden. In an interconnect stack, dielectric stack, adjust-depth block, Cdp block, or flow block, any lines that are in a **#qtfHide**…**#qtfEndHide** block are hidden. Any tables in a **#qtfHide**…**#qtfEndHide** block are also hidden. Keywords **#qtfHide** and **#qtfEndHide** can be defined anywhere except in an ignore-data block, parameter block verbatim block, or inside a table.

The qtfx tool hides only the lines of data within a data block, even when the entire data block is inside a **#qtfHide**…**#qtfEndHide** block. In a conductor block, for example, qtfx never hides the following lines: **qtfConductorStack**; the subsequent *header* line that defines columns **name**, **z0**, and so on; and, **qtfEndConductorStack**.

Several objects become hidden if they involve hidden data, even when they are not inside a **#qtfHide**…**#qtfEndHide** block. Tables associated *only* with hidden layers become hidden. Conformal dielectrics and **adjustDepth** layers with a hidden base layer (**layer**) become hidden. Stubs and sublayers with a hidden attach layer (**attach**) become hidden. Cdp data that depends on any hidden layer becomes hidden. Generated flow data that involves hidden layers becomes hidden.

Existing flow data that is merely translated from another QTF file is not affected. To ensure that flow data related to hidden layers becomes hidden, remove the flow data and use qtfx to retranslate the QTF. The qtfx tool generates new flow data, hidden as necessary.

When the **#qtfHide** command includes a list of one or more vendors, qtfx implements character-based encryption based on associated DLLs defined by the **QUICKCAP_VENDOR_CHAR_CRYPT** environment variable. Each vendor must match a vendor key supported by one of the DLLs. The DLLs for character-based decryption are described in the vendor-supported encryption and decryption section in the *gds2cap User Guide and Technical Reference*.

QTF Parameters

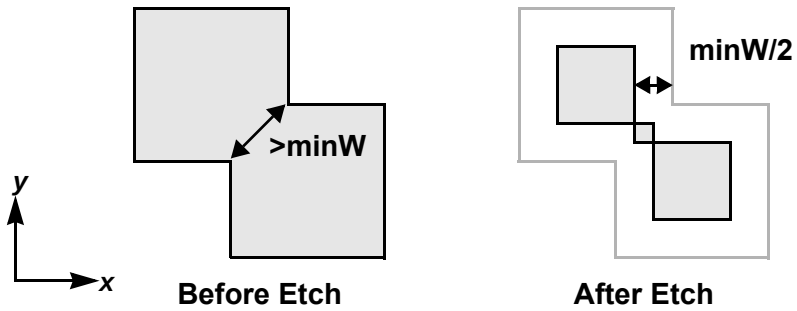
QTF includes some values that are not layer specific. These are defined in the **qtfParms** section.

qtfParms|**qtfParameters**
parameterDef(s)
qtfEndParms|**qtfEndParameters**

The **qtfParms** section includes the following parameters:

avgEtchFactor[**:**] ***pct*** (Default: 30%)
Specifies the expand and shrink values used (a fraction of min(**wMin**,**sMin**)/2) for calculating density-based width and spacing. The value can be a fraction (**0.01** to **0.99**) or a percentage (**1%** to **99%**). Avoid using a value larger than 70 percent. The lateral and vertical distance between two inside corners of a shape can be 70 percent of the minimum spacing without violating non-Manhattan DRC rules, causing an error is shown in Figure 1.

Figure 1: A shape with interior points spaced slightly larger than **minW** (at 45°), before and after an etch of **minW**/2**.**



conformalMethod[**:**] **current**|**deprecated** (Default: **current**)
Specifies whether to use the current or a deprecated method for handling multiple conformal dielectrics that ambiguously define the dielectrics on the side (**out**) over layers with sloped sidewalls. The **current** method recognizes successive conformal dielectrics with the same **out** value. The side dielectric is determined by the earlier one. The keyword **latest** is a synonym for **current**. The **deprecated** method generates side dielectrics for both conformal layers, giving ambiguous results.

corner [**=**|**:**] ***cornerString***]
Specifies the technology corner.

Note: This string is not used by gds2cap or QuickCap except as a comment in the log file.

deltaTemperature [**=|:**] ***temperature***

Specifies the operating temperature relative to the reference temperature for any thermal coefficients. The units should be consistent with any thermal-coefficient formulas. The operating temperature can only be defined one time. When **deltaTemperature** is specified, or when both **operatingTemperature** and **referenceTemperature** are specified, qtfx uses the relative operating temperature to generate **scaleR** values or tables for each layer with **TC1** or **TC2** values.

The following example specifies a relative operating temperature of 75:

```
deltaTemperature: 75
```

densityGrid [=|:] ***width***

Specifies the density grid to be used for automated gds2density runs (required to create density maps). A density grid is required for tools such as gds2cap that precalculates a density map on a grid and interpolate as necessary, rather than calculating the density in real time. If **densityGrid** is not specified, the grid size defaults to the smaller of (1) the smallest **densityWindow** width, and (2) half of the **densityWindow** width. The specified grid size cannot be larger than the minimum **densityWindow** width.

The following example specifies a density grid of 5 μm:

```
densityGrid: 5
```

densityPadding[**=|:**] **none**|**symmetric**|**topRight** (Default: **none**) *New QTF parameter*

Specifies how the layout bounds are padded to align with the density grid.

The arguments of the **densityPadding** parameter are

- **none** (default): The density grid is decreased independently in x and y so that the layout size is a multiple of the density-grid size.
- **symmetric**: The layout is symmetrically padded so that it is a multiple of the density-grid size.
- **topRight**: The layout is padded on the top (maximum y coordinate) and right (maximum x coordinate) so that it is a multiple of the density-grid size.

densityWindow [=|:] ***width*** [**@** ***weight***]]

Specifies the density window to be used for automated gds2density runs (required to create density maps). The ***weight*** value can have a **%** suffix, in which case it is interpreted as a percentage. Multiple density windows can be specified if the total weight is 1 (100%). The ***weight*** need not be specified for the last density window, in which case it receives a weight so that the total is 1 (100%). If **densityGrid** is not specified, the grid size defaults to half of the smallest **densityWindow** width.

The following example specifies a density window widths of 10 μm, 20 μm, and 50 μm with corresponding weights of 0.2 (20%), 0.4 (40%), and 0.4 (40%). The grid size is 5 μm, which is half the smallest window width.

```
densityWindow: 10 @ 20%
densityWindow: 20 @ 40%
densityWindow: 50
```

drawnResistanceLayer[**:**] **etchBased**
drawnResistanceLayer[**:**] **explicit**[**DrawnLayer**] (Default)
drawnResistanceLayer[**:**] **implicit**[**DrawnLayer**]

Specifies the drawn layer that gds2cap uses for QTF resistance calculation in the absence of a QTF etch. For **etchBased**, the resistance calculation uses dimensions in the final layer (**Wdr** is the same as **Wsi**, and **Ddr** is the same as **Dsi**) even with a gds2cap-based etch (non-QTF). For **explicit** (the default), the resistance calculation uses dimensions in the drawn layer only if it is explicitly named as the **drawn** property of the gds2cap **layer** statement. For **implicit**, the resistance calculation uses dimensions in the drawn layer even when it is only implied by an etch (non QTF) in the gds2cap **layer** statement. For more information, see the *gds2cap User Guide and Technical Reference*.

encrypt [**=|:**] [**weak|strong**]

Specifies the encryption level. For weak encryption, qtfx encrypts generated QTF data, but other tools such as RGEN, QCP, gds2cap, or QuickCap do not hide any information they generate. Support for strong encryption depends on the tool used.

externalDensity[**=|:**] ***density*** (Default: No density effect)

Specifies the density value (**0** to **1**, or **0%** to **100%**) outside the layout bounding box.

node [**=|:**] ***nodeString*** [[**,**] ***versionString***]

Specifies the technology node and version. Neither of these strings is used by gds2cap or QuickCap.

Note: The technology node and version are intended to represent a broader description than **technology**. This string is not used by gds2cap or QuickCap except as a comment in the log file.

The following example represents version v1.5 of the *28 nm* technology node.

```
technology: 28nm_M9, v1.5c
      node: 28nm, v1.5
```

operatingTemperature [**=|:**] ***temperature***
Specifies the operating temperature for any thermal coefficients. The units should be consistent with any thermal-coefficient formulas. The operating temperature can only be defined one time. When both **operatingTemperature** and **referenceTemperature** are specified, or when **deltaTemperature** is specified, qtfx uses the relative operating temperature to generate **scaleR** values or tables for each layer with **TC1** or **TC2** values.

The following example specifies an operating temperature of 100:

```
operatingTemperature: 100
```

polyLdir[**:**] **export**[**FromGperArea**] (Default)
polyLdir[**:**] **internal**[**ToGperArea**]
Specifies whether a **polyLdir** property of a **GperArea** table can determine the polygon lengthwise direction in general (**export**, the default) or whether it is only used for the **GperArea** calculation (**internal**). In gds2cap, **polyLdir** affects **polyL()** and **polyW()** functions, and the length direction segmenting high-aspect vias. For more information, see the *gds2cap User Guide and Technical Reference*.

processFoundry[**:|=**] ***foundryString***
processNode [**:|=**] ***nodeString***
processVersion [**:|=**] ***versionString***
processType [**:|=**] ***typeString***
processCorner [**:|=**] ***cornerString***
Specify process-related information. The qtfx tool allows general strings, but some tools might require floating-point values for the node (units of nm) and version. The gds2cap tool outputs any process-related information in the headers of output files.

referenceDirection [**=|:**] **x|y** (Default: **y**)
Specifies whether etch values are for an x or y reference direction. Any tools that implement QTF etches must swap directional etch tables if the reference direction is changed. See Directional Etch Tables on page 11-43 for related information.

referenceTemperature [**=|:**] ***temperature*** (Default: **25**)
Specifies the reference temperature for any thermal coefficients. The units should be consistent with any thermal-coefficient formulas. The default (25) is consistent with defaults for StarRC, HSPICE, and other applications. The reference temperature can only be defined one time. When both **operatingTemperature** and **referenceTemperature** are specified, or when **deltaTemperature** is specified, qtfx uses the relative operating temperature to generate **scaleR** values or tables for each layer with **TC1** or **TC2** values.

The following example specifies a reference temperature of 20:

```
referenceTemperature: 20
```

quickcap [**=|:**] ***quotedString***
Specifies a verbatim string to be passed to QuickCap. Multiple QuickCap verbatim strings can be specified, each using the **quickcap** keyword. Unlike the **qtfVerbatim** structure, the verbatim text must be in quotation marks.

The following example is a QuickCap **range** command that overrides the default value (based on **scale**) of QuickCap:

```
QuickCap: "range 75nm"
```

resolution [**=|:**] ***length***
Specifies the resolution. This is equivalent to gds2cap **max xyPosErr**, and QuickCap **resolution**. The resolution can only be defined one time.

The following example specifies a 1Å resolution (same as gds2cap/QuickCap default):

```
resolution: 1e-4
```

rgen [**=|:**] ***quotedString***
Specifies a verbatim string to be passed to RGEN. Multiple RGEN verbatim strings can be specified, each using the **rgen** keyword. Unlike the **qtfVerbatim** structure, the verbatim text must be in quotation marks.

scale [**=|:**] ***length***
Specifies the scale of the problem. This is passed to QuickCap to serve as the value to base all default values for length parameters of QuickCap. The scale can only be defined one time.

The following example indicates a 45 nm process:

```
scale: 0.045
```

scaleLayout [**=|:**] ***factor***
Specifies a factor for scaling layout data. This is equivalent to the gds2cap **scaleXY** declaration. The layout scale can only be defined one time.

The following example specifies that all layout data is to be scaled by a factor of 0.9 (90%):

```
scaleLayout: 0.9
```

singleAttachedVia[**:**] **stub|sublayer** (Default: **sublayer**)
Specifies whether a via with a single **attach** layer is treated as a stub or as a sublayer. (See **attach** on “attach (layer names) Conductor stack” on page 11-13.) For **sublayer** (the default), the layer inherits any **ignoreCoupling**, **groundCoupling**, and graphics behavior of the layer to which it is attached.

technology [**=|:**] ***technologyString*** [[**,**] ***revString***]
Specifies a description of the technology and an optional revision string. A quoted string is required to define a multiple-word description.

Note: The technology and revision is intended to represent a narrower description than **node**. Neither of these strings is used by gds2cap or QuickCap except as comments in the log file

The following example represents revision v1.5c of the *28nm_M9* technology.

```
technology: 28nm_M9, v1.5c
      node: 28nm, v1.5
```

translator [**=|:**] ***string***
Specifies the name of the translator used to generate the original QTF data. The qtfx tool generates this parameter in the form qtfx_*ID* when translating without a QTF format. The translator parameter allows a program reading QTF to address any issues that were uncovered since the original translation to QTF.

trapMethod[**:**] **parmBased** (Default)
trapMethod[**:**] **fixedBias**
trapMethod[**:**] f**ixedSideTan**
trapMethod[**:**] **transitional**
trapMethod[**:**] **deprecated**
Specifies the trap method to be used. The **parmBased** method applies the fixed-bias or fixed-sideTan approach for each layer, depending whether that layer has defined values for **bias**[**X**|**Y**] or synonym **trap**[**X**|**Y**], or for **sideTan**[**X**|**Y**]. The **fixedBias** method uses a fixed bias, even for layers with defined values for **sideTan**[**X**|**Y**]. The **fixedSideTan** method uses a fixed sideTan, even for layers with defined values for **bias**[**X**|**Y**] or synonym **trap**[**X**|**Y**].

The **transitional** and **deprecated** methods support earlier behavior, which is similar to the **fixedSideTan** behavior. These methods can introduce errors involving interactions between global thickness variation and sloped sides.

The **-qtfTrapMethod** command-line option of gds2cap takes precedence over the **trapMethod** QTF parameter.

Verbatim

Verbatim data (for other programs) is defined in **qtfVerbatim** sections.

qtfVerbatim tech|rgen|QuickCap
verbatim
qtfEndVerbatim

Specifies verbatim text (*not* in quotation marks) to be passed to gds2cap, to RGEN, or to QuickCap. Initial spaces are ignored.

The following example contains QuickCap **dRes0** and **dRes** commands that override the default value (based on **scale**) of QuickCap:

```
qtfVerbatim QuickCap
  dRes0 0.01 um
  dRes  0.02 um
qtfEndVerbatim
```

With appropriate data in a **qtfVerbatim tech** block, a QTF file can be used directly as a gds2cap technology file.

Layer Properties (Stacks)

All layer properties are assigned in one of three types of stacks. A stack structure is delimited by keywords of the form **qtf...Stack** and **qtfEnd...Stack**. Lateral-conductor layers and via layers are defined in *conductor* stacks. Planar and conformal dielectric layers are defined in *dielectric* stacks. Adjust-depth layers are defined in *adjust-depth* stacks. For examples, see “Lateral Conductors” on page 11-23, “Vias” on page 11-24, “Planar Dielectric Layers” on page 11-27, “Conformal Dielectric Layers” on page 11-28, and “Stack Variation” on page 11-29.

The first line, the *stack header*, contains keywords describing the properties defined in the column of the stack. The first column must always be **name**, the name of the layer.

Each subsequent line (until the **qtfEnd...Stack** keyword) contains an entry for each keyword in the stack header. The first entry must be a unique layer name. Depending on the property indicated in the stack header, an entry in the table must be a layer name, a numeric value, a table name (see “QTF Ignored Data” on page 11-35), or multiple dashes (**--**, **---**, and so on), to indicate that the property is not specified. A line in the body of the stack can contain fewer items than the header line, in which case the properties in the last columns are properties that are not specified. A reference to a table that QTF can simplify to a single value is equivalent to a reference to that value.

After the **name** column, properties applicable to the stack can be listed in any order. The only exception is the **attach** column for via layers in a conductor stack. The **attach** column must be last, if anywhere. Via layers require two or more **attach** layers.

A stack header can include the names of any property, even properties that do not apply to any layers in the stack. The only exception is the **attach** property, which can only be specified in conductor stacks. For a description of the properties that can or must be defined for the various layer types, see “Dielectric Layers” on page 11-26 (planar and conformal dielectrics), and “Stack Variation” on page 11-29 (adjust-depth layers).

The following elements can appear in a stack.

above (layer reference) *Any stack*

Specifies the name of a layer just below the current layer. The referenced layer can be in any stack. The bottom (**z0**) of the current layer (in the stack) is set to the top (**z1**) of the **above** layer.

Note: A layer in a stack cannot have both **z0** and **above** specified.

AdrMin (positive values) *Conductor stack*
AdrMax

Specifies the range of the **Adr** table argument. If either is specified, both must be specified. **AdrMin** and **AdrMax** are decreased and increased, respectively, to reflect the range of the **Adr** table argument in as-input tables related to the interconnect or via layer (before any table reduction). **AdrMin** must be less than **AdrMax**.

area (positive value) *Conductor stack*

Specifies the area.

attach (layer names) *Conductor stack*

Specifies the conductor layers associated with a via layer. The **attach** column must be last. When only a single attach layer is specified, the via layer is treated like a stub layer or a sublayer according to the value of the **singleAttachedVia** QTF parameter: stub or a sublayer (the default), see page 11-10. Each **attach** property value must be a conductor-layer name or the string **ground** or **substrate** (synonyms). A via is assumed to contact two layers. If a contact might attach to different layers, they must be specified in the order of preference. For example: M1, NSD, PSD, PTAP, and NTAP.

below (layer reference) *Any stack*

Specifies the name of a layer just above the current layer. The referenced layer can be in any stack. The top (**z1**) of the current layer (in the stack) is set to the bottom (**z0**) of the **below** layer.

Note: A layer in a stack cannot have both **z1** and **below** specified.

bias[**X**|**Y**] (value) *Conductor stack*
trap[**X**|**Y**]

Specifies the width bias (width at the top minus the width at the bottom). The **bias** and **trap** keywords are equivalent. Use **biasX** and **biasY** when the bias is different in x and y. A **bias**[**X**|**Y**] property must be a value, not a table reference. When translating a QTF file, qtfx generates the keyword **bias** in the translated file. A layer cannot define values for both **bias**[**X**|**Y**] and **sideTan**[**X**|**Y**]. If the **trapMethod** QTF parameter is **fixedSideTan**, then a **bias**[**X**|**Y**] value is effectively a **sideTan**[**X**|**Y**] value (*sideTan* = *bias*nomThk/2*), and QuickCap extracts capacitance using the **fixedSideTan** method. Otherwise, QuickCap extracts capacitance using the **fixedBias** method.

desc (string) *Any stack*

Specifies a description of the conductor layer. The **-stdTypes** qtfx option (on page 9-6) generates standard types.

Though the qtfx tool accepts any string, it recognizes the following *standard* via types:

TAP
Connection between diffusion and well.

CONT
Connection between the lowest metal layer and either poly or diffusion.

DCONT, DCONTx
Connection between the lowest metal layer and diffusion (a diffusion contact). When qtfx finds multiple diffusion-contact layers (DCONT, N_DCONT and P_DCONT, for example), it designates one representative layer as **DCONT** and the others as **DCONTx**.

PCONT, PCONTx
Connection between the lowest metal layer and poly (a poly contact). When qtfx finds multiple poly contact layers (PCONT, N_PCONT and P_DCONT, for example), it designates one representative layer as **PCONT** and the others as **PCONTx**.

VIA
Connection between metal layers, except for connections involving **MET0**.

VIA0
Connection between **MET0** and any metal layers.

Though the qtfx tool accepts any string, it recognizes the following *standard* interconnect types:

OD, **ODx**
Diffusion layer. The qtfx tool generates types. When qtfx finds multiple diffusion layers (SD, NSD, and PSD, for example), it designates one representative layer as **OD** and the others as **ODx**.

GPOLY, **GPOLYx**
Gate-poly layer. When qtfx finds multiple gate-poly layers (GATE, N_GATE, and P_GATE, for example), it designates one representative layer as **GPOLY** and the others as **GPOLYx**.

FPOLY, **FPOLYx**
Field-poly layer. When qtfx finds multiple field-poly layers (FPOLY, N_FPOLY, and P_FPOLY, for example), it designates one representative layer as **FPOLY** and the others as **FPOLYx**.

POLY, **POLYx**
Poly layer (when gate poly and field poly cannot be identified separately). When qtfx finds multiple poly layers (POLY, N_POLY, and P_POLY, for example), it designates one representative layer as **POLY** and the others as **POLYx**.

MET0, **MET1**, **METx**
Upper interconnect layers. The **MET1** layer is generally the lowest interconnect layer. Higher interconnect layers are **METx**. Some fabrication processes can include **MET0**.

down (value) *Dielectric stack*
Specifies the distance from the bottom of the underlying conductor down to the bottom of a conformal dielectric layer. The **down** property must be a value, not a table reference, and must include any effect of underlying conformal layers. For example, for a DMG2b layer of width 0.02 µm that is outside of a DMG2a layer of width 0.01 µm, specify the DMG2b layer first with a **down** value of **0.03**, followed by the DMG2a layer with a value of **0.01**. A conformal layer requires **down** or **z0**, but cannot have both.

dZ (table) *Conductor stack*
dZ1–**dZ8**
Specifies a change in height. The values must be in tables. The only acceptable constant is **0**, in which case the value is ignored. You must apply the height change before you perform global thickness variation (**qtfAdjustDepth** data). The enumerated forms allows specification of height changes from different sources. The change in height from each source is additive. For example, **dZ** values 0.2 (microns) and -0.1 change the final height by 0.1. For any layer, a **dZ** value cannot be specified with **dZ1**–**dZ8**. A **dZ** value depends on layer class if represented by a 1D indexed table. For information, see “Indexed Tables” on page 11-40.

eps (positive value) *Dielectric stack*
Specifies the dielectric constant of a planar or conformal dielectric layer. The **eps** property must be a value, not a table reference.

eps (table, (xy) (undirected) table) *Conductor stack*
Specifies an isotropic (uniform) dielectric in the gap between wires. This must be a table that is only a function of **Sdr** or **Ssi**. An **eps** table can be undirected, see “Indexed Tables” on page 11-40. An undirected table includes indices that reference two layer classes by number. It is undirected because a table indexed [i][j] is also used for [j][i]. Only one of the **eps**, **epsXY**, and **epsXYscale** properties can be specified for any given conductor layer.

epsAvg *Conductor stack*
Defines an average background value. This implies that dielectrics generated from an **eps** or **epsXY** property do not have any effect when the value matches with the **epsAvg** value. An **epsAvg** value can only be specified on a layer with the **eps** or **epsXY** property.

epsXY (table, (xy) (undirected) table) *Conductor stack*

Specifies the effective dielectric value in the gap between wires, to be evaluated separately in the x- and y-directions. This must be a table that is only a function of **Sdr** or **Ssi**. An **epsXY** table can be directional (see “Directional Tables” on page 11-38), undirected (see “Indexed Tables” on page 11-40), or both. An undirected table includes indices that reference two layer classes by number. It is undirected because a table indexed [i][j] is also used for [j][i]. Only one of the **eps**, **epsXY**, and **epsXYscale** properties can be specified for any given conductor layer.

epsXYscale (table, or undirected indexed table) *Conductor stack*

Specifies scale factors to be applied to lateral capacitance. Similar to **epsXY**, the table must reference **Sdr** or **Ssi**. An **epsXYscale** table can be directional (see “Directional Tables” on page 11-38), undirected (see “Indexed Tables” on page 11-40), or both. An undirected table includes indices that reference two layer classes by number. It is undirected because a table indexed [i][j] is also used for [j][i]. Only one of the **eps**, **epsXY**, and **epsXYscale** properties can be specified for any given conductor layer.

etch (value, (xy) (ndexed) table) *Conductor stack*

Specifies the etch of a conductor layer. The **etch** property, which can be a value or table name, specifies the distance that an edge moves to convert a drawn layer into a local (*silicon*, or *etched*) layer. An etch table, and any tables it references, cannot reference silicon density, spacing, or width (**Dsi**, **Ssi**, or **Wsi**). A conductor layer cannot have both an **etch** and a **width** property. An **etch** table can be directional (see “Directional Tables” on page 11-38), class based (see “Indexed Tables” on page 11-40), or both.

etchL (value or table reference) *Conductor stack*
etchW

Specifies the etch of the larger dimension (length, **etchL**) or smaller dimension (width, **etchW**).

RGEN uses these properties for a via etch when they are defined. The gds2cap tool, however, uses **etch**, which qtfx generates together with equivalent **etchL** and **etchW** tables when the ITF or iRCX input file specifies these etches.

etch0 (value, (xy) (ndexed) table) *Conductor stack*

Specifies a retarget etch, which generates an effective drawn layer. References from other tables to **Ddr**, **Sdr**, or **Wdr** reference the density, spacing, or width *after* the retarget etch. An **etch0** table can be directional (see “Directional Tables” on page 11-38), class based (see “Indexed Tables” on page 11-40), or both.

A directional retarget etch should be consistent with the value of the QTF **referenceDirection** parameter direction, which has the default value of **y** (if not specified in the **qtfParms** section). See **referenceDirection** on page 11-9 and “Table Arguments” on page 11-43 for more information related to directional etches.

etch1–**etch8** (value, (xy) (indexed) table) *Conductor stack*
Specifies one of up to eight etches of a conductor layer. The **etch*N*** property, which can be a value or table name, specifies the distance that an edge moves in one step of a multiple-etch process to convert a drawn layer into a local (*silicon* or *etched*) layer. Neither an etch table nor any tables it references can reference silicon density, spacing, or width (**Dsi**, **Ssi**, or **Wsi**). A conductor layer *can* have both **etch** and **etch*N*** properties. In this case, **etch**, should be used only for an approximate representation of the etch. When only one **etch*N*** property is specified. **etch** cannot be specified. When possible, qtfx merges two successive etches if at least one of them is a constant. Etch tables can be directional (see “Directional Tables” on page 11-38), class based (see “Indexed Tables” on page 11-40), or both.

etchR (value, table, or xy table reference) *Conductor stack*
Specifies the effective etch of a lateral-conductor layer to be used for resistance calculations. The **etchR** property must be a value or table name. The resistance etch is applied to the drawn layer for evaluation of **rhoDr** and **rshDr**. Specifying **etchR** without specifying either **rhoDr** or **rshDr** generates an error. The **etchR** value depends on layer class if represented by a 1D indexed table. For information, see “Indexed Tables” on page 11-40.

gPerAreaDr (positive value or table reference) *Conductor stack*
gPerAreaSi
Specifies the conductivity per unit area of a lateral-conductor layer to be used for resistance calculations. A **gPerAreaDr** or **gPerAreaSi** property must be a value or table name. The resistance of a via is given by *drawnArea*/**gPerAreaDr**, or *localArea*/**gPerAreaSi**. The **gPerAreaDr**|**Si** value depends on layer class if represented by a 1D indexed table.
For information, see “Indexed Tables” on page 11-40.

layer (layer name) *Adjust-depth or dielectric stack*
Specifies the conductor layer associated with an adjust-depth or conformal layer. The **layer** property must be the name of a conductor layer.

marker (string) *Dielectric stack*
Specifies a marker layer that must touch **layer** of a conformal dielectric to generate the dielectric region. The marker layer does not need to be a QTF layer.

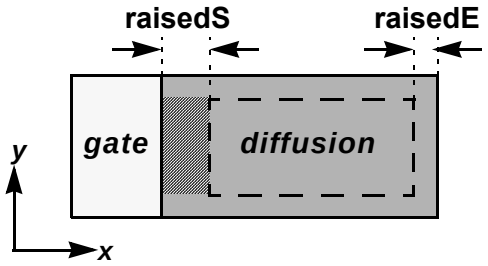
out (value) *Dielectric stack*
Specifies the lateral (xy) expansion of the underlying conductor for a conformal dielectric layer. The **out** property must be a value, not a table reference, and must include any expansion of underlying conformal layers. For example, for a LINER layer of width 0.2 µm that is outside of a SPACER layer of width 0.1 µm, specify the LINER layer first with an **out** value of **0.3**, followed by the SPACER layer with a value of **0.1**.

raisedE (value) *Conductor stack*
raisedK
raisedS
raisedT

Specifies that a section of a conductor **raisedE** from the edge, and **raisedS** from gate layers is raised by thickness **raisedT**. Figure 11-1 shows a diffusion region (dark), a gate region (light), and the raised region (dashed outline). The **raisedK** value defines the dielectric value of a region between the raised section and an adjacent gate layer,

Note: Not fully implemented. Although these values are recognized, they are not incorporated in any integrated schemes.

Figure 11-1: For a diff region (dark), the raised area (dashed region) is distance raisedE from the edge and spaced raisedS from the gate region (light). The region between the raised diffusion region and the gate has a dielectric value of raisedK.



rContact (positive value or table reference) *Conductor stack*

Specifies the contact resistance of a via layer. A **rContact** property must be a value or table name.

rhoDr (positive value or table reference) *Conductor stack*
rhoSi

Specifies the resistivity of a lateral-conductor layer to be used for resistance calculations. The **rhoDr** and **rhoSi** properties must be values or table names. The number of squares of materials is based on the drawn layer for **rhoDr**, or on the local (*silicon*, or *etched*) layer for **rhoSi**. The associated thickness is taken to be **thkR**, as specified, or as calculated from the nominal values (**z0** and **z1**) and any thickness adjustments (**thkB** and **thkT**). The **rhoDr**|**Si** value depends on the layer class if represented by a 1D indexed table. For information, see “Indexed Tables” on page 11-40.

rshDr (positive value or table reference) *Conductor stack*
rshSi

Specifies the sheet resistance of a lateral-conductor layer to be used for resistance calculations. The **rshDr** and **rshSi** properties must be values or table names. The number of squares of materials is based on the drawn layer for **rshDr**, or on the local (*silicon*, or *etched*) layer for **rshSi**. The **rshDr**|**Si** value depends on the layer class if represented by a 1D indexed table. For information, see “Indexed Tables” on page 11-40.

scaleR (value or table reference) *Conductor stack*

Specifies a scaling factor to apply to the resistance. A **scaleR** property must be a value or table name. When the relative operating temperature *dT* is defined (**deltaTemperature**, or **operatingTemperature** - **referenceTemperature**), the qtfx program generates **scaleR** from **TC1** and **TC2**: *scale* = 1 + **TC1****dT* + **TC2****dT*^{2}.

SdrMin (positive values) *Conductor stack*
SdrMax

Specifies the range of the **Sdr** table argument. If either is specified, both must be specified. **SdrMin** and **SdrMax** are decreased and increased, respectively, to reflect the range of the **Sdr** table argument in as-input tables related to the interconnect or via layer (before any table reduction). **SdrMin** must be less than **SdrMax**.

sMin (positive value) *Conductor stack*

Specifies the minimum drawn spacing of a conductor layer. The **sMin** property must be a value, not a table reference.

sideTan[**X**|**Y**] (value)

Specifies the tangent of the slope of the sidewalls. Use **sideTanX** and **sideTanY** when the tangent is different in x and y. A **sideTan**[**X**|**Y**] property must be a value, not a table reference. The **sideTan** is the offset between the top and bottom lateral coordinate divided by the thickness. For a positive **sideTan** value, a shape is wider at the top than at the bottom. A layer cannot define values for both **sideTan**[**X**|**Y**] and **bias**[**X**|**Y**]. If the **trapMethod** QTF parameter is **fixedBias**, then a **sideTan**[**X**|**Y**] value is effectively a **bias**[**x**|**y**] value (*bias* = 2**sideTan*/*nomThk*), and QuickCap extracts capacitance using the **fixedBias** method. Otherwise, QuickCap extracts capacitance using the **fixedSideTan** method.

TC1 (value or table reference) *Conductor stack*
TC2

Specifies the thermal coefficients of a conductor layer to be used for resistance calculations. The **TC1** and **TC2** properties must be values or table names. The resistance is scaled by a factor of 1 + **TC1****dT* + **TC2****dT*^{2}, where *dT* is the operating temperature minus the reference temperature.

Layers without any resistance parameter can include the **TC1** and **TC2** parameters that allows them to be applied to resistance values defined outside the QTF database.

thk (value or layer reference) *Any stack*

Specifies the nominal layer thickness. When neither **z0** nor **above** is specified, **z0** is set to **z1**-**thk**. When neither **z1** nor **below** is specified, **z1** is set to **z0**+**thk**. When both **z0** and **z1** can be determined without **thk**, **thk** must be equivalent to **z1**-**z0**.

thkB (positive table reference) *Conductor stack*
thkB1–**thkB8**

Specifies the bottom thickness table of a conductor layer. The **thkB** property, which must be a table name, specifies the thickness of the layer from the nominal top (**z1**) to the bottom of the layer. Such a table does not affect the absolute heights of the stack (other conductors, and dielectric layers). Stack variation is only affected by the **thkT** property of an adjust-depth layer. A thickness table that reduces to a single value generates an error.

The enumerated forms allows specification of thickness from different sources. The change in thickness from each source is the table value minus the nominal thickness. For a nominal thickness of 1 (micron), for example, **thkB** values of 1.1 and 1.2 with a **thkT** value of 0.95 gives a final thickness of 1.25. For any layer, a **thkB** value cannot be specified with **thkB1**–**thkB8**.

A **thkB** value depends on the layer class if represented by a 1D indexed table. For information, see “Indexed Tables” on page 11-40.

thkR (positive value or table reference) *Conductor stack*
thkR1–**thkR8**

Specifies one of up to eight independent thickness values to be used for resistance calculations (**rhoDr**, **rhoSi**). The **thkR** or **thkR*N*** property must be a value or table name. The **thkR** keyword is equivalent to **thkR1**. When you specify multiple **thkR** tables for a layer, the total thickness is the nominal thickness, *nomT*, plus **thkR1**-*nomT*, plus **thkR2**-*nomT*, and so on. When all table arguments in one **thkR** table are included in another, qtfx combines the two tables. If no **thkR** or **thkR*N*** property is specified, the resistance calculation for **rhoDr** or **rhoSi** uses the thickness as calculated from the nominal values (**z0** and **z1**) and the thickness adjustments (**thkB** and **thkT**).

A **thkR** value depends on the layer class if represented by a 1D indexed table. For information, see “Indexed Tables” on page 11-40.

thkT (positive value or table reference) *Any stack*
thkT1–**thkT8**

Specifies the thickness of a conductor (lateral conductor or via), planar dielectric, or adjust-depth layer. As a value, **thkT** property can be specified only if it is the same as **z1** - **z0**, or if it can be used to derive **z0** from **z1**, or **z1** from **z0**. The **thkT** property can be a table name in lateral-conductor layers, and must be a table name in adjust-depth layers, for which it is a required property.

For a lateral-conductor layer (conductor stack), a **thkT** table specifies the *top* thickness of the layer (from the nominal bottom (**z0**) to the top of the layer), contrasted with **thkB**, which specifies the *bottom* thickness. Such a table does not affect the absolute heights of the stack (other conductors, and dielectric layers). Stack variation is only affected by the **thkT** property of an adjust-depth layer.

For an adjust-depth layer, **thkT** specifies the name of a table defining the actual layer thickness. A thickness table that reduces to a single value generates an error. Any adjust-depth table reference to drawn or local spacing or width (**Sdr**, **Ssi**, **Wdr**, **Wsi**) is interpreted as an *effective* spacing or width value (**SdrDty**, **SsiDty**, **WdrAvg WsiAvg**), extrapolated from the relationship between density and layer expansion. Referencing a set of class-based (indexed) tables, the value is based on a weighted average of the tables. See ***table1d*[avg]** on page 11-42

The enumerated forms allows specification of thickness from different sources. The change in thickness from each source is the table value minus the nominal thickness. For a nominal thickness of 1 (micron), for example, **thkT** values of 1.1 and 1.2 with a **thkB** value of 0.95 gives a final thickness of 1.25. For any layer, a **thkT** value cannot be specified with **thkT1**–**thkT8**.

A **thkT** value depends on the layer class if represented by a 1D indexed table. For information, see "Indexed Tables" on page 11-40.

up (value) *Dielectric stack*

Specifies the distance from the top of the underlying conductor up to the top of a conformal dielectric layer. The **up** property must be a value, not a table reference, and must include any effect of underlying conformal layers. For example, for a PASS4b layer of width 2 μm that is over a PASS4a layer of width 1 μm, specify the PASS4b layer first with an **up** value of **3**. A conformal layer requires **up** or **z1**, but cannot have both.

vFillType (string) *Conductor stack*
vFillRatio (positive value)
vFillSpacing (positivevalue)
vFillWidth (positivevalue)

These *fill* properties are not used by gds2cap.

WdrMin (positive values) *Conductor stack*
WdrMax

Specifies the range of the **Wdr** table argument. If either is specified, both must be specified. **WdrMin** and **WdrMax** are decreased and increased, respectively, to reflect the range of the **Wdr** table argument in as-input tables related to the interconnect or via layer (before any table reduction). **WdrMin** must be less than **WdrMax**.

width (positive table reference) *Conductor stack*

Specifies the width table of a conductor layer. The **width** property, which must be a table name, specifies the width of the local (*silicon*, or *etched*) layer as a function of the drawn layer. A width table, and any tables it references, must not reference silicon density, spacing, or width (**Dsi**, **Ssi**, or **Wsi**). A width table or one of the tables it references, *must* reference drawn width (**Ddr**). A conductor layer cannot have both a **width** and an **etch** property. A width table that reduces to a single value generates an error.

Because all width tables are converted to etch tables, appropriately simplified, qtfx outputs etch tables rather than width tables.

wMin (positive value) *Conductor stack*
Specifies the minimum drawn width of a conductor layer. The **wMin** property must be a value, not a table reference.

z0 (value or layer reference) *Any stack*
Specifies the height of the bottom of the layer. The **z0** property must be a value or a layer reference, not a table reference. As a table reference, **z0** of the current layer (in the stack) is set to as **z0** of the referenced layer. If **z0** and **z1** are both specified, **z0** must be less than or equal to **z1**. Values **z0** and **z1** cannot be equal if any property implies thickness is not zero and is also defined: **bias**, **sideTan**, **thkB**[**1**-**8**], **thkT**[**1**-**8**], **thkR**, **rhoDr**, or **rhoSi**.

In conductor stacks, if **z0** is not specified and cannot otherwise be inferred, both **thkT** (as a value) and **z1** must be specified.

Note: A layer in a stack cannot have both **z0** and **above** specified.

z1 (value or layer reference) *Any stack*
Specifies the height of the top of the layer. The **z1** property must be a value or a layer reference, not a table reference. As a table reference, **z1** of the current layer (in the stack) is set to **z1** of the referenced layer. If **z0** and **z1** are both specified, **z0** must be less than or equal to **z1**. Values **z0** and **z1** cannot be equal if any property implies thickness is not zero and is also defined: **bias**, **sideTan**, **thkB**, **thkT**, **thkR**, **rhoDr**, or **rhoSi**.

In conductor stacks, if **z1** is not specified and cannot otherwise be inferred, both **thkT** (as a value) and **z0** must be specified.

Note: A layer in a stack cannot have both **z1** and **below** specified.

Conductor Layers

Conductor layers (lateral conductors, and vias) are defined in **qtfConductorStack** sections.

qtfConductorStack
stackHeader
conductor(s)
qtfEndConductorStack

Lateral conductors and vias are generally defined in separate conductor stacks because some data only applies to lateral conductors, and some only applies to vias.

Lateral Conductors

Lateral-conductor layers include gds2cap layers that are functionally implemented with layer types of **float**, **ground**, **interconnect**, or **resistor**, but *not* **via**. Lateral conductors can be defined in any order. Properties relevant to lateral-conductor layers are as follows:

above (layer reference): Can be used instead of **z0**
AdrMin, **AdrMax** (positive values): Optional
below (layer reference): Can be used instead of **z1**
area (positive value): Optional
bias (value): Optional (not compatible with **biasX**, **biasY**, or **sideTan**[**X**|**Y**])
biasX, **biasY** (value): Optional (not compatible with **bias** or **sideTan**[**X**|**Y**])
desc (string): Optional
dZ, dZ1–**dZ8** (table): Optional
eps (table): Optional
epsAvg (value): Optional
epsXY (table): Optional
epsXYscale (table): Optional
etch (value, table or xy table reference): Optional, but cannot be specified with **width**
etchL (value or table reference): Optional
etchW (value or table reference): Optional
etch0 (value, table, or xy table reference): Optional
etch1–**etch8** (value, table, or xy table reference): Optional
etchR (value, table, or xy table reference): Optional (requires **rhoDr** or **rshDr**)
name (string): Required (first column)
raisedE, **raisedK**, **raisedS**, **raisedE** (value): Optional
rhoDr, **rhoSi**, **rshDr**, **rshSi** (positive value or table reference): Optional
scaleR (value or table reference). Optional (requires **rhoDr**, **rhoSi**, **rshDr**, or **rshSi**).
SdrMin, **SdrMax** (positive values): Optional
sMin (positive value): Optional
sideTan (value): Optional (not compatible with **sideTanX**, **sideTanY**, or **bias**[**X**|**Y**])
sideTanX, **sideTanY** (value): Optional (not compatible with **sideTan** or **bias**[**X**|**Y**])
TC1 (value or table reference): Optional
TC2 (value or table reference): Optional
thkB, **thkB1**–**thkB8** (positive table reference): Optional
thkR, **thkR1**–**thkR8** (positive value or table reference): Optional (requires **rhoDr** or **rhoSi**)
thkT, **thkT1**–**thkT8** (positive value or table reference): At least two of **z0**, **z1**, and **thkT**[**1**–**8**] (as a value) must be specified
vFillRatio (positive value): Optional
vFillSpacing (positive value): Optional
vFillType (string): Optional

vFillWidth (positive value): Optional
WdrMin, **WdrMax** (positive values): Optional
width (positive table reference): Optional, but cannot be specified with **etch**
wMin (positive value): Optional
z0 (value): At least two of **z0**, **z1**, and **thkT** (as a value) must be specified
z1 (value): At least two of **z0**, **z1**, and **thkT** (as a value) must be specified

In the following example that defines lateral-conductor layers, M2 and M3 reference the same table for **etch** and the same table for **rhoSi**.

```
qtfConductorStack
   name z0   z1   bias  etch  width    rhoSi  rshDr
   M4   2.5  3    0.03  ---   width_MC ---    0.01
   M3   1.6  1.9  0.02  ---   width_MB rho_MB ---
   M2   1    1.3  0.02  ---   width_MB rho_MB ---
   M1   0.5  0.7  0.01  ---   width_MA rho_MA ---
   POLY 0.26 0.36 ---   0.02  ---      ---    12
   DIFF 0.15 0.25 ---   ---   ---      ---    10
qtfEndConductorStack
```

The QTF language assigns properties for *physical* layers. *Functional* conductor layers can be defined in gds2cap. For example, the gds2cap technology file can include the following without a QTF declarations.

```
beginConductor DIFF(2)
   interconnect PSD = (DIFF - POLY) - NWELL clearR rsh=9
   interconnect NSD = (DIFF - POLY) * NWELL
endConductor
```

PSD and NSD inherit properties from QTF. The **clearR** property of PSD allows the user to override the QTF resistance information.

Vias

Via layers include gds2cap layers that are functionally implemented with the **via** layer type. Vias can be defined in any order. Properties relevant to via layers are as follows:

above (layer reference): Can be used instead of **z0**
AdrMin, **AdrMax** (positive values): Optional
area (positive value): Optional
attach (layer names): Required (one entry per stub layer or sublayer, two or more entries per via)

below (layer reference): Can be used instead of **z1**
bias (value): Optional not compatible with **biasX**, **biasY**, or **sideTan**[**X**|**Y**])
biasX, **biasY** (value): Optional (not compatible with **bias** or **sideTan**[**X**|**Y**])
desc (string): Optional
eps (table): Optional
epsAvg (value): Optional
epsXY (table): Optional
etch (value or table reference): Optional, but cannot be specified with **width**
etchL (value or table reference): Optional
etchW (value or table reference): Optional
etch0 (value, table, or xy table reference): Optional
etch1–**etch8** (value, table, or xy table reference): Optional
gPerAreaDr (positive value or table reference): Optional
gPerAreaSi (positive value or table reference): Optional
name (string): Required (first column)
rContact (positive value) Optional
scaleR (value or table reference). Optional (requires **rhoDr**, **rhoSi**, **rshDr**, or **rshSi**).
SdrMin, **SdrMax** (positive values): Optional
sMin (positive value): Optional
sideTan (value): Optional (not compatible with **sideTanX**, **sideTanY**, or **bias**[**X**|**Y**])
sideTanX, **sideTanY** (value): Optional (not compatible with **sideTan** or **bias**[**X**|**Y**])
TC1 (value or table reference): Optional
TC2 (value or table reference): Optional
thkT, **thkT1**–**thkT8** (positive value): Cannot specify just one of **z0**, **z1**, and **thkT**[**1**–**8**] (via depth can be inferred from attach layers)
WdrMin, **WdrMax** (positive values): Optional
width (positive table reference): Optional, but cannot be specified with **etch**
wMin (positive value): Optional
z0 (value): Cannot specify just one of **z0**, **z1**, and **thkT** (via depth can be inferred from attach layers)
z1 (value): Cannot specify just one of **z0**, **z1**, and **thkT** (via depth can be inferred from attach layers)

In the following example that defines via layers, PCONT and DCONT can be defined as separate physical layers even if they are derived from the same layer (CONT).

```
qtfConductorStack
   name  z0   z1  rContact attach
   VIA3  1.9  2.5 4        M4 M3
   VIA2  1.3  1.6 6        M3 M2
   VIA1  0.7  1   8        M2 M1
   PCONT 0.36 0.5 40       M1 POLY
```

```
   DCONT 0.25 0.5 70       M1 DIFF
qtfEndConductorStack
```

Alternatively, PCONT and DCONT can be defined as a single physical layer, in which case the gds2cap technology file is required to override the inherited depth.

Resistance Corners

Resistance corners are defined in **qtfResistanceCorners** sections, as follows:

qtfResistanceCorners
  name corner ***resistanceProperties***
  corner(s)
qtfEndResistanceCorners

The first two columns of a corner must be the name of a layer previously specified in a QTF conductor stack (**name**) and the name of the corner (**corner**). Subsequent columns consist of any required resistance related properties: **rshDr**, **rshSi**, **rhoDr**, **rhoSi**, **gPerAreaDr**, **gPerAreaSi**, **Rcontact**, **etchR**, **thkR**, **TC1**, **TC2**, and **rScale**. Interconnect-specific properties such as **rshDr** cannot be specified for via layers, and via-specific properties such as **Rcontact** cannot be specified for interconnect layers. Different layers can reference the same resistance corner. The same layer can be specified multiple times with different resistance corners.

Any resistance corner with *no* resistance properties (**rshDr**, **rshSi**, **rhoDr**, and **rhoSi** for interconnects; or **gPerAreaDr**, **gPerAreaSi**, and **Rcontact** for vias) inherits those properties from the nominal corner (the layer in the **qtfConductorStack** block). Any resistance corner with neither a **TC1** nor a **TC2** property inherits those properties from the nominal corner. Any resistance corner with neither an **etchR** nor a **thkR** property inherits those properties from the nominal corner.

Dielectric Layers

Planar and conformal dielectric layers are defined in **qtfDielectricStack** sections.

qtfDielectricStack
  stackHeader
  dielectric(s)
qtfEndDielectricStack

A *conformal layer*, here, refers only to the component that is around an underlying conductor, and does not include the planar component. Planar and conformal components are generally defined in separate dielectric stacks because some data only applies to planar components, and some only to conformal components.

Planar Dielectric Layers

A planar dielectric layer is completely defined by three values, the bottom z value, the top z value, and the dielectric constant. In the QTF file, dielectrics *must* be defined from the top down. If the top dielectric does not specify a **z1** value, it is taken to specify the background dielectric constant. The bottom of the bottom dielectric is taken to be the top of the ground plane. The **z0** value of a dielectric (if defined) must match the **z1** value of the next lower dielectric. If the **z0** value of a dielectric is not defined, it is set to the **z1** value of the next lower dielectric.

Planar dielectric layers can include the following properties:

above (layer reference): Can be used instead of **z0**
below (layer reference): Can be used instead of **z1**
desc (string): Optional
eps (positive value): Required
name (string): Required (first column)
thkT (positive value): Optional, unless **z1** needs to be inferred from **z0** and **thkT**
z0 (value): Required unless it can be inferred from **z1** of the next lower dielectric
z1 (value): Required (except for the top layer) unless it can be inferred from **z0** and **thkT**.

In the following example that defines planar-dielectric layers, LINER, PASS1, and PASS2 are the planar components of conformal layers. The conformal components are defined in a dielectric stack shown in “Conformal Dielectric Layers” on page 11-28.

```
qtfDielectricStack
   name   eps z0   z1
   air    1   4    ---
   PASS4b 4.2 3    4
   PASS4a 3.9 2.5  3
   ILD3   4.2 1.9  2.5
   IMD3   4.2 1.6  1.9
   ILD2   3.9 1.3  1.7
   IMD2   4.2 1    1.3
   ILD1   3.9 0.7  1
   IMD1   4.2 0.5  0.7
   ILD    3.9 0.27 0.5
   LINER  6.5 0.26 0.27
   FOX    4.2 0    0.26
qtfEndDielectricStack
```

The ground plane is at 0, the bottom of FOX. The background dielectric as defined here is the same as if it were not defined as a dielectric constant of 1 above PASS2.

Conformal Dielectric Layers

A *conformal dielectric layer*, here, refers to the conformal component of a dielectric, and does not include the planar component. Conformal dielectrics must be defined from low precedence to high. When one conformal dielectric is over another, the outer conformal dielectric must be defined first. Expansion properties (**down**, **out**, **up**) of a conformal layer needs to include expansion values of underlying layers. For example, a 2 μm conformal layer on top of a 1 μm layer should have **out** and **up** values of **3**.

Conformal dielectric layers can include the following properties:

above (layer reference): Can be used instead of **z0**
below (layer reference): Can be used instead of **z1**
desc (string): Optional
down (value): Either **down** or **z0** must be specified, but not both
eps (positive value): Required
layer (layer name): Required
marker (string): Optional
name (string): Required (first column)
out (value): Optional
up (value): Either **up** or **z1** must be specified, but not both
z0 (value): Either **z0** or **down** must be specified, but not both
z1 (value): Except for the top (first) dielectric, either **z1** or **up** must be specified, but not both

The following example that defines conformal-dielectric layers includes gate oxide (GOX) as a layer that is just over the DIFF region. Note that GOX, although it might appear to be a planar dielectric, is actually just conformal on the top of DIFF. For overlapping conformal dielectrics, the second one has precedence. In this stack, PASS4a_M4 (1 μm thick) must be after PASS4b_M4 (2 μm thick), DMG3a (0.01 μm thick) must be after DMG3b (0.02 μm thick), DMG2a (0.01 μm thick) must be after DMG2b (0.02 μm thick), and SPACER (0.1 μm thick) must be after LINER_PO (0. 2μm thick). The **up**, **down**, and **out** values are relative to the layer geometry, and are not relative to any underlying dielectric.

```
qtfDielectricStack
   name      eps z0   z1   layer up   down out
   PASS4a_M4 4.2 4    ---  M4    3    ---  3
   PASS4b_M4 3.9 3    ---  M4    1    ---  1
   DMG3b     4.2 ---  1.9  M3    ---  0.03 0.03
   DMG3a     3.9 ---  1.9  M3    ---  0.01 0.01
   DMG2b     4.2 ---  1.3  M2    ---  0.03 0.03
   DMG2a     3.9 ---  1.3  M2    ---  0.01 0.01
   LINER_PO  6.5 0.27 ---  POLY  0.1  ---  0.3
```

```
   SPACER    5.5 0.27 ---  POLY  0    ---  0.1
   GOX       3.9 0.25 0.26 DIFF  ---  ---  ---
qtfEndDielectricStack
```

All conformal components have the required data: **name**, **eps**, **z0** or **up**, **z1** or **down**, and **layer**.

The QTF language is intended to define a single stack. When two distinct stacks exist, for example the stack over diffusion and the stack over field oxide, distinct sections of one stack must be defined using conformal properties.

Stack Variation

Stack variation is defined in a **qtfAdjustDepthStack** section through *adjust-depth* layers.

qtfAdjustDepthStack
  stackHeader
  adjustDepthLayer(s)
qtfEndAdjustDepthStack

A stack variation is defined by a nominal layer (an *adjust-depth* range from **z0** to **z1**), an associated conductor layer that affects stack variation (does not necessarily overlap the adjust-depth range), and a thickness table that specifies how the stack varies without local parameters of the layer.

The global nature of an **adjustDepth** layer requires a smooth variation (versus position) for reasonable physical models. When the table referenced by the **thkT** property has a local property as an argument (**Sdr**, **Ssi**, **Wdr**, and **Wsi**), it is effectively replaced by a density-based value (**SdrDty**, **SsiDty**, **WdrAvt**, and **WsiAvg**). A 1D indexed table ***table1D*** is treated as a weighted average ***table1D*[avg]**, described on page 11-42. The densities used for the density-based average are calculated on the same grid used for other density-related properties: **SdrDty**, **SsiDty**, **WdrAvg**, and **WsiAvg**.

Adjust-depth layers must include all of the following parameters:

above (layer reference): Can be used instead of **z0**
below (layer reference): Can be used instead of **z1**
desc (string): Optional
layer (layer name): Required
name (string): Required (first column)
thkT (positive table reference): Required
z0 (value): Required
z1 (value): Required

The following example that defines adjust-depth layers uses the same table for both layers. The nominal thickness of both layers is the same, 0.3 μm. A **thkB** value of 0.3 corresponds to no change in stack thickness. A value of 0.4 increases the layer thickness to 0.4 μm, raising higher layers by 0.1 μm.

```
qtfAdjustDepthStack
   name    z0  z1  thkT layer
   ILD3_M3 1.3 1.6 thkB M3
   ILD2_M2 0.7 1   thkB M2
qtfEndAdjustDepthStack
```

Though both layers use the same table, the table references density, spacing or width of M3 for the top layer, and of M2 for the lower layer. The nominal bottom and top (**z0** and **z1**) of an adjust-depth layer can be anywhere in the stack (independent of the **layer** value), but must not overlap the range of any other adjust-depth layer.

Cdp Data

Cdp data describes the amount of device capacitance that exists outside the device and allows QuickCap to remove device model capacitance from the results. In the QuickCap input deck, a **Cdp** declaration specifies a virtual element which exists between two nets and specifies the amount of capacitance of the device model.

A line of QTF Cdp data consists of a primary layer (**layer**), an interaction (recognition) layer (**marker**), values (**fF/um** and **fF/um^2**), and any number of secondary layers (**attach**). QTF Cdp data is equivalent to the **CdpPerLength** and **CdpPerArea** properties of an interconnect layer of gds2cap. A Cdp value can be specified per length (**fF/um**), per area (**fF/um^2)**, or both. For information about how Cdp elements are generated from the Cdp data, see the descriptions of **“fF/um”** and **“fF/um^2”**.

QTF Cdp data is defined in a table that begins with **qtfCdpData** and ends with **qtfEndCdp**. Similar to a conductor, dielectric, or adjust-depth stack, the first line of the Cdp table describes the columns and each successive line describes an element of the Cdp data. In the following example, all values are specified per length.

```
qtfCdpData
   layer marker fF/um attach
   gpoly v1_8n  0.141 NOD
   gpoly v1_8p  0.140 POD
   gpoly v2_5n  0.162 NOD
   gpoly v2_5p  0.159 POD
qtfEndCdpData
```

A Cdp table can include the following columns:

attach
Specifies the name of the second interconnect layer. The **attach** column must be the last column. If you specify multiple **attach** layers, the gds2cap tool searches them in the specified order to locate the second net used for each Cdp element. If you do not specify any **attach** layer, the gds2cap tool sets the generated Cdp elements to ground.

fF/um
Specifies the per-length Cdp value, in units of fF per micron. The value of an associated Cdp element is the *interaction length* multiplied by **fF/um**. The interaction length is the length of an edge of a shape in the interconnect layer (**layer**) that is within a shape on the interaction layer (**marker**). A single interconnect shape can have multiple edges within the interaction layer, leading to multiple Cdp elements. For example, the gate of a MOSFET generally has a Cdp element associated with the drain, and another associated with the source. The gds2cap tool searches the **attach** layers to find a shape that touches the interaction edge. Each Cdp data requires **fF/um** or **fF/um^2**, or can have both.

fF/um^2
Specifies the per-area Cdp value, in units of fF per square micron. The value of a Cdp element is the *interaction area* multiplied by **fF/um^2**. The interaction area is the overlap area between a shape in the interconnect layer (**layer**) and a shape on the interaction layer (**marker**)). A single interconnect shape can have multiple interactions, leading to multiple Cdp elements. For example, a single M2 interconnect can include multiple M1/M2 parallel-plate design capacitors, resulting in a Cdp element for each case. The gds2cap tool searches the **attach** layers to find a shape that overlaps the interaction area. Each Cdp data requires **fF/um^2** or **fF/um**, or can have both.

layer
Specifies the name of an interconnect layer. A layer must be defined for each Cdp element and different Cdp lines can reference the same layer.

marker
Specifies the name of an interaction layer. You need not define the **marker** layer elsewhere in the QTF data, though it must be defined somewhere within the gds2cap technology file for gds2cap to incorporate the QTF Cdp information. The interaction layer is required for Cdp elements with any **attach** layers, but is optional for Cdp data without **attach** layers, which is consistent with the gds2cap **CpPerLength** and **CpPerArea** properties.

Stub and Sublayer Data

Stub data is represented between **qtfStubData** and **qtfEndStubData** commands. Sublayer data is represented between the **qtfSublayerData** and **qtfEndSublayerData** commands. The data generates bits of interconnect or resistor layers that generally are attached at the top or the bottom, effectively changing the cross section of the layer without changing the resistance. A sublayer is affected by any **ignoreCoupling** or **groundCoupling** commands that involve the parent layer. A stub layer must be explicitly included in **ignoreCoupling** or **groundCoupling** commands for the same consideration.

A line of QTF stub or sublayer data must include an **attach** layer, **z0**, **z1**, and at least one layer and etch (**layerAndEtch**). A **marker** layer (does not need to be a QTF layer) is optional.

attach (interconnect layer)
Specifies a conductor layer to attach the sublayer (or stub) and to serve as the base layer used to generate the sublayer. When the etch of the first layer is negative, the sublayer is based on the **attach** layer and expanded. Otherwise, the sublayer is based on the first **layerAndEtch** layer, etched. For each successive etch, with a positive etch value etched layer is ANDed with the sublayer region, or with a negative etch the expanded layer is subtracted from the sublayer region.

layerAndEtch (interconnect layer and etch)
Specifies one or more etches, each etch consisting of the name of an interconnect layer and without a zero etch value.

marker (string)
Specifies a marker layer that must touch the conductor layer to generate the sublayer or stub. The marker layer does not need to be a QTF layer. If not specified, all polygons on the **attach** layer generate sublayers.

z0 (value)
Specifies the height of the bottom of the sublayer or stub layer. The **z0** property must be a value, not a table reference or layer name.

z1 (value)
Specifies the height of the top of the sublayer or stub layer. The **z1** property must be a value, not a table reference or layer name.

Flow Data

The QTF language includes flow data, which is defined in a QTF block beginning with **qtfFlowData** and ending with **qtfEndFlowData**. Recognized columns are **layer** (the primary layer), **toLayer** (the last layer, when a layer range is appropriate to the type of model), **z0** and **z1** (height range), and **pct** (estimated level of error associated with the model). Several of the flow models are recognized by qtfx, but not yet implemented in any flow, pending further studies.

The QTF language recognizes the following technology models, indicated by the **layer** type (planar dielectric, conformal dielectric, interconnect, or adjust-depth layer).

Eliminate an adjust-depth layer (**layer** is an adjust-depth layer)
Eliminate one or more adjust-depth layers.

Note: Not fully implemented. Although this technology model can be specified in the flow data, qtfx does not derive this flow model, nor does gds2cap implement this flow model.

Eliminate a conformal layer (**layer** is a conformal layer)
Eliminate a range of conformal layers (**layer** and **toLayer** are conformal layers)
Eliminate one or more conformal dielectrics, compensating by resizing the base layer.

Note: Not fully implemented. Although this technology model can be specified in the flow data, qtfx does not derive this flow model, nor does gds2cap implement this flow model.

Average planar dielectrics between **z0** and **z1** (no layer type)
Replaces the region between **z0** and **z1** with a dielectric average of the planar dielectrics in that range. This is the only flow model in which **z0** and **z1** are specified, rather than **layer**.

Note: Not fully implemented. Although this technology model can be specified in the flow data, qtfx does not derive this flow model, nor does gds2cap implement this flow model.

Eliminate a single planar dielectric (**layer** is a planar dielectric)
Eliminate a range of planar dielectrics (**layer** and **toLayer** are planar dielectrics)
Moves together the top of the planar dielectric that is just below the lowest planar dielectric in the range and the bottom of the planar dielectric that is just above the highest planar dielectric in the range.

Note: Not fully implemented. Although this technology model can be specified in the flow data, qtfx does not derive this flow model, nor does gds2cap implement this flow model.

Eliminate trapezoidal edges (**layer** is an interconnectlayer)
Model edges as vertical, rather than sloped.

Note: Not fully implemented. Although this technology model can be specified in the flow data, qtfx does not derive this flow model, nor does gds2cap implement this flow model.

QTF flow data estimates the accuracy impact of various approximations to the technology model. The first line in a **qtfFlowData** block defines the columns. Each subsequent lines defines a technology model, depending on the layers referenced.

layer All flow models except dielectric average

Specifies the name of the primary layer. The model depends on the type of layer:

--- (not fully implemented)

Average planar dielectrics with height range **z0** to **z1**.

adjustDepth layer (not fully implemented)

Remove an adjustDepth layer.

Conformal dielectric (not fully implemented)

Remove one or more conformal dielectrics. To remove multiple conformal dielectric layers, specify **toLayer** as another conformal dielectric layer over the same base layer.

Planar dielectric

Remove one or more planar dielectrics. To remove multiple planar dielectric layers, specify **toLayer** as another planar dielectric layer over the same base layer.

Interconnect (not fully implemented)

Remove any trap value.

toLayer Remove planar or conformal dielectrics

Specifies the name of a second conformal or planar dielectric layer. When **layer** is a planar dielectric layer, **toLayer** must be a planar dielectric layer, or undefined (**---**). When **layer** is a conformal dielectric layer, **toLayer** must be a conformal dielectric layer over the same base layer, or undefined (**---**).

z0 Dielectric average

Specifies the height at the bottom of a region for dielectric averaging. Neither **layer** nor **toLayer** can be specified, and **z1** *must* be specified.

z1 Dielectric average

Specifies the height at the top of a region for dielectric averaging. Neither **layer** nor **toLayer** can be specified, and **z0** *must* be specified.

pct All flow models

Specifies the estimated error introduced by the flow approximation, in units of percentage (**1** means 1%).

QTF Ignored Data

The QTF language includes a mechanism to flag data to be ignored without editing the QTF data. This mechanism allows you to simplify and preserve the complete characteristics of the body of a QTF file. The QTF translator leaves all data intact unless you specify the **-removeIgnoredData** option. The gds2cap tool removes the data flagged as ignored.

A QTF *ignore-data* section that begins with **qtfIgnoreData** and ends with **qtfEndIgnoreData** describes the data to be ignored. Each command in the block can include a list of layers related to the data to be ignored. Without such a list, all layers are considered relevant to the ignored data. Any layer in the layer list can begin with the keyword **above**, **below**, or **at**, in which case it references all layers above, below, or overlapping the specified layer, respectively.

The ignore-data section can include the following commands:

adjust[**Depth**][**Layers**] [**(removed)**][**:**] [***layerList***]
: Specifies **adjustDepth** layers to be removed from the QTF database. You can reference any **adjustDepth** layer by specifying name of the layer or the base layer. For example, **adjustDepth: above metal1** ignores existing **adjustDepth** layers for which **z0** is at or above **z1** of metal1. When no layers are specified, the command removes all **adjustDepth** layers.

  With the **(removed)** keyword, the ignored data is considered removed and the command is essentially a comment. The qtfx tool, with **-removeIgnoredData**, adds *(removed)* so that the generated QTF data has a record of the ignored data.

cap[**Layers**] [**(removed)**][**:**] ***layerList***
: Specifies layers in a conductor stack for which capacitance extraction is to be avoided. This command removes all associated conformal layers and QTF Cdp data. For example, **cap: below metal1** removes existing conformal layers and Cdp data of the layers on which **z1** is at or below **z0** of metal1. The ***layerList*** can include names of technology-file layers that are not defined in the QTF data. The gds2cap tool flags any related layers as **notQuickcapLayer** and removes any associated capacitance data, including **Cdp** expressions.

  Each layer entry can be of the form ***layer***, **at** ***layer***, **above** ***layer***, or **below** ***layer***. These refer, respectively, to ***layer***, to all layers overlapping (in z) ***layer***, to all layers above ***layer***, or all layers below ***layer***.

  With the **(removed)** keyword, the ignored data is considered removed and the command is essentially a comment. The qtfx tool, with **-removeIgnoredData**, adds *(removed)* so that the generated QTF data has a record of the ignored data.

conductor[**Properties**] [**(removed)**][**:**] ***propertyList*** [**of** ***layerList***]
Specifies properties to be removed from the conductor stack. If no layers are specified, the command removes specified properties from all layers. If any conductor layers are specified, the command removes properties only from the specified layers.

Each layer entry can be of the form ***layer***, **at** ***layer***, **above** ***layer***, or **below** ***layer***. These refer, respectively, to ***layer***, to all layers overlapping (in z) ***layer***, to all layers above ***layer***, or all layers below ***layer***.

With the **(removed)** keyword, the ignored data is considered removed and the command is essentially a comment. The qtfx tool, with **-removeIgnoredData**, adds *(removed)* so that the generated QTF data has a record of the ignored data.

The following conductor-stack properties cannot be specified in the ignore-data block: **z0**, **z1**, **SdrMin**, **SdrMax**, **WdrMin**, **WdrMax**, **AdrMin**, and **AdrMax**, **etch1–etch8**, and **thkR1–thkR8**.

When removing the **etch** property from a layer, qtfx also removes existing **etch1–etch8** properties as well as any **etchL** and **etchW** property, replaces table arguments of etched parameters (**Asi**, **Dsi**, **Lsi**, and **Ssi**) of the layer by names of the corresponding drawn arguments (**Adr**, **Ddr**, **Ldr**, and **Sdr**), and changes existing silicon-based parameters (**rhoSi**, **rshSi**, and **gPerAreaSi**) of the layer to their corresponding drawn-layer based arguments (**rhoDr**, **rshDr**, and **gperAreaDr**).

When removing the **thkR** property from a layer, qtfx removes **thkR1–thkR8** properties.

conformal[**Layers**] [**(removed)**][**:**] [***layerList***]
Specifies conformal-dielectric layers to be removed from the QTF database. You can reference conformal layers by specifying the name of the conformal layer or the base layer. For example, **conformal: at gpoly** removes existing conformal layers involving layers with a nominal height (**z0** to **z1**) overlapping the height of **gpoly**. When no layers are specified, the command removes all conformal layers.

Each layer entry can be of the form ***layer***, **at** ***layer***, **above** ***layer***, or **below** ***layer***. These refer, respectively, to ***layer***, to all layers overlapping (in z) ***layer***, to all layers above ***layer***, or all layers below ***layer***.

With the **(removed)** keyword, the ignored data is considered removed and the command is essentially a comment. The qtfx tool, with **-removeIgnoredData**, adds *(removed)* so that the generated QTF data has a record of the ignored data.

planar[**Layers**] [**(removed)**][**:**] [***layerList***]
Specifies planar-dielectric layers to be removed from the QTF database. When no layers are specified, the command removes all planar layers.

Each layer entry can be of the form ***layer***, **at *layer***, **above *layer***, or **below *layer***. These refer, respectively, to ***layer***, to all layers overlapping (in z) ***layer***, to all layers above ***layer***, or all layers below ***layer***.

With the **(removed)** keyword, the ignored data is considered removed and the command is essentially a comment. The qtfx tool, with **-removeIgnoredData**, adds *(removed)* so that the generated QTF data has a record of the ignored data.

Tables

Many layer properties can be values or tables. A value is a number. A table is 1D (one table argument) or 2D (two table arguments). The table name has no implicit meaning. Rather, the table name allows reference by a property value or by the argument of another table. The application of a table is dictated by how it is referenced. If named as the **rshDr** property of a conductor layer, for example, a table is used to calculate sheet resistance based on drawn geometry.

The first line of a table specifies the name, directional information, table arguments, and options.

qtfTable *tableName*[***dir***][***class(es)***]([***xArg***[,***yArg***]]) [***postProcProps***] [***directionProp***]
body
qtfEndTable

Defines a numeric table.

A table can have an associated direction ***dir***. In this case, a second table can have the same name referencing the perpendicular direction. For information about direction specification, see “Directional Tables” on page 11-38.

A table can be indexed to apply to a specific class or pair of classes. The same table name can be used with different indices to define a set of class-based values. For information about

- class specification, see “Indexed Tables” on page 11-40.
- table arguments ***xArg*** and ***yArg***, see “Table Arguments” on page 11-43.
- postprocessing properties, see “Postprocessing Properties” on page 11-45.

All direction property must be specified after any postprocessing properties. For information about

- direction properties, see “Directional Properties” on page 11-46.
- the body of the numeric table, see “Numeric Tables” on page 11-48.

qtfInverseTable *tableName*[*dir*][*class(es)*]([*xArg*[,*yArg*]]) [*postProc*] [*dir*]
body
qtfEndInverseTable

Defines an *inverse* table, which bases interpolation or extrapolation on the multiplicative inverse of the values in the table body. For example, in an standard table, the midpoint between 3 and 5 is (3+5)/2=4. In an inverse table, the midpoint is 2/(1/3+1/5) = 3.75.

qtfDeriveTable *tableName*(*xArg*[,*yArg*]) [*postProcProps*] [*directionProp*]
expression
[indexBounds([*index*])
value(s)]
qtfEndDeriveTable

Defines a derived table is similar to a numeric table. The command line is similar to that of a numeric table. A derived table, however, cannot be used for directional etches, directional dielectrics, or for class-based data.

For information about

- table arguments ***xArg*** and ***yArg***, see “Table Arguments” on page 11-43.
- postprocessing properties, see “Postprocessing Properties” on page 11-45.

All direction property must be specified after postprocessing properties. For information about

- direction properties, see “Directional Properties” on page 11-46.
- the expression of the derived table, see “Derived Tables” on page 11-50.
- the table bounds, see “Derived-Table Bounds” on page 11-50.

qtfDeriveInverseTable *tableName*[*dir*][*class(es)*]([*xArg*[,*yArg*]]) [*postProc*] [*dir*]
body
qtfEndDeriveInverseTable

Derives an inverse table from the named file and bases interpolation or extrapolation on the multiplicative inverse of the values in the table body, similar to **qtfInverseTable**.

Tables can be defined in any order, and can be before or after the stacks that reference them. A table name that is neither referenced as the argument of another table, nor as the value of a layer property (in a stack) results in an error.

Directional Tables

The **epsXY** and **epsXYscale** tables can be directional. Directional tables are indicated by suffixing the table name with **[x]**, **[y]**, **[*directionLayer*]**, or **[!*directionLayer*]**. Two tables can have the same name if they reference perpendicular directions, such as **etch_M1[x]()** and **etch_M1[y]()**, or **etch_CONT[OD]()** and **etch_CONT[!OD]()**. Directional tables can be defined for a retargeting etch (**etch0**), for any general etches (**etch**, **etch1**, **etch2**, and so on), for the resistance etch (**etchR**), and for effective dielectric values (**epsXY** and **epsXYscale**).

xy Directional Tables

A x or y directional is indicated by a name suffixed by **[x]** or **[y]**. For example, an etch for M1 can be referenced in the conductor stack as *etch_M1*, and defined by tables *etch_M1[x]* and *etch_M1[y]*. These tables can be 0D (a constant), 1D, or 2D. The suffix determines the direction of the etch (perpendicular to the direction of the edge) or affected dielectric, and must be consistent with the reference direction, **referenceDirection** QTF parameter, see page 11-9. If **referenceDirection** is not defined, the direction must be consistent with a reference direction of y.

For etching, if the **referenceDirection** is x (horizontal), the **[x]** etch is the etch of lines perpendicular to the reference direction and **[y]** is the etch of lines parallel to the reference direction. If the reference direction is y (vertical), the **[x]** etch is the etch of lines parallel to the reference direction and **[y]** is the etch of lines perpendicular to the reference direction.

qtfTable *tableName*[x]([***xArg***[,***yArg***]]) ***options***
qtfTable *tableName*[y]([***xArg***[,***yArg***]]) ***options***

An etch, **epsXY** or **epsXYscale**, table can reference **x** or **y** to define the etch direction or direction of the dielectric to be modified. Two tables with the same ***tableName*** must either reference the same direction layer, **[*directionLayer*]** and **[!*directionLayer*]**, or the two Manhattan directions **[*x*]** and **[*y*]**.

Layer-Based Directional Tables

A layer-based direction is either parallel or perpendicular to a named layer (a *direction* layer). The direction layer must be defined in the gds2cap technology file with a **cross**[**ing**][**Layer**] property. For more information, see the *gds2cap User Guide and Technical Reference*.

Evaluation of layer-based direction is best when the direction layer overlaps the locations for which direction is required. For a layer-based etch or conductivity of DCONT, for example, use OD as a direction layer because it contains DCONT, not POLY, which is separate from DCONT.

qtfTable *tableName*[*directionLayer*]([***xArg***[,***yArg***]]) ***options***
qtfTable *tableName*[!*directionLayer*]([***xArg***[,***yArg***]]) ***options***

An etch, **epsXY** or **epsXYscale** table can reference a layer name to define the etch direction or direction of the dielectric to be modified. The direction can be parallel to the direction of the specified layer (format: **[*directionLayer*]**, above), or perpendicular (format: **[!*directionLayer*]**, above). Two tables with the same ***tableName*** must either reference the same direction layer, **[*directionLayer*]** and **[!*directionLayer*]**, or the two Manhattan directions **[*x*]** and **[*y*]**. The direction of ***directionLayer*** is defined by specifying a cross layer, defined elsewhere in the gds2cap technology file. In the following example, the etch direction associated with the first QTF table is towards or away from the gate. The layer statement is within the gds2cap technology file, but it is part of the native gds2cap technology language and not part of the QTF format.

```
qtfTable etch_CONT[OD](Sdr,Wdr)
   ...
qtfTable etch_CONT[OD](Sdr,Wdr)
   ...
layer OD ... cross=POLY
```

Indexed Tables

The QTF language supports 1D and 2D indexed tables for properties that depend on layer classes. The **dZ**, **eps**[**XY**[**scale**], **etchR**, **gPerAreaDr**|**Si**, **rhoDr**|**Si**, **rhsDr**|**Si**, and **thkB**|**T**|**R** properties can be 1D indexed tables. The **etch** and **etch0** properties can be 1D or 2D indexed tables.

qtfTable *tableName*[*iSource*](*arg(s)*) *options* 1D indexed etch table

An etch table can reference an index to indicate the associated layer class. The index is an integer 0 to 32, in square brackets (**[]**). The table can be directional, where ***tableName*** includes a suffix of **[x]**, **[y], [*direction*],** or **[!*direction*]** before the index pair. Layer classes must be defined by other programs that incorporate the QTF database. The gds2cap technology file, for example, can include a layer function **layerClasses()** to name the layer classes associated with a parent layer. For a 1D table, the etch table is selected based on the layer class of the source polygon (etched), and does not depend on the layer class of the nearest polygon, used to find spacing.

In a gds2cap technology file, the first layer defined in the **layerClasses()** layer function corresponds to an index value of 1, the second layer is 2, and so on. When the last layer includes the **unknownClass** property, the index value is 0. When *tableName*[0]() is undefined (involves unknown layer class), gds2cap uses *tableName*[1]().

qtfTable *tableName*[*iSource*][*iSpacing*](*arg(s)*) *options* 2D indexed etch table

An etch table can reference an index pair to indicate a pair of associated layer classes. The index pair consists of two integers 0 to 7, each in square brackets (**[]**). The table can be directional, where ***tableName*** includes a suffix of **[x]**, **[y], [*direction*],** or **[!*direction*]** before the index pair. Layer classes must be defined by other programs that incorporate the QTF database. The gds2cap technology file, for example, can include a layer function **layerClasses()** to name the layer classes associated with a parent layer. For a 2D table, the etch table is selected based on the layer class of the source polygon (etched), and based on the layer class of the nearest polygon, used to find spacing.

When a table with an unknown layer class is not defined, table with known layer classes serves as the default table. The gds2cap tool maps the unknown class (0) to the first layer class (1). When the other class is already 1, gds2cap maps 0 to 2. Table 11-1 lists default tables for three defined layer classes.

Table 11-1:Default Tables Corresponding to Tables With Unknown Layer Class, for Three Layer Classes

Undefined Table	Default Table
tableName[0][0]	*tableName*[1][2]
tableName[0][1]	*tableName*[2][1]
tableName[0][2]	*tableName*[1][2]
tableName[0][3]	*tableName*[1][3]
tableName[1][0]	*tableName*[1][2]
tableName[2][0]	*tableName*[2][1]
tableName[3][0]	*tableName*[3][1]

qtfTable *tableName*[*i*][*j*](*arg(s)*) *options* Undirected indexed etch table

An **eps**, **epsXY** or **epsXYscale** table can reference an index pair to indicate a pair of associated layer classes. The index pair consists of two integers 0 to 7, each in square brackets (**[]**). The table can be directional, where ***tableName*** includes a suffix of **[x]**, **[y], [*direction*],** or **[!*direction*]** before the index pair. Layer classes must be defined by other programs that incorporate the QTF database. The gds2cap technology file, for example, can include a layer function **layerClasses()** to name the layer classes associated with a parent layer. For a 2D table, the table is selected based on the layer class of the source polygon, and based on the layer class of the nearest polygon, used to find spacing.

Table References

A table can be referenced by name from any of the several properties of interconnects and vias (in a conductor stack), from the **thkT** property in an adjust-depth stack, as an argument to a table, or in an **alias** command in a **qtfTable** or **qtfDeriveTable** block. In general, the table reference is simply the name of a table. An indexed table that is not directional (no suffix of **[x]**, **[y], [*direction*],** or **[!*direction*]**) can be referenced in different ways, depending on usage.

table *Simple-table reference*
***table1D*[*index*]**
***table2D*[*index1*][*index2*]**

Any single non-directional table (no suffix of **[x]**, **[y], [*direction*],** or **[!*direction*]**) can be referenced by name. A reference from another table (as an argument) or from an **alias** command in a **qtfTable** or **qtfDeriveTable** block must be in one of these forms.

table1D *1D class-based reference*

Properties related to etch, height, or thickness change, and resistance can reference a set of 1D indexed tables by specifying the name of the table without any index specification. The actual table used when applying a property to a shape depends on the layer class of that shape.

A 1D table referenced by the **thkT** property of a **qtfAdjustDepthStack**, however, is treated as a weighted sum ***table1D*[avg]**. See the following description.

***table1D*[avg]** *Weighted-average reference*
***table1D*[avg2]**
***table1D*[max]**

Density-weighted table averages can be used for all properties that accept a table as a value, except for **eps**, **epsXY**, and **epsXYscale**. A 1D table ***table1D*** referenced by the **thkT** property of a **qtfAdjustDepthStack** without any index specification is treated as ***table1D*[avg]**.

For a table name with a **[avg]** suffix, the average is weighted by the densities. For three classes with tables **t[1]**, **t[2]**, and **t[3]**, and partial densities of D1, D2, and D3, the average is

(D1***t[1](*args*)** + D2***t[2](*args*)** **+** D3***t[3](*args*))** **/** (D1+D2+D3)

For a table name with a **[avg2]** suffix, the average is based on the squares of the densities. For three classes with tables **t[1]**, **t[2]**, and **t[3]**, and partial densities of D1, D2, and D3, the average is

$(D1^2$***t[1](*args*)** + $D2^2$***t[2](*args*)** **+** $D3^2$***t[3](*args*))** **/** $(D1^2+D2^2+D3^2)$

For a table name with a **[max]** suffix, the value is the table with the largest density. In the case where different classes have the same maximum density, the value is the average of the associated tables. Though this function does not vary smoothly with position, values are calculated on the same grid that is used for other density-related properties.

table2D_directed *2D class-based directed table*

A directed 2D table can be referenced by the **etch**, **etch0**, and **etch1**–**etch8** properties. In a set of 2D class-based directed tables, the first index refers to the layer class being etched, and the second index refers to the layer class used to evaluate spacing.

table2D_nondirected *2D class-based non-directed table*

A non-directed 2D table can be referenced by the **eps**, **epsXY**, and **epsXYscale** properties in the interconnect stack. In a set of 2D class-based non-directed tables, the index order does not affect results: table[1][2] is the same as table[2][1], for example.

Table Arguments

A table argument indicates a property of the associated layer used to evaluate the table. The following are the recognized table arguments.

[...]*tableArg*[...]
[...]1/*tableArg*[...]

Prefixing or suffixing an argument with **...** in a QTF table changes the behavior for values above or below the range of values defined in the table for that argument. Prefixing with **...** implements extrapolation for low table-argument values. Suffixing with **...** implements extrapolation for high table-argument values.

Prefixing an argument with **1/** in a QTF table implements an interpolation method based on the inverse of the argument. Index values in the table are still based on the table argument, not the inverse of the table argument. This format can be used with a table-argument keyword, such as **Adr** and **Wsi**, or with a reference to another table, such as **thk_M1**. Inverse-based interpolation requires positive index values.

Inverse-based interpolation provides a physically consistent method for evaluating tables involving resistance that in simple models is related to the inverse of a table argument. The following are a few examples involving inverse values.

- Via resistance (defined using **gPerAreaDr**, **gPerAreaSi**, and **rContact**) is inversely proportional to area, length, and width (**Adr**, **Asi**, **Ldr**, **Lsi**, **Wdr**, and **Wsi**) in simple models.
- Interconnect resistance (**rshDr** and **rshSi**) is inversely proportional to width (**Wdr** and **Wsi**) in simple models.
- Interconnect resistivity (**rhoDr** and **rhoSi**) is inversely proportional to width and thickness (**Wdr**, **Wsi**, **thk**[**C**|**R**], and thickness-table reference).

For example, for inverse-based interpolation **1/Adr**, the interpolated value corresponding to **Adr**=1.5 is the average of values for **Adr**=1 and **Adr**=3, because 2/3 (1/1.5) is halfway between 1 and 1/3.

A[**dr**]
Uses the area of the drawn polygon (before any etch).

Asi
Uses the area of the local (*silicon*, or *etched*) polygon (after etch).

D[**dr**]
Uses the density of the drawn layer (before any etch).

Dsi
Uses the density of the local (*silicon*, or *etched*) layer (after etch).

L[**dr**]
Uses the length of the drawn layer (before any etch). **Ldr** is only valid as a table argument for the following properties: **etchW**, **etchL**, and (for vias only) **gPerAreaDr**, **gPerAreaSi**, **rScale**, **TC1**, and **TC2**. By default, the length is the larger of the two dimensions of a rectangle. QTF allows the length to be defined as the dimension in x, y, parallel to a layer direction, or perpendicular to a layer direction. For more information, see “Directional Properties” on page 11-46.

Lsi
Uses the length of the local (*silicon*, or *etched*) layer (after etch). **Lsi** is only valid as a table argument for the following properties of a via layer that includes an etch: **gPerAreaDr**, **gPerAreaSi**, **rScale**, **TC1**, and **TC2**. By default, the length is the larger of the two dimensions of a rectangle. QTF allows the length to be defined as the dimension in x, y, parallel to a layer direction, or perpendicular to a layer direction. For more information, see “Directional Properties” on page 11-46.

S[**dr**]
Uses the spacing of the drawn layer (before any etch). For adjust-depth calculations, use an effective spacing, based on densities associated with the drawn layer.

SdrDty
Uses an average spacing, based on the density and the density-based average width. **SdrDty** is **WdrAvg***(1/**Ddr**-1).

Ssi
Uses the spacing of the local (*silicon* or *etched*) layer (after etch). For adjust-depth calculations, use an effective spacing, based on densities associated with the local layer.

SsiDty
Uses an average spacing, based on the density and the density-based average width. **SsiDty** is **WsiAvg***(1/**Dsi**-1).

thk[**C**|**R**]
Uses the thickness in a table related to interconnect resistance: **rScale**, **TC1**, or **TC2** (only for lateral conductors), **rshDr**, **rshSi**, **rhoDr**, and **rhoSi**. When the thicknesses differ for capacitance and for resistance, use **thkC** or **thkR**.

W[**dr**]
Uses the width of the drawn layer (before any etch). For adjust-depth calculations, use an effective width, based on densities associated with the drawn layer. By default, the width is the smaller of the two dimensions of a rectangle. QTF allows the width to be defined as the dimension in x, y, parallel to a layer direction, or perpendicular to a layer direction for tables that can accept **Ldr** or **Lsi** as a table argument: **etchW**, **etchL**, and (for vias only) **gPerAreaDr**, **gPerAreaSi**, **rScale**, **TC1**, and **TC2**. For more information, see “Directional Properties” on page 11-46.

WdrAvg
Uses a density-based average width. The average width is calculated by considering the quadratic behavior of density as a function of (small) etch. The average width is twice the etch value at which the density extrapolates to 0 percent.

Wsi
Uses the width of the local (*silicon* or *etched*) layer (after etch). For adjust-depth calculations, use an effective width, based on densities associated with the local layer. By default, the width is the smaller of the two dimensions of a rectangle. QTF allows the width to be defined as the dimension in x, y, parallel to a layer direction, or perpendicular to a layer direction for tables that can accept **Ldr** or **Lsi** as a table argument: **etchW**, **etchL**, and (for vias only) **gPerAreaDr**, **gPerAreaSi**, **rScale**, **TC1**, and **TC2**. For more information, see “Directional Properties” on page 11-46.

WsiAvg
Uses a density-based average width. The average width is calculated by considering the quadratic behavior of density as a function of (small) etch. The average width is twice the etch value at which the density extrapolates to 0 percent.

tableName
Any table argument that is not a recognized index name is interpreted as the name of a table. For example, a table might have index names of **Wdr** (references drawn width) and *polyT* (references a table named *polyT*). A table can be defined before or after a table reference.

Postprocessing Properties

After the argument list, a numeric or derived table can include one or more postprocessing properties. These properties are applied in the order defined by the values in the numeric table or derived from the expression. The same property can be specified multiple times.

min [**=**] ***min*** ***qtfDeriveTable*** *property*
Ensures that no value in the table is smaller than ***min***. This property is only allowed in derived tables. It is equivalent to changing the derived table expression from ***expression*** to **max(*expression*,*min*)**.

max [**=**] ***max*** ***qtfDeriveTable*** *property*
Ensures that no value in the table is larger than ***max***. This property is only allowed in derived tables. It is equivalent to changing the derived table expression from ***expression*** to **min(*expression*,*max*)**.

offset [**=**] ***offset*** ***qtfDeriveTable*** *or* **qtfTable** *property*
Offsets each value by ***offset***. In a derived table, **offset** is equivalent to changing ***expression*** to **(*expression*)**+***offset***.

scale [**=**] ***scale*** ***qtfDeriveTable*** *or* **qtfTable** *property*
Scales each value by ***scale***. In a derived table, **scale i**s equivalent to changing ***expression*** to **(*expression*)***scale***.

Directional Properties

By default, length (**Ldr** or **Lsi**) is the larger of the two dimensions of a rectangle, and width (**Wdr** or **Wsi**) is the smaller dimension. QTF allows the length and width to be defined as the dimensions in x, y, parallel to a layer direction, or perpendicular to a layer direction for tables that can accept **Ldr** or **Lsi** as an index name: **etchW**, **etchL**, and (for vias only) **gPerAreaDr**, **gPerAreaSi**, **rScale**, **TC1**, and **TC2**.

A **qtfTable** command can end with a property that affects how directional arguments are interpretted.

qtfTable ***tableName***[***dir***][***class(es)***]([***xArg***[,***yArg***]]) [***postProcProps***] [***directionProp***]
The direction property specifies the direction of any **Ldr**, **Lsi**, **Wdr**, and **Wsi** table arguments. All direction must be specified after postprocessing properties. The direction property can be any of the following values:

Default (no ***direction*** specified)
When no direction is specified, length (**Ldr** or **Lsi**) is the larger of the two dimensions and width (**Wdr** or **Wsi**) is the smaller.

L|**W in x**|**y**
The length or width is measured in the x or y-direction. **L in x** is equivalent to **W in y**. When you specify without the standard reference direction, the sense of **x** and **y** is swapped. This occurs, for example, when the QTF data specifies **referenceDirection y**, and the gds2cap command line specifies **-qtfReference x**. The **L...** and **W...** properties are only valid for tables that accept **Ldr** or **Lsi** as an argument: **etchW**, **etchL**, and (for vias only) **gPerAreaDr**, **gPerAreaSi**, **rScale**, **TC1**, and **TC2**.

L|W parallel|perpendicular [**to**] ***directionLayer***

The length or width is measured parallel or perpendicular to the direction layer. **L parallel to *directionLayer*** is equivalent to **W perpendicular to *directionLayer***. In the following example, L is perpendicular to OD (which is perpendicular to POLY). The direction layer should have overlapping objects (DCONT is within OD). Specifying **L parallel to POLY** is *not* recommended.

```
qtfTable gPerAreaDr_DCONT[(Ldr,Wdr) L perpendicular to OD
   ...
layer OD ... cross=POLY
```

The **L...** and **W...** properties are only valid for tables that can accept **Ldr** or **Lsi** as an argument: **etchW**, **etchL**, and (for vias only) **gPerAreaDr**, **gPerAreaSi**, **rScale**, **TC1**, and **TC2**.

polygonBased

Specifies that width, length, and area arguments are based on polygon values. Table 11-2 shows the gds2cap polygon-based function equivalent to each **qtfTable** argument. The **polygonBased** option does not affect density arguments **Ddr** and **Dsi**. In the gds2cap technology file (not part of the native QTF language), the measurement of polygon length, width, and spacing can be modified by various layer properties. For example, you can use **polyStoLayer** to name the layer to measure spacing, and **polyWdirection** to specify the direction to measure width.

Table 11-2:gds2cap Polygon Functions Corresponding to qtfTable Arguments for Polygon Based

qtfTable Argument	Corresponding Polygon Function
Adr	**polyA(*drawnLayer*)**
Asi	**polyA()**
Ddr	No effect
Dsi	No effect
Ldr	**polyL(*drawnLayer*)**
Lsi	**polyL()**
Sdr	**polyS(*drawnLayer*)**
Ssi	**polyS()**
Wdr	**polyW(*drawnLayer*)**
Wdr	**polyW()**

Numeric Tables

A 1D or 2D numeric table can be defined through the **qtfTable** declaration. A table name that is neither referenced as the index of another table, nor as the value of a layer property (in a stack) results in an error. An index name can be a recognized index name (**Adr**, **Asi**, **Ddr**, **Dsi**, **Ldr**, **Lsi**, **Sdr**, **Ssi**, **Wdr**, or **Wsi**) or the name of another table.

alias[**:**] ***tableName(s)*** *New **qtfTable** command*

Defines table aliases for the table named in the preceding **qtfTable** line. The **alias** command must be the first command in the **qtfTable** or **qtfDeriveTable** block. Multiple **alias** commands are permitted. By default, qtfx preserves table aliases. The **-tableAlias** output option of qtfx can be used to instantiate aliased tables (eliminating the alias command) or establish table aliases. See the **-tableAlias** description on page 9-7.

Each **alias** command includes a comma-delimited list of table aliases. Each table alias is the complete name, optionally with an argument list in parentheses. When you include the argument list, any argument that is different from the corresponding table argument (**qtfDeriveTable** or **qtfTable**) must be of the same general type: area (**Adr** and **Asi**); density (**Ddr** and **Dsi**); spacing (**Sdr**, **SdrDty**, **Ssi**, and **SsiDty**); width (**Wdr**, **WdrAvg**, **Wsi**, and **WsiAvg**); length (**Ldr** and **Lsi**); thickness (**thk**, **thkC**, and **thkR**); temperature (**T**); or table references (table name).

In the following example, rho_M9b, and rho_M8 are aliased to rho_M9a, but rho_M8 uses average silicon width and a different thickness table.

```
qtfTable rho_M9a(Wsi,thk_M9)
alias: rho_M9b,rho_M8(WsiAvg,thk_M8)
...
qtfEndTable
```

The following example defines tables for classes 1 through 3 that have the same data:

```
qtfDeriveTable rho_M1[1](Wdr,thk_M1[1])
alias: rho_M1[2](Wdr,thk_M1[2])
alias: rho_M1[3](Wdr,thk_M1[3])
```

A 1D table includes two rows: one for the values of the index, and one for the associated data.

qtfTable ***tableName***[***dir***][***class***](***arg***) [***postProcProps***] [***directionProp***]
columnIndexValues
data
qtfEndTable

Values in the table are modified by any postprocessing properties, **offset** and **scale**. For information, see “Postprocessing Properties” on page 11-45. The **offset** and **scale** properties are useful for adjusting data from other sources. For example, the original thickness-table data might specify an incremental change to thickness. Since the QTF thickness tables need to specify total thickness, you can use an **offset** value equal to the nominal layer thickness. If the thickness data is a percentage change in thickness, use an **offset** value of **1** (indicates total relative thickness) and a **scale** factor equal to the nominal layer thickness.

A 2D table has an index row beginning with an asterisk, *****, followed by the values of the index associated with each column, followed by two or more rows, each of which contains the value of the index associated with the row and the associated data for all columns.

qtfTable ***tableName***[***dir***][***class***](***colArg,rowArg***) [***postProcProps***] [***directionProp***]
  **** colArgValues***
  rowArgValue1 rowData1
  . . .
  rowArgValueN rowDataN
qtfEndTable

In the following example of a table, each column corresponds to a constant spacing, and each row corresponds to a constant width.

```
qtfTable width_MC(Sdr,Wdr)
  *    1    2    3    4
   1   1    1.04 1.08 1.1
   2   1.9  1.94 1.96 1.96
   3   2.84 2.90 2.92 2.92
   4   3.84 3.90 3.92 3.92
qtfEndTable
```

The qtfx tool replaces any width table by an equivalent etch table. In addition, any table it reads is reduced by removing columns and rows that introduce less than approximately 0.1% error. The following **etch** table is generated from the preceding **width** table. Because the etch for lines of width 4 μm is the same as for lines of width 3 μm, qtfx eliminates the row for 4 μm wide lines.

```
qtfTable etch_MC(Sdr,Wdr)
  *         1        2        3        4
   1        0        -0.02    -0.04    -0.05
   2        0.05     0.03     0.02     0.02
   3        0.08     0.05     0.04     0.04
qtfEndTable
```

Derived Tables

A 1D or 2D table derived from an expression can be defined using the **qtfDeriveTable** declaration.

qtfDeriveTable ***tableName*(*xArg*[[*xs*]][, *yArg*[[*ys*]]]) [***postProcProps***] [***directionProp***]
  [**alias ...**]
  expression
  [**indexBounds(**[***arg***]**)**
  value(s)]
qtfEndDeriveTable

An index name (***xArg*** and ***yArg***) can be a recognized index name (**Adr**, **Asi**, **Ddr**, **Dsi**, **Ldr**, **Lsi**, **Sdr**, **Ssi**, **Wdr**, or **Wsi**) or the name of another table.

One or more alias commands can immediately follow the **qtfDeriveTable** command:

alias[**:**] ***tableName(s)*** *New **qtfTable** command*

Defines table aliases for the table named in the preceding **qtfTable** line. The **alias** command must be the first command in the **qtfTable** or **qtfDeriveTable** block. Multiple **alias** commands are permitted. By default, qtfx preserves table aliases. The **-tableAlias** output option of qtfx can be used to instantiate aliased tables (eliminating the alias command) or establish table aliases. See the **-tableAlias** description on page 9-7. For more information, see page 11-48.

The expression begins on the first line after the **qtfDeriveTable** command. The expression itself should reference the argument names. The expression is a standard mathematical expression, as recognized by gds2cap. For information, see the *gds2cap User Guide and Technical Reference*. Any postprocessing operations (**min**, **max**, **offset**, and **scale**) are applied to the result of the expression in the order specified. For more information, see “Postprocessing Properties” on page 11-45.

In addition to compiling the expression, qtfx derives a table based on the expression. Enough rows and columns are inserted so that the maximum interpolation error is no more than about 1%. The gds2cap tool uses the numeric table. Other tools using QTF have the option of evaluating the expression.

Derived-Table Bounds

qtfDeriveTable ***tableName*(*xArg*[[*xs*]][, *yArg*[[*ys*]]]) [***postProcProps***] [***directionProp***]
  expression
  [**indexBounds(**[***arg***]**)**
  value(s)]
qtfEndDeriveTable

The table range can be specified by the **indexBounds** property specified after the expression. In the absence of explicit index bounds, the range of the argument values can be specified by ***xs*** and ***ys*** after the table arguments. If not specified, the range is inferred from other tables associated with the same layer that reference the related table arguments. If **Sdr** is referenced by any table on the same layer, density values are clipped to the range $W/(W+S_{max})$ … $W/(W+S_{min})$ if width W is an argument, or to the range $W_{min}/(W_{min}+S_{max})$ … $W_{max}/(W_{max}+S_{min})$ if width is *not* an argument. When an argument is a table reference, the range is the same as the range of the body of that table. If no range can be determined, qtfx reports an error.

indexBounds([*index*]) ***qtfDeriveTable*** *property*

The index bounds begin with the **indexBounds()** keyword on a line after the derived-table expression. It is followed by one or more rows of values.

For a 1D derived table (a single index value), **indexBounds()** must be specified without any index specification. The following line contains two values: the minimum and maximum value of the derived-table argument.

For a 2D derived table (two arguments), **indexBounds()** must specify one of the two index names. Each following lines contains three values: a value for the specified index, and the minimum and maximum values of the other index. The list terminates at the **qtfEndDeriveTable** command.The following example of a derived table defines the valid range of density **Dsi** (5% to 95%). The range for **Wdr** must be inferred from other QTF data.

```
qtfDeriveTable polyT(Wdr,Dsi[0.05,0.95]
  (Wdr<2)? 0.3 + 0.1*(Dsi-0.5) - 0.1*(Wdr-2):
           0.3 + 0.1*(Dsi-0.5)
```

When generating QTF files, qtfx generates a table based on the derived-table expression. The size of the table is expanded to limit the interpolation errors to about 1%, although the body of the table is restricted to no more than 1024 elements. The following is the table that results from the preceding derived-table example (results depend on the inferred valid ranges):

```
qtfTable polyT(Wdr,Dsi)
  *         1        1.98438  2.03125
   0.05     0.355    0.256563 0.255
   0.95     0.445    0.346562 0.345
qtfEndTable
```

The following example uses **indexBounds** to define limits and includes postprocessing operations.

```
qtfDeriveTable thk_M1(Wsi,Ddr) offset=1 min=0.5 max=1.5
[(Wsi<1)? 1-(Wsi/1):log(1/Wsi)]+[(Ddr<50%)? 1-(Ddr/50%):log(50%/Ddr)]
indexBounds(Wsi)
0.1 50% 95%
1.0  5% 95%
```

```
1.9  5% 50%
qtfEndDeriveTable
```

The **min** and **max** properties establishes values bounds (0.5 and 1.5) after applying the offset. Bounds for **Ddr** are provided at the end as a function of the width **Wsi**. Using qtfx to translate this derived table generates the following.

```
qtfTable thk_M1(Wsi,Ddr)
  *         0.1      0.2125   0.325    ... 0.94375  1        1.05625 ... 1.84375  1.9
   0.05     1.5      1.5      1.5      ... 1.5      1.5      1.5     ... 1.2882   1.25815
   0.10625  1.5      1.5      1.5      ... 1.5      1.5      1.5     ... 1.1757   1.14565
   0.1625   1.5      1.5      1.5      ... 1.5      1.5      1.5     ... 1.0632   1.03315
   0.21875  1.5      1.5      1.5      ... 1.5      1.5      1.5     ... 0.950698 0.920646
   0.246875 1.5      1.5      1.5      ... 1.5      1.5      1.45153 ... 0.894448 0.864396
   0.275    1.5      1.5      1.5      ... 1.5      1.45     1.39528 ... 0.838198 0.808146
   0.303125 1.5      1.5      1.5      ... 1.45     1.39375  1.33903 ... 0.781948 0.751896
   0.33125  1.5      1.5      1.5      ... 1.39375  1.3375   1.28278 ... 0.725698 0.695646
   ...      ...      ...      ...      ... ...      ...      ...    ...  ...      ...
   0.6125   1.5      1.5      1.47206  ... 0.853309 0.797059 0.742334... 0.5      0.5
   0.66875  1.5      1.4967   1.3842   ... 0.765448 0.709198 0.654473... 0.5      0.5
   0.696875 1.5      1.4555   1.343    ... 0.724252 0.668002 0.613277... 0.5      0.5
   0.725    1.5      1.41594  1.30344  ... 0.684686 0.628436 0.573712... 0.5      0.5
   0.753125 1.49038  1.37788  1.26538  ... 0.646627 0.590377 0.535652... 0.5      0.5
   0.78125  1.45371  1.34121  1.22871  ... 0.609963 0.553713 0.5     ... 0.5      0.5
   0.8375   1.38419  1.27169  1.15919  ... 0.540437 0.5      0.5     ... 0.5      0.5
   0.865625 1.35116  1.23866  1.12616  ... 0.507406 0.5      0.5     ... 0.5      0.5
   0.89375  1.31918  1.20668  1.09418  ... 0.5      0.5      0.5     ... 0.5      0.5
   0.921875 1.2882   1.1757   1.0632   ... 0.5      0.5      0.5     ... 0.5      0.5
   0.95     1.25815  1.14565  1.03315  ... 0.5      0.5      0.5     ... 0.5      0.5
qtfEndTable
```

A. Environment Variables

This appendix describes the following environment variables that affect licensing.

setenv QUICKCAP_LICENSE_FILE *licenseFile*
Specifies the license file to be used. This environment variable can be used in lieu of the FLEXlm environment variable **LM_LICENSE_FILE** if the license file is different from the license file for other programs.

setenv QUICKCAP_LICENSE_PREF restricted[**[*program*]**]**[**=*count*]**
setenv QUICKCAP_LICENSE_PREF NXonly[**[*program*]**]**
setenv QUICKCAP_LICENSE_PREF NXorMCPU[**[*program*]**]** (Default)
Specifies licensing options. By default, any tool in the auxiliary package first attempts to check out a QUICKCAP_MCPU license, and on failure checks out a QUICKCAP_NX license. **QUICKCAP_LICENSE_PREF** is also used by gds2cap and QuickCap.

restricted[**[*program*]**]**[**=*count*]
When **restricted** is specified with a positive ***count***, counts the number of licenses available to determine which to use. This might take some time depending on how many licenses are defined and on the configuration of the license servers. In previous releases, gds2cap, QuickCap (except for the parent process), or any tool in the auxiliary package counts licenses even when **restricted** is not specified.

When **restricted** is specified with no ***count***, programs that do not require a QUICKCAP_NX license use a QUICKCAP_MCPU license. Specifying **restricted** with a ***count*** of 0 is equivalent to **NXonly**.

NXonly[**[*program*]**]**
Checks out only QUICKCAP_NX license. If you have only QUICKCAP_NX licenses, specifying **NXonly** reduces the effort gds2cap uses to check out a license.

NXorMCPU[**[*program*]**]** (default)
If unable to check out a QUICKCAP_MCPU license, check out a QUICKCAP_NX license.

The **QUICKCAP_LICENSE_PREF** environment variable can contain multiple **restricted**, **NXonly**, and **NXorMCPU** specifications. A specification can be targeted for gds2cap, QuickCap or for a particular auxiliary program such as cap2sigma or gds2tag by including the program name in square brackets immediately after **restricted**, **NXonly**, or **NXorMCPU**. Any targeted restrictions should precede any general restriction (no ***program*** specification).

setenv SNPSLMD_LICENSE_FILE *path* *Licensing*

Specifies the license file or path. If defined, **SNPSLMD_LICENSE_FILE** takes precedence over **QUICKCAP_LICENSE_FILE** and **QUICKCAP_LICENSE_PATH**. Users upgrading from an earlier version might require a new license file from Synopsys.