

Sentaurus™ Interconnect User Guide

Version N-2017.09, September 2017

SYNOPSYS®

Copyright and Proprietary Information Notice

©2017 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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 <https://www.synopsys.com/company/legal/trademarks-brands.html>. 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

About This Guide	xix
Related Publications	xix
Conventions	xix
Customer Support	xx
Accessing SolvNet.	xx
Contacting Synopsys Support	xx
Contacting Your Local TCAD Support Team Directly.	xxi
Acknowledgments.	xxi
<hr/>	
Chapter 1 Getting Started	1
Overview.	1
Setting Up the Environment	2
Starting Sentaurus Interconnect	2
Starting Different Versions of Sentaurus Interconnect	3
Using a Command File	3
<hr/>	
Chapter 2 Working With the Simulator	5
Overview.	5
Interactive Visualization	6
Interface to Sentaurus Visual.	6
Setting Up the Interface	8
Loading Command Files	8
Inserting Breakpoints in the Flow	8
Deleting Breakpoints in the Flow.	9
Indicating Status of Steps	9
Updating the Structure	10
Controlling the Interface to Sentaurus Visual With the graphics Command	10
Command-Line Options	11
Interactive Mode	12
Fast Mode	12
Terminating Execution	13
Environment Variables	13
File Types	14
Syntax for Creating Command Files.	16
Tcl Input.	16
Specifying Materials	18

Contents

Aliases	19
Default Simulator Settings: SINTERCONNECT.models File	20
Compatibility With Previous Releases	20
Parameter Database	21
Like Materials: Material Parameter Inheritance	23
Interpolation Between Like Materials	23
Interface Parameters	23
Regionwise Parameters and Region Name-Handling	24
Parameter Database Browser: Viewing the Defaults	25
Starting the Parameter Database Browser	26
PDB Browser Functions	27
Viewing Parameter Information	28
PDB Browser Preferences	29
Viewing Parameters Stored in TDR Files	30
Understanding Coordinate Systems	31
Wafer Coordinate System	31
Simulation Coordinate System	31
Creating and Loading Structures and Data	33
Defining the Structure: The line and region Commands	33
Creating the Structure and Initializing Data	34
Defining the Crystal Orientation	35
Automatic Dimension Control	36
Interpolating Field Data From an External Structure File Using the load Command	37
Loading 1D Profiles: The profile Command	38
Saving Structures	38
Saving a Structure for Restarting Simulations	39
Saving a Structure for Device Simulation	40
Saving 1D Profiles for Inspect	41
Saving 1D TDR Files From 2D and 3D Simulations, and 2D TDR Files From 3D Simulations	42
The select Command (More 1D Saving Options)	42
The datexcodes.txt File	43
References	43

Chapter 3 Computing Mechanical Stress	45
Overview	45
Material Models	46
Viscoelastic Materials	47
Maxwell Model	47
Standard Linear Solid Model	48
Purely Viscous Materials	50

Shear Stress–Dependent Viscosity	50
Purely Elastic Materials	51
Anisotropic Elastic Materials	51
Cubic Crystal Anisotropy	52
Hexagonal Crystal Anisotropy	52
Orthotropic Model	53
Plastic Materials	55
Incremental Plasticity	55
Deformation Plasticity	58
Viscoplastic Materials	59
Anand Model	59
Power Law Creep	62
Swelling	65
Temperature-Dependent Mechanical Properties	66
Plane Stress Analysis	67
Equations: Global Equilibrium Condition	68
Second-Order Finite-Element Analysis	69
Mechanics Simulations on Mixed or Hybrid Meshes	70
Mechanics Simulations With Composite Solid Shell Element	71
Boundary Conditions	73
Example: Applying Boundary Conditions	75
Deprecated Syntax	75
Variable Boundary Condition	76
Pressure Boundary Condition	77
Advanced Dirichlet Boundary Condition	77
Advanced Neumann Boundary Condition	78
Point-Force Boundary Condition	78
Point-Displacement Rate Boundary Condition	79
Periodic Boundary Condition	80
Coupling Boundary Condition	81
Time-Step Control	82
Stress-Causing Mechanisms	82
Densification	82
Selectively Switching Off Grid Movement	83
Grain Growth	84
Thermal Mismatch	84
Edge Dislocation	85
Inverse Piezoelectric Effect	88
Effective Surface Tension and Pressure due to Fluid	89
Intrinsic Stress	90
Stress Rebalancing After Etching and Deposition	91

Contents

Automated Tracing of Stress History	91
Saving Stress and Strain Components	92
Description of Output Fields	92
Tracking Maximum Stresses	98
Principal Stresses	99
Principal Strains	100
Nodal Stress and Strain at Like-Material Interface	101
Submodeling	101
Crack Analysis	103
V-Notch Cracks	105
Bulk Cracks	105
Cohesive Zone Modeling	106
Exponential Law	106
Triangular Law	107
Damage Variable	109
References	111
<hr/>	
Chapter 4 Mechanics Postprocessing	113
Stress-Induced Mobility Enhancement	113
Displacement Field	114
J-Integral	115
C(t)-Integral	118
Jv-Integral	118
References	119
<hr/>	
Chapter 5 Thermal Analysis	121
Overview	121
Thermal Model	121
Boundary Conditions	123
Bottom Boundary Condition	124
Side Boundary Condition	124
Defining Thermal Contacts	126
Applying Boundary Conditions to Thermal Contacts	126
Thermal Submodeling	127
Thermal Resistance Matrix	127
Thermal RC Network	128
Output Data Fields	128
Example: 2D Thermal Analysis With Joule Heating	129
References	130

Chapter 6 Electrical Current Analysis	131
Overview	131
Electrical Current Model	131
Wire Size–Dependent Electrical Conductivity	133
Proximity Field–Based Model	133
Nearest Distance–Based Model	135
Grain Size–Dependent Electrical Conductivity	135
Boundary Conditions	136
Defining Electrical Contacts	137
Applying Boundary Conditions to Electrical Contacts	137
Resistance Matrix	138
Output Data Fields	139
Example: 2D Electrical Current Analysis	139
References	142

Chapter 7 Electrostatic Analysis	143
Overview	143
Electrostatic Model	143
Boundary Conditions	145
Defining Electrodes	145
Applying Boundary Conditions to Electrodes	145
Capacitance Matrix	146
Output Data Fields	146
Example: Electrostatic Analysis of Strip Between Two Ground Planes	146
References	148

Chapter 8 Mixed-Mode Analysis	149
Overview	149
Compact Models	150
Example: Compact Models	152
SPICE Circuit Files	154
System Section	155
Circuit Devices	156
Temperature Dependency	157
SPICE Circuit Models	158
SPICE Circuit Parameters	158
User-Defined Circuit Models	159
Example: Mixed-Mode Electrical Current Analysis	159

Contents

Chapter 9 Complementary Models	163
Grain Growth Model	163
Initializing the Grain Size	165
Output Data Fields	165
References	166
Chapter 10 Advanced Calibration	167
Overview	167
Using Advanced Calibration	167
Additional Calibration by Users	168
Chapter 11 Mesh Generation	169
Overview	169
Mesh Refinement	170
Viewing Mesh Refinement	171
Static Mesh Refinement	171
Standard Refinement Boxes	171
Interface Axis-Aligned Refinement Boxes	172
Interface Offsetting Refinement Boxes	173
Refinement Inside a Mask	173
Refinement Near Mask Edges or Mark Corners	174
Uniform Mesh Scaling	176
Adaptive Mesh Refinement	177
Adaptive Refinement Criteria	177
Relative Difference Criteria	178
Absolute Difference Criteria	179
Logarithmic Difference Criteria	179
Inverse Hyperbolic Sine (asinh) Difference Criteria	180
Gradient Criteria	180
Local Dose Error Criteria	180
Interval Refinement Criteria	182
Summary of Refinement Parameters	183
Localizing Adaptive Meshing Using refinebox Command	183
Examples	184
Adaptive Meshing During Analysis	184
Tips for Adaptive Meshing	186
Manipulating Refinement Boxes: transform.refinement Command	187
Settings for Sentaurus Mesh	188
Displaying Refinement Boxes	192

Controlling the Mesh During Moving-Boundary Problems	194
TSUPREM-4 Moving-Boundary Meshing Library	194
Control Parameters in TS4 Mesh Library	194
Moving Mesh and Mechanics Displacements	196
Controlling the Grid Spacing	196
Cleaning Up the Grid	196
Miscellaneous Tricks	197
Meshing for 3D Moving-Boundary Problems	197
MovingMesh Algorithm	198
UseLines: Keeping User-Defined Mesh Lines	201
Using line Commands After init Command	201
Dimension Within Current Spatial Dimension	202
Dimension Greater Than Current Spatial Dimension	202
Using line Commands to Create Virtual Spacing	203
UseLines and the transform Command	203
Reflection	203
Stretch	203
Rotation	204
Translation	204
Cut	204
Example: Testing line Commands	204
Example: Showing Clearing Lines for a New Structure	205
Data Interpolation	205
Data Interpolation Near Boundaries	206
Troubleshooting	207

Chapter 12 Structure Generation	209
Overview	209
Etching	210
Isotropic Etching	213
Anisotropic and Directional Etching	214
Polygonal Etching and CMP	218
Fourier Etching	219
The shadowing and shadowing.nonisotropic Options	221
Crystallographic Etching	222
Trapezoidal Etching	222
Two-Dimensional Trapezoidal Etching	223
Three-Dimensional Trapezoidal Etching	225
Trapezoidal Etching Using force.full.levelset Option	226
Piecewise Linear Etching	227
Etching Beams	228

Contents

Etching Tips	228
Deposition	229
Isotropic Deposition	232
Polygonal Deposition and Fill	232
Crystallographic Deposition	233
Fourier Deposition	233
Trapezoidal Deposition	234
Selective Deposition	235
Fields in Deposited Layers	236
Handling Stress in Etching and Deposition	236
Suppressing Mesh Generation in 3D Simulations	237
Shape Library	238
PolyHedronSTI	238
PolyHedronSTIacc	241
PolyHedronSTIaccv	241
PolyHedronCylinder	242
PolyHedronEllipticCylinder	243
PolygonSTI	244
PolygonWaferMask	245
PolyHedronEpiDiamond	245
Tailored Structure Generation	246
Pillar Structure With Hexahedral Mesh	246
Examples of Generating Different Pillar Structures	247
Multiple-Pillar Structure With Hexahedral Mesh	249
Masks and Lithographic Patterning	251
Photoresist Masks	254
Boolean Masks	254
Line Edge Roughness Effect	256
Mirror Boundary Conditions	258
Geometric Transformations	259
Refinement Handling During Transformation	259
Contact Handling During Transformation	260
Reflection	260
Refinement Handling During Reflection	261
Stretch	261
Refinement Handling During Stretch	262
Cut	262
Refinement Handling During Cut	263
Flip and Backside Processing	263
Refinement Handling During Flip	263
Rotation	264

Refinement Handling During Rotation	264
Translation	264
MGOALS	264
MGOALS Boundary-Moving Algorithms.	265
MGOALS Boundary-Moving Parameters	266
MGOALS 3D Boundary-Moving Algorithms.	269
Summary of MGOALS Etching	270
MGOALS Backward Compatibility	271
Boundary Repair Algorithm	271
Structure Assembly in MGOALS Mode	272
Multithreading	273
Insertion: Internal Mode	273
Inserting Segments.	273
Inserting Polygons	274
Inserting Polyhedra	274
Reading Polyhedra From TDR Boundary File.	275
Creating a Cuboid (Brick) Polyhedron.	275
Extruding 2D Polygons	275
Creating Polyhedron From Its Constituent Polygonal Faces	275
Insertion: External Mode	276
Inserting Polyhedra	276
Sentaurus Structure Editor Interface	277
Sentaurus Topography Interface	280
Sentaurus Topography 3D Interface	282
References.	283

Chapter 13 ICWBEV Plus Interface for Layout-Driven Simulations	285
Overview.	285
Introducing ICWBEV Plus for TCAD Users	286
Opening GDSII Layout Files.	286
User Interface of ICWBEV Plus	287
Sentaurus Markups	288
Stretch Utility.	290
Renaming Markups	292
Auxiliary Layers	293
Text Labels	294
Editing Polygons	295
Resizing a Rectangle	295
Converting a Rectangle to a Polygon.	295
Nonaxis-Aligned Simulation Domains	296
Sentaurus Markup Files and TCAD Layout Files.	297

Contents

Saving the Sentaurus Markup File	298
Contents of Sentaurus Markup File	299
Reloading the Markup File	300
Saving the TCAD Layout File	301
Contents of TCAD Layout File	302
Reloading the TCAD Layout File	303
Starting ICWBEV Plus in Batch Mode and Using Macros	304
ICWBEV Plus Macros	304
Tcl-Based Macros for Layout Parameterization	304
TCAD Layout Reader	305
Loading the TCAD Layout	306
Finding Simulation Domains	306
Finding Layer Names and Layer IDs	306
Selecting the Simulation Domain	307
Loading a GDSII Layout	307
Finding Domain Dimensions	308
Finding Bounding Box of Domain	308
Interface With line Commands	309
Creating Masks	309
Mask-Driven Meshing	312
Layout-Driven Contact Assignment	313
Aligning Wafer and Simulation Domain	316
Additional Query Functions	316
<hr/>	
Chapter 14 Extracting Results	319
Overview	319
Saving Data Fields	319
Selecting Fields for Viewing or Analysis	320
Obtaining 1D Data Cuts	320
Examples	321
Determining the Dose: Layers	323
Extracting Values and Level Crossings: interpolate	323
Extracting Values During solve Step: extract	324
Optimizing Parameters Automatically	325
Fitting Routines	326
FitArrhenius Command	326
FitLine Command	326
Storing Time Versus Resistance, Current, and Voltage	327

Chapter 15 Numerics	329
Overview	329
Setting Parameters of the Iterative Solver ILS	330
Partitioning and Parallel Matrix Assembly	332
Matrix Size Manipulation	334
Node and Equation Ordering	334
Time Integration	335
Time-Step Control	336
Time-Step Control for PDEs	336
Error Control for PDEs	338
Time-Step Control for Mechanics	338
Convergence Criteria	339
Time-Step Adjustment	340
Time-Step Cutback	341
References	342
Chapter 16 Writing Partial Differential Equations Using Alagator	343
Available Operators and Variables	343
Binary and Unary Operators	343
Simple Functions	345
Differential Functions	345
String Names	346
Solution Names and Subexpressions	346
Constants and Parameters	346
Basics of Specifying Partial Differential Equations	347
Building Partial Differential Equations	348
Setting the Boundary Conditions	349
Dirichlet Boundary Condition	349
Segregation Boundary Condition	350
Using Terms	350
Modifying Built-in Equations and Terms	351
UserAddEqnTerm and UserSubEqnTerm	351
UserAddToTerm and UserSubFromTerm	352
Using Callback Procedures to Build Models	353
Callbacks During Execution of the solve Command	353
The solvePreProcess Procedure	354
Building and Solving PDEs	355
The solvePostProcess Procedure	355
Specifying Callback Procedures Using Keywords	356
Common Features of Using Keywords	356

Contents

Cleaning Up Equation Strings: InitProc	357
Constructing Equation Strings: EquationProc	358
Summary	358
References	359

Appendix A Commands	361
Syntax Conventions	361
Example of Command Syntax	362
Common Arguments	363
Quantities and Units for Command Arguments	364
alias	368
ArrBreak	369
Arrhenius	370
beam	371
bound	373
boundary	374
circuit	377
Compatibility	380
contact	381
contour	387
crack	389
current_ramp	393
CutLine2D	395
CZM	396
define	398
defineproc	399
DeleteRefinementboxes	400
deposit	401
doping	408
element	410
Enu2G	411
Enu2K	412
equation	413
etch	414
exit	421
extract	422
fbreak	424
fcontinue	424
fexec	425
fproc	425
fset	425

generic_step	426
graphics	428
grid	432
help	444
icwb	445
icwb.composite	449
icwb.contact.mask	451
icwb.create.all.masks	454
icwb.create.mask	455
init	458
insert	464
integrate	467
interface	470
interpolate	473
j_integral	475
KG2E	477
KG2nu	478
layers	479
line	481
line_edge_roughness	485
load	487
LogFile	491
mask	493
mater	497
math	501
mgoals	510
mobility	516
mode	518
optimize	522
option	526
paste	531
pdbDelayDouble	533
pdbdiff	534
pdbExprDouble	535
pdbGet and Related Commands	536
pdbIsAvailable	538
pdbLike	539
pdbSet and Related Commands	540
pdbUnSet-Related Commands	543
photo	544
plot.1d	546

Contents

plot.2d	549
plot.xy	553
point	555
point.xy	557
polygon	559
polyhedron	563
power_ramp	567
print.1d	570
print.data	572
printCapacitanceMatrix	573
printRC	575
printResistanceMatrix	577
printThermalRC	579
printThermalResistanceMatrix	581
profile	583
RangeRefineboxes	588
refinebox	591
region	601
resistance	605
sde	606
select	609
Set3DDeviceMeshMode	613
Set3DMovingMeshMode	615
SetFastMode	616
SetMechanicsMeshMode	617
SetPerformanceMode	618
SetPlxList	619
SetTDRList	620
SetTemp	621
SetTS4MechanicsMode	622
slice	623
solution	626
solve	629
sptopo	632
stdiff	633
stressdata	634
strip	642
struct	643
supply	647
System	651
tclsel	653

temp_ramp	655
term	658
topo	660
transform	661
transform.mask	667
transform.refinement	671
translate	676
update_hoop_radial_stress	677
update_principal_strain	678
update_principal_stress	678
voltage_ramp	679
WritePlx	681

Appendix B Resistance, Capacitance, and Thermal Resistance **683**

Overview	683
Electrical Current Analysis: Resistance Matrix	683
Electrostatic Analysis: Capacitance Matrix	685
Floating Conductors	687
Merging Contacts	687
Distributed RC Extraction	688
Thermal Analysis: Thermal Resistance Matrix	691
Thermal Analysis: Thermal RC Network	693
Example: Electrostatic Analysis With Floating Conductor	695
References	697

Contents

About This Guide

The Synopsys Sentaurus™ Interconnect tool is an advanced 1D, 2D, and 3D simulator suitable for IC interconnect reliability analysis. It features modern software architecture and state-of-the-art models to address current and future interconnect technologies, and is capable of mechanical, thermal, and electrical analysis.

Sentaurus Interconnect simulates process simulation steps, etching, and deposition along with mechanical, thermal, and electrical analyses in 1D, 2D, and 3D. Three-dimensional capabilities include meshing of boundary files using the MGOALS module, and an interface to Sentaurus Structure Editor, which is the 3D geometry editing tool based on the ACIS solid modeling library.

Sentaurus Interconnect uses the Alagator scripting language that allows users to implement and solve their own nonmechanics partial differential equations. Alagator can be used to solve various electrical and thermal analysis equations. Three-dimensional simulations are handled in exactly the same way as for one dimension and two dimensions. Therefore, all of the advanced models and user programmability available in one dimension and two dimensions can be used in three dimensions.

Related Publications

For additional information, see:

- The TCAD Sentaurus release notes, available on the Synopsys SolvNet® support site (see [Accessing SolvNet on page xx](#)).
- Documentation available on SolvNet at <https://solvnet.synopsys.com/DocsOnWeb>.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Blue text	Identifies a cross-reference (only on the screen).
Bold text	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field or an option.
Courier font	Identifies text that is displayed on the screen or that the user must type. It identifies the names of files, directories, paths, parameters, keywords, and variables.

Convention	Description
<i>Italicized text</i>	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.
Menu > Command	Indicates a menu command, for example, File > New (from the File menu, select New).

Customer Support

Customer support is available through the Synopsys SolvNet customer support website and by contacting the Synopsys support center.

Accessing SolvNet

The SolvNet support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNet site:

1. Go to the web page at <https://solvnet.synopsys.com>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

If you need help using the site, click **Help** on the menu bar.

Contacting Synopsys Support

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

- Go to the Synopsys [Global Support Centers](#) site on [synopsys.com](https://www.synopsys.com). There you can find e-mail addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNet site or the Synopsys Global Support Centers site and [open a case online](#) (Synopsys user name and password required).

Contacting Your Local TCAD Support Team Directly

Send an e-mail message to:

- support-tcad-us@synopsys.com from within North America and South America.
- support-tcad-eu@synopsys.com from within Europe.
- support-tcad-ap@synopsys.com from within Asia Pacific (China, Taiwan, Singapore, Malaysia, India, Australia).
- support-tcad-kr@synopsys.com from Korea.
- support-tcad-jp@synopsys.com from Japan.

Acknowledgments

Sentaurus Interconnect is based on the 2000 and 2002 releases of FLOOPS written by Professor Mark Law and coworkers at the University of Florida. Synopsys acknowledges the contribution of Professor Law and his advice in the development of Sentaurus Interconnect. Go to <http://swamp.mse.ufl.edu/> for more information about TCAD at the University of Florida.

About This Guide
Acknowledgments

CHAPTER 1 Getting Started

This chapter provides an overview of the functionality of Sentaurus Interconnect and describes how to start the tool.

The TCAD Sentaurus Tutorial provides a range of projects to assist users to work with the tool. To access the tutorial, go to:

```
`${STROOT}/tcad/`${STRELEASE}/Sentaurus_Training/index.html
```

where `STROOT` is an environment variable that indicates where the Synopsys TCAD distribution has been installed, and `STRELEASE` indicates the Synopsys TCAD release number.

Overview

Sentaurus Interconnect is a complete and highly flexible, multidimensional, IC interconnect reliability analysis environment. With its modern software architecture and extensive breadth of capabilities, Sentaurus Interconnect is a state-of-the-art simulation tool. It offers unique predictive capabilities for modern silicon and nonsilicon technologies such as analyzing the reliability of semiconductor interconnect structures. In particular, Sentaurus Interconnect identifies hot spots in interconnect structures that are susceptible to void formation, de-bonding, and cracking due to physical phenomena such as stress and temperature excursions. These reliability concerns arise from both the manufacturing process and circuit operation.

Sentaurus Interconnect accepts as input a sequence of commands that is either entered from standard input (that is, at the command prompt) or composed in a command file. Simulations are performed by issuing a sequence of commands that corresponds to the individual process steps or analysis modes. In addition, several commands allow you to select physical models and parameters, grid strategies, and graphical output preferences, if required. You should place parameter settings in a separate file, which is sourced at the beginning of command files using the `source` command.

In addition, the Alagator scripting language allows you to describe and implement your own models, and thermal and electrical analysis equations.

Setting Up the Environment

The `STROOT` environment variable is the TCAD Sentaurus root directory, and you must set this variable to the installation directory of TCAD Sentaurus. The `STRELEASE` environment variable can be used to specify the release of the software to run, for example, N-2017.09. If `STRELEASE` is not set, the default version is used which is usually the last version installed.

To set the environment variables:

1. Set the TCAD Sentaurus root directory environment variable `STROOT` to the TCAD Sentaurus installation directory, for example:

```
* Add to .cshrc
setenv STROOT <Sentaurus directory>

* Add to .profile, .kshrc, or .bashrc
STROOT=<Sentaurus directory>; export STROOT
```

2. Add the `<Sentaurus directory>/bin` directory to the user path.

For example:

```
* Add to .cshrc:
set path=(<Sentaurus directory>/bin $path)

* Add to .profile, .kshrc, or .bashrc:
PATH=<Sentaurus directory>/bin:$PATH
export PATH
```

Starting Sentaurus Interconnect

You can run Sentaurus Interconnect in either interactive mode or batch mode. In interactive mode, a whole process flow can be simulated by entering commands line-by-line as standard input. To start Sentaurus Interconnect in interactive mode, enter the following on the command line:

```
> sinterconnect
```

Sentaurus Interconnect displays version and host information, followed by the Sentaurus Interconnect command prompt. You now can enter Sentaurus Interconnect commands at the prompt:

```
sinterconnect>
```


This is a flexible way of working with Sentaurus Interconnect to test individual process steps or short sequences, but it is inconvenient for long process flows. It is more useful to compile the command sequence in a command file, which can be run in batch mode or inside Sentaurus Workbench.

To run Sentaurus Interconnect in batch mode, load a command file when starting Sentaurus Interconnect, for example:

```
> sinterconnect input.cmd
```

Starting Different Versions of Sentaurus Interconnect

To list all available releases in the TCAD Sentaurus installation directory, enter:

```
> sinterconnect -releases
```

To list all versions in a particular release directory, enter:

```
> sinterconnect -versions
```

You can select a specific release and version number of Sentaurus Interconnect using the `-rel` and `-ver` options as follows:

```
> sinterconnect -rel <rel_number> -ver <version_number>
```

For example:

```
> sinterconnect -rel M-2016.12
```

The following command starts the simulation of `nmos_sis.cmd` using version 1.2 of release M-2016.12 as long as this version is installed:

```
> sinterconnect -rel M-2016.12 -ver 1.2 nmos_sis.cmd
```

Using a Command File

Instead of entering Sentaurus Interconnect commands line-by-line, the required sequence of commands can be saved to a command file, which can be written entirely by users. To save time and reduce syntax errors, you can copy and edit examples of command files available from the TCAD Sentaurus Tutorial or in this user guide.

If a command file has been prepared, run Sentaurus Interconnect by typing the following command:

```
sinterconnect <command_filename>
```

1: Getting Started

Using a Command File

Alternatively, you can automatically start Sentauros Interconnect through the Scheduler in Sentauros Workbench. The command file has the extension `.cmd`. (This is the convention used in Sentauros Workbench.)

The command file is checked for correct syntax and then commands are executed in sequence until the simulation is stopped by the command `exit` or the end of the file is reached. Since Sentauros Interconnect is written as an extension of the tool command language (Tcl), all Tcl commands and functionalities (such as loops, control structures, creating and evaluating variables) are available in command files. This results in some limitations in syntax control if the command file contains complicated Tcl commands. You can switch off syntax-checking with the command-line option `-n`, for example:

```
sinterconnect -n commandfile
```

Sentauros Interconnect ignores character strings starting with a hash (`#`) character (although Sentauros Workbench interprets `#` as a special character for conditional statements). Therefore, this special character can be used to insert comments in the simulation command file.

A file with the extension `.log` is created automatically whenever Sentauros Interconnect is run from a command line, that is, outside the Sentauros Workbench environment. This file contains the runtime output, which is generated by Sentauros Interconnect and is sent to standard output. When Sentauros Interconnect is run using a command file `<root_filename>_sis.cmd`, the output file is named `<root_filename>_sis.log`.

When Sentauros Interconnect is run from Sentauros Workbench, the output file is named `<root_filename>_sis.out`.

For a complete list of commands, see [Appendix A on page 361](#).

CHAPTER 2 Working With the Simulator

This chapter provides an overview of how Sentaurus Interconnect operates.

The syntax and features of the command file are described, followed by an overview of the parameter database, which contains all of the model parameters, and technical details regarding the running of the tool.

For new users, see [Interactive Mode on page 12](#), [Syntax for Creating Command Files on page 16](#), and [Creating and Loading Structures and Data on page 33](#). For advanced users who need to adjust model parameters, see [Parameter Database on page 21](#).

Overview

To familiarize users with the different formatting used in this documentation, input commands from either a command file or the command line are presented this way:

```
sinterconnect -v
```

An example of output is:

```
*****  
***                               Sentaurus Interconnect          ***  
***                               Version N-2017.09              ***  
***                               (1.7, RHEL64)                  ***  
***                                                                 ***  
***                               Copyright (C) 1993-2002        ***  
***                               The board of regents of the University of Florida ***  
***                               Copyright (C) 1994-2017        ***  
***                               Synopsys, Inc.                  ***  
***                                                                 ***  
*** This software and the associated documentation are confidential ***  
*** and proprietary to Synopsys, Inc. Your use or disclosure of this ***  
*** software is subject to the terms and conditions of a written ***  
*** license agreement between you, or your company, and Synopsys, Inc. ***  
*****
```

```
Compiled Fri Jul 14 23:54:02 PDT 2017 on tcadprod8
```

```
Started at: Tue Jul 11 10:25:16 2017 (CEST)  
User name: jbrown
```

2: Working With the Simulator

Interactive Visualization

```
Host name: topo2
PID: 25333
Architecture: x86_64
Operating system: Linux rel. 2.6.18-274.12.1.el5 ver. #1 SMP Tue Nov 8 21:37:35
EST 2011
```

Interactive Visualization

The options for interactive visualization in Sentaurus Interconnect are:

- An X-Windows-based graphical display (which will be phased out in future releases)
 - This viewer can be used for 1D and 2D simulations, and is launched with either the `plot.1d` or `plot.2d` command (see [plot.1d on page 546](#) and [plot.2d on page 549](#)).
- An interface to Sentaurus Visual (which will eventually replace the X-Windows display)

Interface to Sentaurus Visual

The interface to Sentaurus Visual can visualize 1D, 2D, and 3D structures and data evolution while the simulation progresses (see [Figure 1 on page 7](#)). The interface is initiated and controlled from Sentaurus Visual, including control of the simulation.

Table 1 Buttons of the Simulation Control panel of Sentaurus Visual








Button	Description
	Load: Load a command file. When a command file is loaded, it is thereafter referred to as the <i>flow</i> .
	Run: Run the flow. Use this button either to start running the flow or to continue execution after pausing the flow. The simulation continues at the location of the cursor (line with a light-yellow background).
	Pause: Pause the running flow. The pause occurs either when the currently executing step (command) is finished or, for a long-running step, when the current time step is completed.
	Reset: Reset the running flow. The running flow stops, the connection to the simulator is terminated, and you return to the start of the flow.
	Run Step: Run the next step in the flow. When you click this button, either a single step (command) is executed or a group of commands enclosed in braces is executed. You must repeatedly click this button to execute the next steps.

Table 1 Buttons of the Simulation Control panel of Sentaurus Visual

Button	Description
	Save: Save the flow.
	Save As: Save the flow under a new name.

In addition, the `graphics` command can be used to control the plot settings in Sentaurus Visual and to select which fields are visible (see [graphics on page 428](#)).

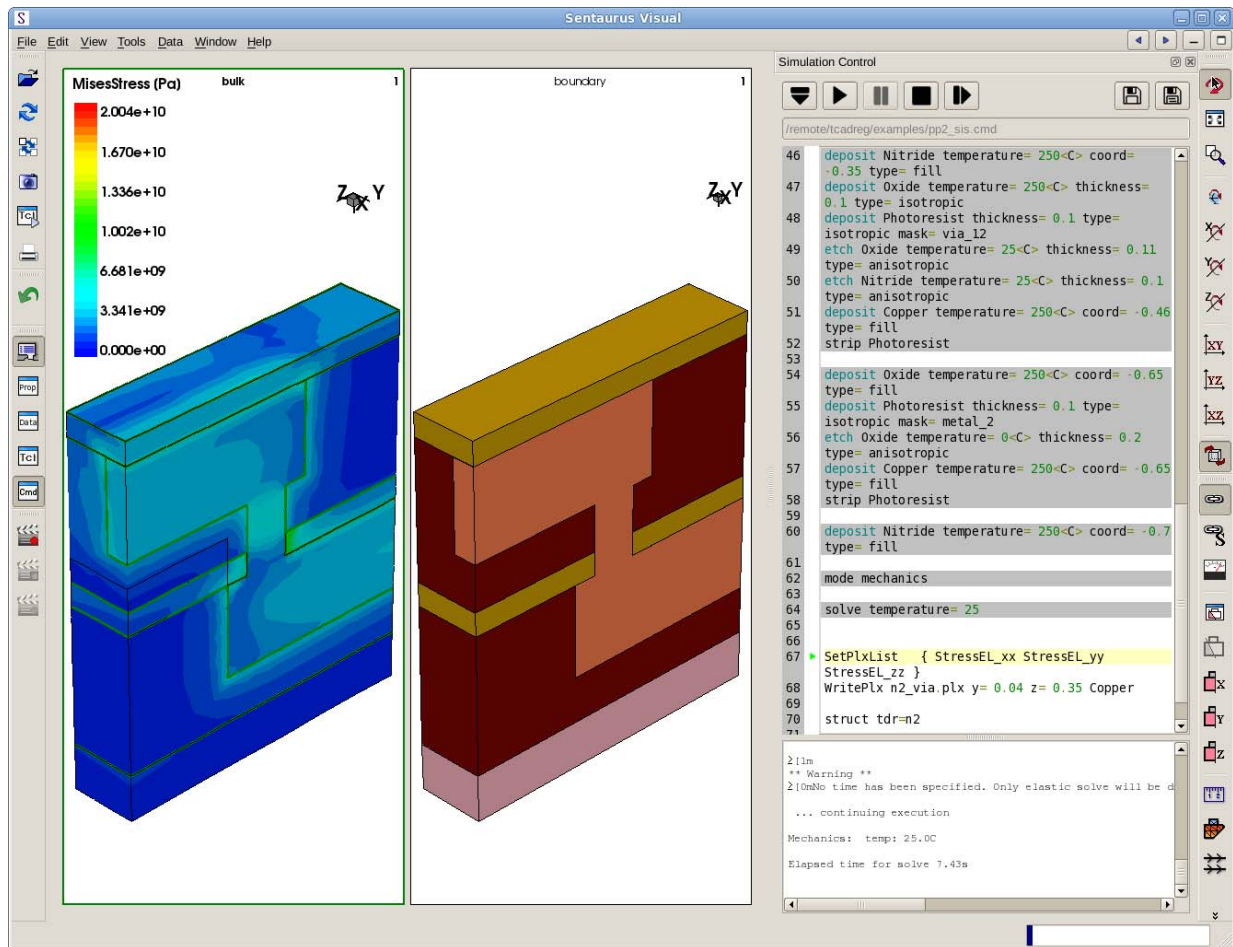


Figure 1 Interface to Sentaurus Visual: upper pane of Simulation Control panel shows command file and lower pane of Simulation Control panel shows log file

Setting Up the Interface

To set up and to run the interface:

1. Launch Sentaurus Visual with the following command-line options:

```
> svisual -spi &
```
2. In Sentaurus Visual, choose **Edit > Preferences**.
3. In the User Preferences dialog box, expand the categories to **Common > Miscellaneous**.
4. In the Simulator group box:
 - a) In the **Command** field, type the tool binary as well as any command-line option required, for example:

```
sinterconnect -n
```
 - b) In the **Communication Option** field, type `--svi`.
5. Click **Save**.

Loading Command Files

To load a command file:

1. In the Simulation Control panel, click the Load button (see [Figure 1 on page 7](#)).
The Load File dialog box is displayed.
2. Select the command file to open.
3. Click **Open**.

In addition, a command file can be loaded from the command line using:

```
svisual -spi <filename>
```

Inserting Breakpoints in the Flow

To set breakpoints to pause the simulation at a particular step in the flow:

1. Click in the left margin, at the line corresponding to the step (command) where you want the flow to stop.
A red circle marks the breakpoint (see [Figure 2 on page 9](#)).
2. Click the **Run** button to execute the entire flow or the Run Step button to execute individual steps.

NOTE You can set multiple breakpoints in a flow.

Deleting Breakpoints in the Flow

To delete a breakpoint:

- Click the red circle in the left margin.

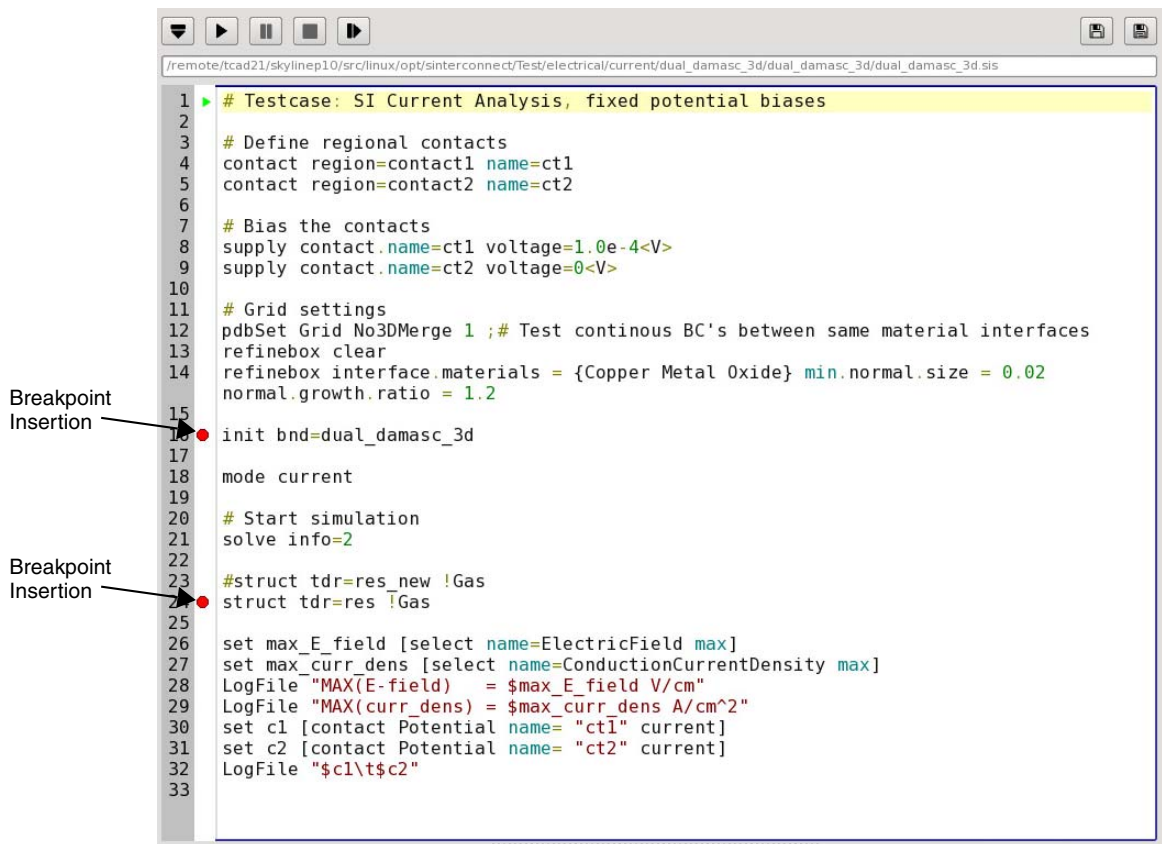


Figure 2 Setting a breakpoint in the flow (red circle indicates location of breakpoint)

Indicating Status of Steps

In the Simulation Control panel, as the flow is being executed, a green triangle in the left margin (in the same location as breakpoints) indicates the step that will be executed next. A red triangle indicates the step that is currently being executed.

Already executed steps are indicated with a gray background. This part of the flow cannot be changed further.

2: Working With the Simulator

Interactive Visualization

In addition, wherever the cursor is in the flow, its position is indicated by highlighting the line with a light yellow background.

Updating the Structure

In 3D simulations, two plots are shown with the titles ‘bulk’ and ‘boundary’ (see [Figure 1 on page 7](#)). This is because, in 3D simulations, both the bulk and boundary are not always up to date. The plot with its title in bold shows the last updated information. For example, if the `insert` command is called in a 3D simulation, the boundary is updated; however, by default, the mesh is not updated. After the insertion operation is completed, the title of the ‘boundary’ plot will be bold, and the title of the ‘bulk’ plot will not be bold.

Controlling the Interface to Sentaurus Visual With the graphics Command

To quickly visualize the evolution of a structure or data, the use of the `graphics` command is not necessary. Simply launch the interface to Sentaurus Visual and adjust the plot settings in Sentaurus Visual.

However, when setting up a flow for the first time or for calibration-type activities, the same command file might need to be run and rerun many times. In these cases, it is convenient to use the `graphics` command to specify the Sentaurus Visual plot settings directly into the command file to avoid having to repeatedly change the plot settings in Sentaurus Visual for each run.

Another case where the `graphics` command is needed is choosing the availability or selection of nonstandard fields for visualization.

In general, `graphics` commands are executed in the usual sequential order as specified in the command file; whereas, settings specified in Sentaurus Visual will be executed immediately. In either case, the latest command to be executed will determine the current settings.

Unless otherwise specified, the arguments of the `graphics` command are independent and can be used in any combination (see [graphics on page 428](#)).

Command-Line Options

Table 2 lists the command-line options that are available in Sentaurus Interconnect.

Table 2 Command-line options

Option	Short name of option (if available)	Function
--batchMode	-b	Switches off graphics.
--diff		Diff mode. To see differences in data and Sentaurus Interconnect parameter settings between two TDR files. Interpolation is used to compare results from different meshes. Usage: sinterconnect --diff <file1> <file2> where <file1> and <file2> are TDR files.
--FastMode	-f	Generates structure, no partial differential equation (PDE) solve, and so on.
--GENESISeMode	-u	Switches off log file creation.
--home <directory>	-o <directory>	Sets SIHOME to <directory>.
--noSyntaxCheck	-n	Switches off syntax check.
--pdb	-p	Runs the Parameter Database (PDB) Browser showing parameters as they are set during runtime. Includes default parameters and parameters from the command file if specified.
--ponly		Same as --pdb, but only shows parameters set in the command file; does not show default parameters.
--quickSyntaxCheck	-q	Only checks syntax of branches that are true.
--svi		Switches on the Sentaurus Visual interface mode. This option is usually specified in the User Preferences dialog box of Sentaurus Visual (see Setting Up the Interface on page 8).
--syntaxCheckOnly	-s	Only checks syntax, no execution.
--xml		Switches on the creation of an XML-style marked-up log file for use in the TCAD Logfile Browser (see Utilities User Guide, Chapter 2 on page 5).
	-v	Prints header with version number.
	-h	Prints use and command-line options.

2: Working With the Simulator

Command-Line Options

Table 2 Command-line options

Option	Short name of option (if available)	Function
	-x	Tests floating-point exception handling.
	-X	Switches off floating-point exception catching.

Interactive Mode

Sentaurus Interconnect runs in interactive mode if no command file is given. In this mode, commands can be entered (at the command prompt) line-by-line and are executed immediately.

It is useful to run Sentaurus Interconnect in the interactive mode for the following reasons:

- When debugging Tcl code, the program does not quit if a Tcl error is found. The error is displayed and you are prompted again for input. You can source a command file repeatedly if required.
- To easily obtain PDB parameter names and defaults with the `pdbGet` command.
- To print the list of built-in functions with the `help` command, and to print the list of Tcl procedures with the `info procs` command.
- To obtain command parameter names and defaults for any built-in command by using the `params` flag available in all built-in functions.

Another use of the interactive mode is to pause the simulation using the `fbreak` command. When the simulation is paused in interactive mode, the state of the simulator can be queried using a number of commands including `grid`, `mater`, and `select`. Pausing the simulation can also be useful when using interactive graphics as described in [Interactive Visualization on page 6](#).

Fast Mode

When working on a new process flow, it is useful to run Sentaurus Interconnect a few times using the fast mode (`-f` command-line option). Developing a new process flow can be complex, involving many `etch`, `deposit`, and `photo` steps, some with masks; sometimes adjustments are required. In the fast mode, all 3D remeshing and all electrical, thermal, and solve commands are ignored. Only process commands for structure generation and analysis are performed. In this mode, when in three dimensions, all `struct` commands will write only a boundary into a TDR file, since the simulation mesh is not synchronized with the modified structure.

Terminating Execution

You can terminate a running Sentaurus Interconnect job in several ways. In some cases, the termination will take time or will fail for other reasons. The most fail-safe method is to use the UNIX command:

```
kill -9 <process_id>
```

where `<process_id>` is the process ID number of the running Sentaurus Interconnect job which can be obtained with the UNIX `ps` command. This sends a signal `SIGKILL` to the corresponding Sentaurus Interconnect job, which will cause the job to terminate immediately.

If Sentaurus Interconnect is run directly from a UNIX shell, usually you can terminate the run by using shortcut keys. The key sequence is interpreted by the shell command, which sends a signal to the job in the foreground. Usually, `Ctrl+C` sends a `SIGINT` signal and `Ctrl+\` (backslash) sends a `SIGQUIT` signal. The running Sentaurus Interconnect job catches all `SIGINT` signals and waits for three signals to be caught (in case it was typed accidentally) before terminating itself.

However, Sentaurus Interconnect does not catch the `SIGQUIT` signal, so this signal will typically cause Sentaurus Interconnect to terminate immediately.

Because the exact behavior may depend on your UNIX shell, the operating system, and the local configuration, refer to the manual for the UNIX shell you are running or contact your local systems administrator for more information.

Environment Variables

The Sentaurus Interconnect binary relies on several supporting files found using the environment variables `SIHOME` and `SCHOME`. To change default models and parameters without modifying the installed Sentaurus Interconnect files, copy the default `SIHOME` and `SCHOME` directories and set the environment variables (`SIHOME` and `SCHOME`) to the location of the modified directories.

By default, `SIHOME` and `SCHOME` are set based on the Synopsys standard environment variables `STROOT` and `STRELEASE`, and by the version number of Sentaurus Interconnect using:

```
SIHOME = $STROOT/tcad/$STRELEASE/lib/sinterconnect-<version number>  
SCHOME = $STROOT/tcad/$STRELEASE/lib/score-<version number>
```

2: Working With the Simulator

File Types

Inside the `SIHOME` directory, there is the subdirectory `TclLib`. Inside the `SCHOME` directory, the two major subdirectories are `TclLib` and `Params`:

- The subdirectory `$SIHOME/TclLib` contains all the default model selections in the file `SINTERCONNECT.models`.
- The Tcl files are located in the subdirectories `$SIHOME/TclLib` and `$SCHOME/TclLib`.
- The subdirectory `$SCHOME/Params` contains the parameter database (see [Parameter Database on page 21](#)).

File Types

The main file types used in Sentaurus Interconnect are:

- Command file (`*.cmd`)

This is the main input file for Sentaurus Interconnect. It contains all the process steps and can be edited. It is referred to as the *command file* or *input file*.

- Log file (`*.log` and `*.out`)

Sentaurus Interconnect generates a `.log` file when it is run from the UNIX command line and an `.out` file when run from Sentaurus Workbench. Whichever file is written contains information about each processing step, and the models and values of physical parameters used in it. The amount of information written to the log file is controlled by the `info` argument, which is available for nearly all commands (see [Common Arguments on page 363](#)). The global default information level (0) can be changed with `pdbSet InfoDefault <n>`. Allowed values of `InfoDefault` are 0, 1, and 2 with higher values indicating more verbose output. Any value higher than 2 will be interpreted as 2.

There is a limit to the size of the log file. By default, it is 1.e9 (~1 GB). The simulation terminates if the limit is reached. This limit can be changed using the double-parameter `Log.File.Limit`:

```
pdbSet Log.File.Limit <value>
```

In addition, you can set the simulation to continue without logging any more entries after the limit is reached using:

```
pdbSet Continue.Past.Log.Limit 1
```

- Marked-up log file (`*.xml`)

When the `--xml` option is specified on the command line, Sentaurus Interconnect generates a separate log file containing XML-like tags. This file contains exactly the same information as the `.log` file. The additional XML-like tags are used to format the `.log` file for efficient access to the information by displaying the structure of the log file content in the TCAD Logfile Browser. Tags for common modules are written automatically. You can

add custom section tags to mark important processing units with the `Section`, `SubSection.Start`, and `SubSection.End` commands. Refer to the *Utilities User Guide* for information about the TCAD Logfile Browser.

- TDR boundary file (*_bnd.tdr)

This file stores the boundaries of the structure without the bulk mesh or fields. It can be used as the structure file for the Sentaurus Mesh meshing engine and can be loaded into Sentaurus Visual for viewing. The name of a TDR boundary file can be specified using the `tdr` argument of the `init` command, and then the loaded boundary will be meshed.

- TDR grid and doping file (*_sis.tdr)

TDR files can be used to split and restart a simulation. Such restart files are saved using the `struct tdr=<c>` command because restarting requires interface data, parameter and command settings, mesh ordering information as well as bulk grid and data. If either `!pdb` or `!interfaces` is specified in the `struct` command, the TDR file will not be suitable for restarting. The TDR file can be loaded into Sentaurus Interconnect using the `init` command, but the results of the subsequent simulation steps might differ in the simulation with the split and restart compared to a simulation of the entire flow in one attempt. TDR files store the following types of information:

- Geometry of the device and the grid.
- Distribution of doping and other datasets in the device.
- The internal structure of the mesh in Sentaurus Interconnect required to restore the simulation mesh to the same state in memory that is present at the time of saving the file. Restart files store coordinates and field values without scaling to prevent round-off errors.
- By default, Sentaurus Interconnect stores all changes to the parameter database made after initial loading of the database and all commands that create objects later referenced, such as refinement boxes and masks in the TDR file. A TDR file can be either reloaded into Sentaurus Interconnect to continue the simulation or loaded into Sentaurus Visual for visualization.

The parameter settings stored in a TDR file can be viewed using `pdbBrowser -nopdb -tdr <tdrfile>` (see [Viewing Parameters Stored in TDR Files on page 30](#)).

For information about the TDR format, refer to the *Sentaurus™ Data Explorer User Guide*.

- XGRAPH file (*.plx)

This file is used to save 1D distributions of the doping concentration or other fields in a specified 1D cross section. This file can be viewed by loading it into Inspect.

Syntax for Creating Command Files

This section describes how to create command files manually. It is important to remember that Sentauros Interconnect is written as an extension of the tool command language (Tcl). This means that the full capability and features of Tcl are available in the command files as well as the interactive mode of Sentauros Interconnect.

Standard Tcl syntax must be followed; for example, a hash symbol (#) at the beginning of a line denotes a comment and the dollar sign (\$) is used to obtain the value of a variable. Major features of Tcl include `for` loops, `while` loops, `if then else` structures, `switch` statements, file input and output, sourcing external files, and defining procedures (functions). Variables can be numbers, strings, lists, or arrays. Refer to the literature for more information [1].

Before the command file is executed, its syntax is checked. This is accomplished by first modifying the command file so that all branches of control structures such as `if then else` and `switch` commands are executed. In addition, a special flag is set so that no structure operations or operations that depend on the structure are performed. This allows the syntax check to run quickly but thoroughly. Sometimes, the modifications made to the command file during syntax checking interfere with the definition or redefinition of Tcl variables, generating a false syntax error. In these cases, switch off syntax checking for the part of a command file using the special `CHECKOFF` and `CHECKON` commands:

```
# Skip syntax check for part of command file
# The CHECKOFF/CHECKON commands must start at the beginning of the line
# and be the only command on the line
CHECKOFF
if { $mode } {
    array set arr $list1
} else {
    set arr $list2    ;# error only if both branches are executed
}
CHECKON
# further commands are syntax checked
```

Tcl Input

Sentauros Interconnect has been designed to optimize the use of Tcl. Some examples of this interaction include:

- Command parameter values are evaluated with Tcl. For example, `expr` can appear in the value of an expression, that is, `parameter=[expr $pp/10.0]` is valid Sentauros Interconnect syntax. This particular expression sets the parameter `parameter` to the value of `pp/10` if the Tcl variable `pp` was previously defined with the Tcl `set` command.

- Tcl expressions may appear in model parameter values in the parameter database. In some cases, Sentaurus Interconnect parameters are set with Tcl commands to be a function of other parameters.
- Sentaurus Interconnect contains many callback procedures, which can be redefined by users to provide flexibility.
- Many modular built-in functions are available for postprocessing, which can be combined into a Tcl script to create powerful analytic tools.
- There are special Sentaurus Interconnect versions of `set` (`fset`) and `proc` (`fproc`), which are stored in TDR files. When simulations are restarted using a TDR file, the settings given by `fset` and `fproc` from the previous simulation will be available.

Other syntax rules to consider when writing command files are:

- One command is entered on one line only. There are two exceptions to this rule:
 - A backslash (\) is used to extend a command on to multiple lines if it appears as the last character on the line.
 - If there is an opening brace, Tcl will assume the command has not finished until the line containing the matching closing brace.
- Command parameters have the following form:
 - Boolean parameters are true if the name appears on the line. They are false if they are preceded by an exclamation mark (!).
 - Parameters that are of type *integer* or *floating point* must appear as `parameter=value` pairs.
 - String parameters are enclosed, in general, in double quotation marks (" "), for example, `parameter="string"`.
 - Lists can be enclosed in double quotation marks or braces, for example:

```
parameter= {item1 item2 ...}  
parameter= "item1 item2 ..."
```

You must have a space between the equal sign and the opening brace.

NOTE It is important to separate the equal sign from the parameter value by a space because Tcl delimiters such as ‘”’ and ‘{’ are ignored if they appear in the middle of a string. Sentaurus Interconnect can handle *no space* between an equal sign and a double quotation mark, but it cannot correct the case where there is *no space* between an equal sign and an opening brace.

- Some parameters take a list of `keyword=value` pairs, for example:

```
epi.doping= {Boron = 1e20 Arsenic = 1e18}
```

2: Working With the Simulator

Syntax for Creating Command Files

NOTE For convenience, all parameters that take a numeric value will have an extra `Tcl eval` applied to them. This means that even if the value of a parameter is enclosed in braces `{ }`, the parameter will still be evaluated. For example:

```
set val1 1.0
set val2 2.0
refinebox xrefine= {$val1 $val2}
```

will be evaluated as:

```
refinebox xrefine= {1.0 2.0}
```

Specifying Materials

Materials are specified in the same way for all PDB commands that require a material parameter. For a bulk material, specify only one material. For an interface material, specify two materials combined with an underscore (`_`).

Some examples of specifying materials with PDB commands are:

```
pdbSet ... Oxide ...           ;# Command applies to oxide.
pdbSet ... Oxide_Silicon ...   ;# Command applies to the Si-SiO2 interface.
```

The order of materials for interfaces is lexical. However, some common material combinations have aliases. For example, `Silicon_Oxide` is an alias for `Oxide_Silicon`.

Materials are specified in the same way for all other commands that require a material parameter. For a bulk material, specify only one material. For an interface material, specify two materials: one without a slash and one with a slash (`/`).

Some examples of specifying materials with other commands are:

```
<command> ... oxide           ;# Command applies to oxide.
<command> ... silicon /oxide  ;# Command applies to the Si-SiO2 interface.
```

The complete list of materials available can be found in the file:

```
$STROOT/tcad/$STRELEASE/lib/score-<version number>/TclLib/tcl/Mater.tcl
```

In that file, the lines that contain `mater` add create a material. For more information about creating new materials, see [mater on page 497](#).

NOTE Materials present in the `Mater.tcl` file do not necessarily have parameters in the parameter database. Attention must be paid to initializing parameters for a new material.

Aliases

Sentaurus Interconnect allows more control over the names of command parameters, the abbreviations of parameter names, as well as interface names. These *aliases* only apply to parameters of built-in Sentaurus Interconnect commands, and the `pdbSet` and `pdbGet` family of commands.

This permits clarity and uniformity to commonly used names. Another benefit is that it is easier to maintain backward compatibility for parameter names while not restricting future parameter names that could conflict with common abbreviations (that is, *V* could refer to either *vacancy* or *void*).

An explicit list of allowed aliases is maintained in the `$SCORE/TclLib` directory (see [Environment Variables on page 13](#) for information about how the location of the `TclLib` directory is determined). The `alias` command is used to view and extend the list of allowed aliases (see [alias on page 368](#)).

To print the list of aliases:

```
sinterconnect> alias -list
```

To view the alias of a parameter name, for example, `Vac`:

```
sinterconnect> alias Vac  
Vacancy
```

If an alias does not exist, the same parameter name is returned:

```
sinterconnect> alias NotAParam  
NotAParam
```

To create a new alias for a parameter name, for example, the alias `Vaca` for the parameter `Vacancy`:

```
sinterconnect> alias Vaca  
Vaca  
sinterconnect> alias Vaca Vacancy  
sinterconnect> alias Vaca  
Vacancy
```

For interface names, aliases for the sides can be used irrespective of the order. For example, if `Ox` is an alias for `Oxide` and `Si` is an alias for `Silicon`, then `Ox_Si` or `Si_Ox` is automatically an alias for the `Oxide_Silicon` interface. This flexibility does not apply to side-specific interface parameters. For example, `TransferRate_Si` is not automatically an alias for `TransferRate_Silicon`.

Default Simulator Settings: SINTERCONNECT.models File

Sentaurus Interconnect starts a simulation by reading the `SINTERCONNECT.models` file in the `$(SIHOME)/TclLib` directory. This file defines various default parameters and directories used during the simulation such as:

- The path for Tcl library files
- Default material names
- The `math` parameters for 1D, 2D, and 3D simulations
- Default solution names
- Default callback procedures

The `SINTERCONNECT.models` file is read once at the beginning of the simulation. You can override any of the default parameters after the file is read.

Compatibility With Previous Releases

Occasionally, the default parameter and model settings change in Sentaurus Interconnect to ensure that the default behavior gives robust, accurate, and computationally efficient results on current production technologies. Usually, when new models and algorithms are developed, they are optional. After some experience is gained, the default can be changed to take advantage of the new model or algorithm.

The old model and algorithm settings are collected into a file for each release and are available so that you can recover results from previous releases. Each file contains only those parameter changes that occurred for that particular release, so that if the release specified in the `Compatibility` command is older than the most recent release, the most recent release parameters are set first, followed by older releases in reverse chronological order (see [Compatibility on page 380](#)).

For example, the command `Compatibility K-2015.06` issued for Version N-2017.09 will first apply parameters consistent with M-2016.12, then with parameters consistent with L-2016.03, and finally parameters consistent with K-2015.06. Aliases are available for the release name so you do not need to know the release foundation letter. For example, `2015.06` can be used instead of `K-2015.06`.

Files with the compatibility parameter settings are stored in `$(STROOT)/tcad/$(STRELEASE)/lib/sinterconnect/TclLib/Compatibility`. These files provide a useful list of all default parameter changes for each release.

NOTE As a result of code corrections and numeric accuracy limitations, exact reproduction of results from previous releases is not always possible.

NOTE If the `Compatibility` command is used, it must be the first command in a command file so that all subsequent commands that depend on the defaults take into account the compatibility setting.

For example, to apply the defaults of Version M-2016.12, the first line of the command file must be:

```
Compatibility 2016.12
```

NOTE The `Compatibility` command does not change the default parameter and algorithm settings for Sentaurus Mesh, Sentaurus Structure Editor, and the MGOALS module. To change the backward compatibility setting for MGOALS, see [MGOALS Backward Compatibility on page 271](#).

Parameter Database

The parameter database stores all Sentaurus Interconnect material and model parameters as well as global information needed for save and reload capabilities. There is a hierarchical tree directory inside the `Params` directory, which stores the default values. (To locate the `Params` directory, see [Environment Variables on page 13](#).)

Data is retrieved using the `pdbGet` command and is set using the `pdbSet` command. The `pdbGet` and `pdbSet` commands are checked for correctness of syntax, and they print the allowed parameter names if a mistake is made. These commands are used to obtain and set all types of data stored in the parameter database: Boolean, string, double, double array, and switch.

The higher-level `pdbSet` and `pdbGet` commands call lower-level type-specific commands (`pdbGetBoolean`, `pdbGetDouble`, `pdbGetDoubleArray`, `pdbGetString`, `pdbGetSwitch`, `pdbGetSwitchString`, `pdbSetBoolean`, `pdbSetDouble`, `pdbSetDoubleArray`, `pdbSetString`, and `pdbSetSwitch`) that are not checked for errors and, therefore, are not recommended for typical use. These commands have a slight performance advantage and are used internally.

You can set some parameters in a region-specific manner. Regions can be named with the `region` and `deposit` commands and, if region-specific parameters exist, they will override the material-specific parameters if any. However, there are many circumstances where this will not give the desired behavior. In that case, you must create a new material that inherits its

2: Working With the Simulator

Parameter Database

parameters from an existing material. Then, you must change the material properties of the new material as needed (see [Like Materials: Material Parameter Inheritance on page 23](#)).

Inside the `Params` directory are subdirectories that define the highest level nodes in the database. Inside each subdirectory is a file `Info`, which contains parameters of that level. In addition, directories in the database have named files that contain parameters, which are under the node defined by the file name. For example, in the `Params` database, there is a directory called `Silicon`, which contains a file `Info`. The parameters inside `Info` are located under the `Silicon` node. As another example, the `Potential` file inside the `Silicon` directory is another node. This node contains parameters that are related to the `Potential` under the `Silicon` node.

Inside the files of the parameter database are commands that set the database parameters. The commands have the form:

```
array set $Base { <NAME> { <TYPE> <VALUE> } }
```

where:

- `<NAME>` is the parameter name.
- `<TYPE>` is one of `Boolean`, `String`, `Double`, `DoubleArray`, or `Switch`.
- `<VALUE>` is a Tcl expression that sets the default value.

It is often necessary to enclose the `<VALUE>` expression in braces. Some Tcl procedures have been created to increase the usefulness of `<VALUE>` expressions, such as the `Arrhenius` function.

If you start Sentaurus Interconnect and call the `pdbGet` command of a parameter that contains an `Arrhenius` function, it will return the temperature-dependent `Arrhenius` function of that parameter. In Sentaurus Interconnect, temperature is a local variable and can be changed with the `SetTemp` command. In addition, the `solve` command changes the local temperature for each time step.

Other functions that appear in the `pdb` parameters are `pdbGet*` functions, which allow parameters to be set as a function of other parameters.

For the `DoubleArray` type, a Tcl list is set that is ordered pairwise:

```
{key1 value1 key2 value2 ...}
```

 where the parameter setting for `key1` is `value1`.

Material parameters can be stored under the known region name. To set and obtain the parameter value, use the region name instead of the material name. If the parameter is not found under the region name, it is taken from the material of that region.

Sentaurus Interconnect writes directly to the parameter database in a number of ways. Mostly this is performed to save information for reload capabilities using the TDR format. Data written

by the program into the parameter database is not available within the default `Params` directory or the Parameter Database Browser, but can be read using the `pdbGet` command.

For information about the TDR format, refer to the *Sentaurus™ Data Explorer User Guide*.

Like Materials: Material Parameter Inheritance

The parameters of a material can be inherited from the parameters of another material using the special `Like` parameter in the PDB. When this is the case, the two materials are referred to as *like materials*. This can be used to specify different settings in different regions. First, a new material is created and made to be *like* an existing material using:

```
mater add name=<c> new.like=<c>
```

where:

- `name` specifies the name of the material to be created.
- `new.like` is the name of the existing material from which all default values are inherited.

NOTE It is important to use the `mater` command instead of directly creating the `Like` parameter because the `mater` command will make all interfaces to the new material like the appropriate interface to the existing material.

Interpolation Between Like Materials

By default, data is interpolated between *like* materials, for example, when you insert a region that overlaps an existing region of a like material. The inheritance direction does not matter; either the inserted material is like the existing material, or the existing material is like the inserted material.

To prevent interpolation between a material and other materials that are like it, use:

```
mater name=<c> add !like.interpolate
```

Interface Parameters

When using the Parameter Database commands and the Alagator language, interfaces are specified as a pair of materials separated by an underscore (`_`), for example, `Gas_Oxide` and `Oxide_Silicon`. The official name follows alphabetic order, and the first letter is capitalized.

However, aliases are provided that allow their order to be reversed; some shorter names are allowed; and all lowercase is generally available.

As an example of setting an interface parameter, the following command sets the numeric tolerance `Abs.Error` at the gas-silicon interface to `1e3`:

```
pdbSet Gas_Silicon Vac Abs.Error 1e3
```

Regionwise Parameters and Region Name-Handling

Many parameters in the parameter database can be specified regionwise including parameters related to meshing and mechanics parameters. Those parameters used by Alagator as part of equations and terms, however, cannot be specified regionwise: this includes all solution-related parameters. For the rest of the parameters, internally, the program checks if there is a regionwise specification of the parameter; if not, the materialwise specification is used.

The name of regions can be specified with the `region` command and `deposit` command; however, the name:

- Must not contain an underscore (`_`) or a period (`.`) because these characters have special meaning.
- Must be different than an existing material name.

During the course of the simulation, geometric operations such as `etch` and `reflect` can split regions in two. If this happens, the history of the region is maintained through its name. For example, if a region is originally named `layer1` and it is etched into two pieces, they will be named `layer1.1` and `layer1.2` according to rules given below.

These two regions will inherit the parameters of `layer1`. Furthermore, parameters for `layer1.1` and `layer1.2` also can be specified separately. If a subsequent step such as a `deposit` reunites `layer1.1` and `layer1.2`, the region will be given the name `layer1`. Conversely, if `layer1.1` is split into two regions, the regions will be named `layer1.1.1` and `layer1.1.2`, and so on. In this way, regionwise parameter specification is preserved for the life of the region or its parts.

The numbering of split regions is performed according to the spatial location of the pieces. The lowest point of each piece to be renamed is found (in the coordinate system of Sentaurus Interconnect, this would be the largest x-coordinate). To avoid numeric noise, the coordinates are compared with a specified epsilon given by `pdbGet Grid RenameDelta` (hereafter, referred to as `RN`). If the x-coordinates of the pieces to be renamed are not within `RN` of each other, the regions are ordered from lowest to highest, that is, from the highest x-coordinate to the lowest. If any piece has its lowest coordinate within `RN`, its y-coordinate is compared, that is, from the lowest coordinate to the highest.

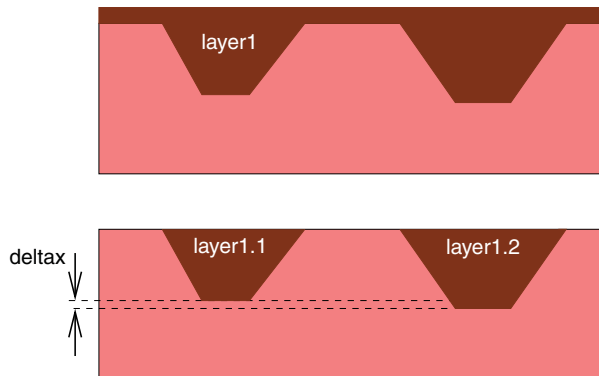


Figure 3 Illustration of region-naming rules

For example, in [Figure 3](#), `layer1` is split into two regions and the quantity `deltax` is less than `RN`, so the region on the left is given the name `layer1.1` and the region on the right is given the name `layer1.2`. If `deltax` had been greater than `RN`, the region on the right would have been given the name `layer1.1` because it would have been considered lower than the region on the left. Similarly, in three dimensions, first `x` and `y` are compared, and if they are both within `RN`, `z` is used for ordering, that is, from the lowest coordinate to the highest.

You can apply the above operation to the whole structure with `grid rename`. In this case, all the regions are renamed similarly to the above rules but, instead of the root being chosen by the user, all regions of the same material have the root given by the names of the materials and the extension is `_<n>` where `<n>` is the region number, for example `Silicon_1`, `Silicon_2`, and so on. This should only be used as a postprocessing step because all region-specific parameters no longer apply when the name of a region has changed.

Sentaurus Interconnect automatically unites regions with the same material type. For example, if silicon is deposited on top of an existing silicon region, both regions are united, so there will be only one silicon region. If regions must be united and region names do not follow the rules mentioned in this section, the united region will take the name of one of the materials in the united region.

Parameter Database Browser: Viewing the Defaults

The Parameter Database (PDB) Browser is a graphical representation of the parameter database that allows you to view and edit parameters. The PDB Browser has three distinct areas (see [Figure 4 on page 26](#)):

- Parameter hierarchy overview in a tree structure representation.
- Parameter information in a spreadsheet representation. The columns are Parameter, Type, Value, Unit, Evaluate, Comment, Tool, and Info Level (hidden by default).

2: Working With the Simulator

Parameter Database Browser: Viewing the Defaults

- Graphic window to plot the parameter dependency on the temperature.

The status bar has three indicators that show:

- The temperature used in temperature-dependent functions such as Arrhenius.
- The temperature point set for the x-axis.
- The x-coordinate and y-coordinate of the pointer in the graphic window.

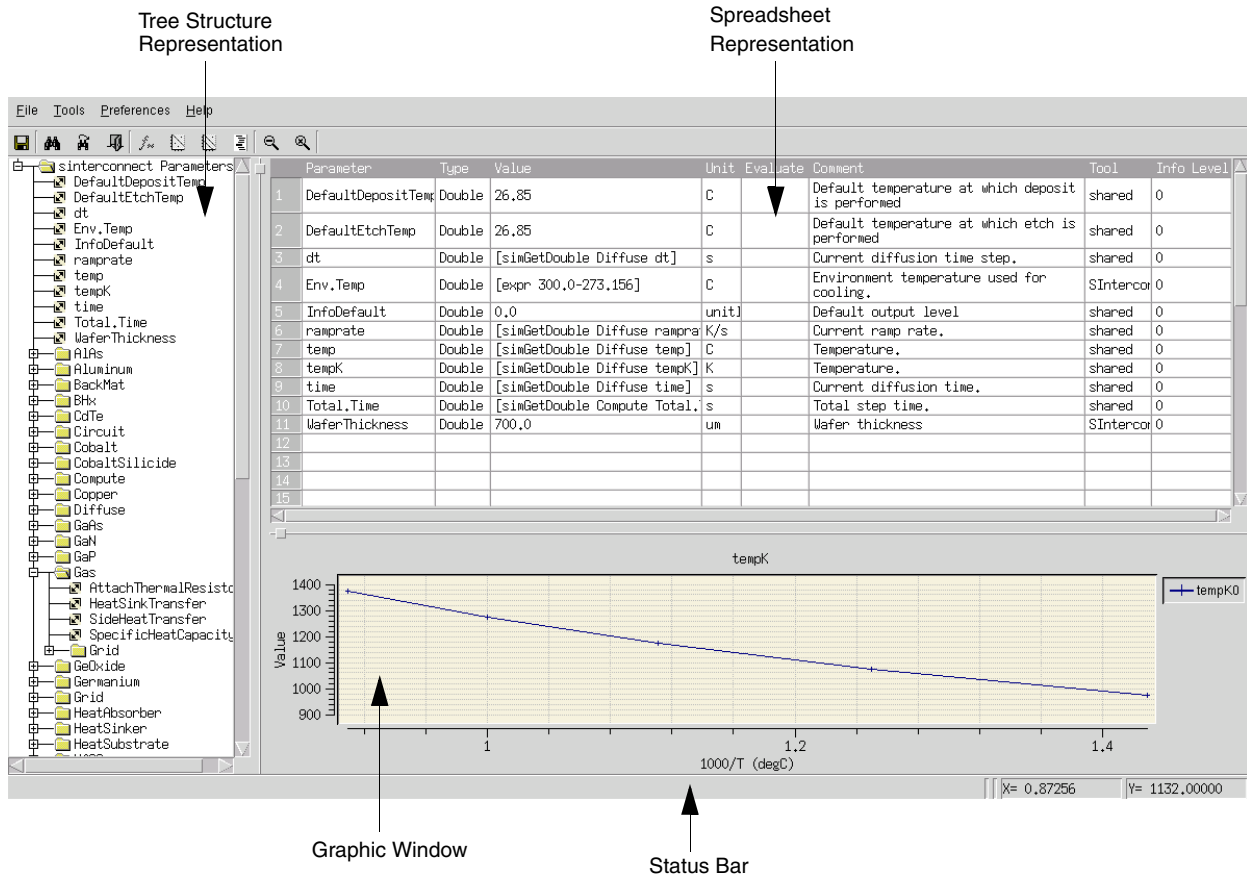


Figure 4 Parameter Database Browser

Starting the Parameter Database Browser

To start the PDB Browser from the command line, type:

```
pdbBrowser
```

This searches for the parameter database in the same location as Sentaurus Interconnect.

You can set the environment variables `SIHOME` and `SCHOME` to change the location of the parameter database for the PDB Browser and Sentaurus Interconnect (see [Environment Variables on page 13](#)).

You can select **Tools > Filter** to choose which parameters to display.

To view parameters in a command file merged with defaults, use:

```
sinterconnect --pdb <command file>
```

To view only the parameters specified as input in a command file, use:

```
sinterconnect --ponly <command file>
```

PDB Browser Functions

The following functions are available from the **File** and **Tools** menus:

File > Export Node

Saves the selected node into a specified file in tab-delimited format.

File > Export Tree

Saves the entire parameter database into a specified file in tab-delimited format. The fields of the file are **Parameter Name**, **Type**, **Value Evaluation**, **Original Value**, and **Comments**.

Tools > Evaluate

Evaluates the value of the selected parameter and displays the result in the Evaluate column of the spreadsheet. Values can contain Tcl expressions.

Tools > Plot

(Applies only to parameters of type *double* and *double array*.) Plots the dependency of the selected parameter on the temperature in logarithmic coordinates versus $1/T$. The default set of temperature values is {700.0 800.0 900.0 1000.0 1100.0}. The resulting graphs are displayed in the graphic window; otherwise, an error message is displayed.

Tools > Plot Over

The same as **Plot** but it does not clear the graphic window of previous graphs.

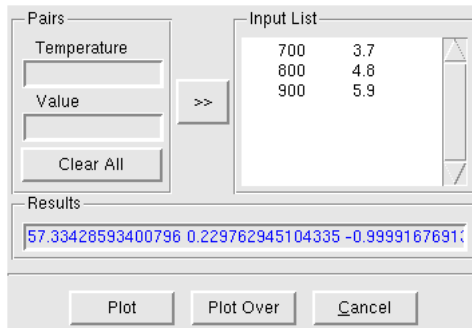
NOTE You can zoom in by dragging the mouse. To zoom out, use the middle mouse button, or click the **Zoom Out** and **Zoom Off** buttons.

2: Working With the Simulator

Parameter Database Browser: Viewing the Defaults

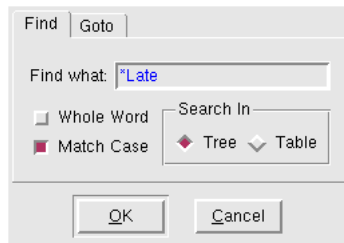
Tools > Arrhenius Fit

Using the Arrhenius Fit dialog box, you can find the best prefactor and energy for an Arrhenius fit of a given profile, taken from the list of temperature–value pairs. The results can be plotted in the graphic window:



Tools > Find, Tools > Find Next

Matches the pattern entered against parameter names according to the options selected in the Find dialog box. Patterns can include regular Tcl expressions. The match is highlighted when found:



Tools > Goto Line

Highlights a table row or tree node that corresponds to the number entered.

Tools > Filter

Selects which parameters to display.

Tools > Info Level

Chooses which parameters to display ranging from basic parameters to all parameters.

Viewing Parameter Information

Double-clicking a nonempty cell in the spreadsheet allows you to view the corresponding parameter information in a separate window. To close the window, click the **Close** button.

NOTE To display a shortcut menu, right-click a parameter and select commands for editing, evaluating, or plotting.

PDB Browser Preferences

The PDB Browser allows you to reset the default settings for the following values using the **Preferences** menu, the shortcut keys, or the shortcut menu of the graphic window:

Preferences > Editor > Change Editor

Resets the default editor.

Preferences > Editor > Reset Update Time

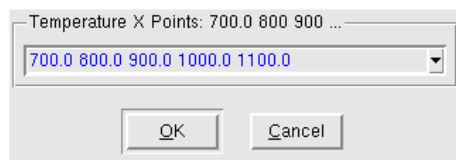
Resets the update interval.

Preferences > Graph > Set Temperature

Sets the global temperature used in the temperature-dependent functions. Default: 1000.0.

Preferences > Graph > Reset X Points

Displays the Reset Temperature Points dialog box where you set the x-axis temperature points. The default set is {700.0 800.0 900.0 1000.0 1100.0}:



Preferences > Graph > Data Point Symbol

Sets the symbol to use for data points.

Preferences > Graph > X Scale

Resets the x-scale to logarithmic or linear.

Preferences > Graph > Y Scale

Resets the y-scale to logarithmic or linear.

Preferences > Tree Node

Hides the node tip or shows the node tip.

2: Working With the Simulator

Parameter Database Browser: Viewing the Defaults

Preferences > Info Level

Shows or hides the Info Level column of the spreadsheet.

Preferences > Font > Family

Changes the font family.

Preferences > Font > Size

Changes the font size.

Preferences > Cursor

Changes the style of the cursor.

Viewing Parameters Stored in TDR Files

Parameters stored in TDR files can be viewed using the `pdbBrowser` command run from the command line instead of through Sentaurus Interconnect. By default, the PDB Browser reads parameters from the Sentaurus Interconnect database directory (which can be changed with the `SIHOME` and `SCHOME` environment variables). In addition, parameters stored in a TDR file can be read in using the `-tdr <filename>` option of the PDB Browser. Parameters that appear in the parameter database are overwritten by those contained in the TDR file, so the resultant parameter set will be the same as if Sentaurus Interconnect had read in the file. On the other hand, it is also useful to know which parameters are only in the TDR file. To read only those parameters, the database reading can be switched off using the `-nopdb` command-line option.

For example, the following command reads the PDB Browser and then reads the parameters from the `n10_sis.tdr` file, overwriting values contained in the parameter database:

```
> pdbBrowser -tdr n10_sis.tdr
```

For example, the following command reads only the parameters in the `n10_sis.tdr` file:

```
> pdbBrowser -nopdb -tdr n10_sis.tdr
```

Understanding Coordinate Systems

Sentaurus Interconnect and related tools use different coordinate systems.

Wafer Coordinate System

The wafer coordinate system is fixed with respect to the wafer flat or notch, and is used to define the relationship of all other coordinate systems to the physical wafer.

The wafer x- and y-axes form a naturally oriented coordinate system when the wafer is drawn with the flat pointing down as shown in [Figure 5](#). This coordinate system is used for layout information, such as mask locations, and for setting a cutline using the `CutLine2D` command.

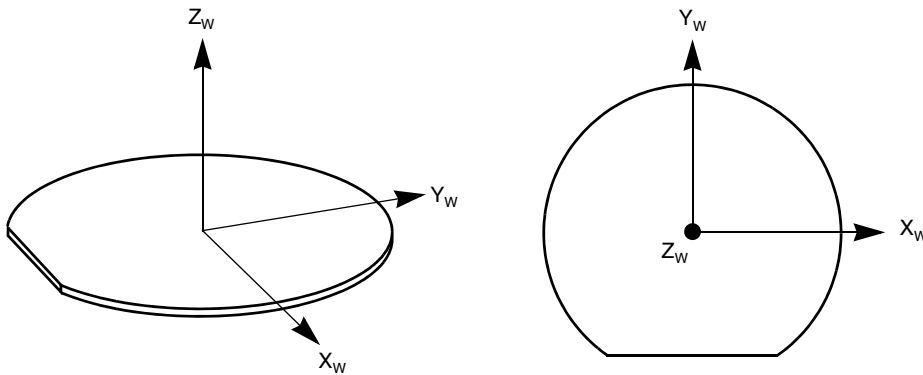


Figure 5 Wafer coordinate system

Simulation Coordinate System

The simulation coordinate system is used to define the mesh for the simulation. The default coordinate system is the *unified coordinate system* (UCS).

The visualization coordinate system is the UCS as well.

2: Working With the Simulator

Understanding Coordinate Systems

In the UCS, the x-axis points into the wafer and the y-axis is rotated with respect to the wafer y-axis. [Figure 6](#) shows the UCS. Simulations in one dimension use only the x-axis. Simulations in two dimensions use only the x- and y-axes.

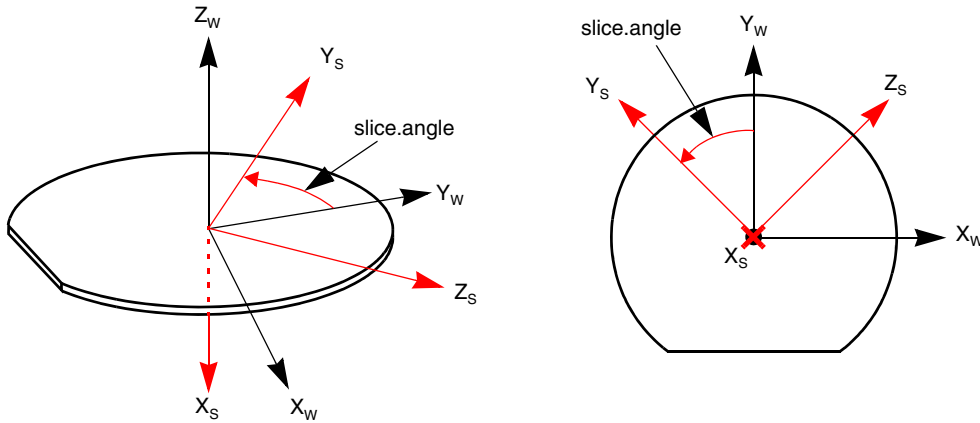


Figure 6 Unified coordinate system (slice.angle= 45)

The rotation of the simulation axes with respect to the wafer axes is given by the `slice.angle` parameter of the `init` command. The slice angle is measured from the wafer y-axis to the simulation y-axis with positive angles counterclockwise about the wafer z-axis.

The default value of `slice.angle` is -90° . This causes the simulation y-axis to match the wafer x-axis, which is the usual cut direction through the layout for 2D simulations (see [Figure 7](#)).

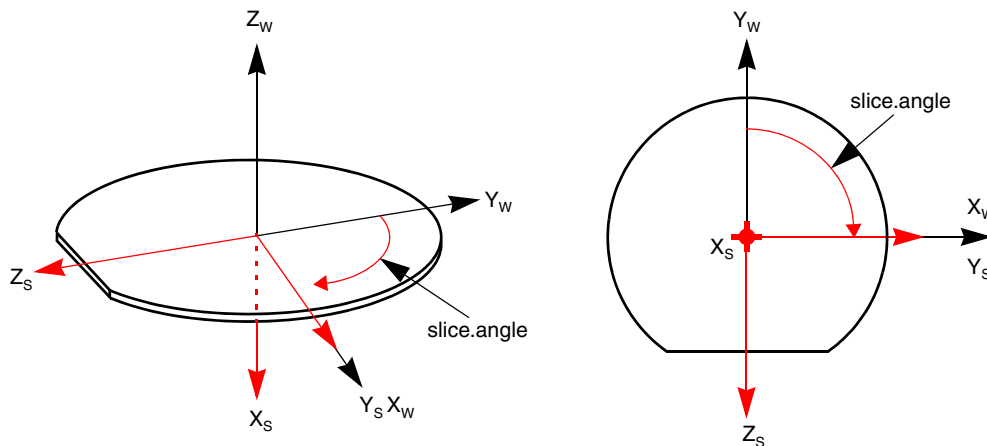


Figure 7 Unified coordinate system when using default value of slice.angle (-90°)

NOTE For backward compatibility, the DF-ISE coordinate system is still available, but it is not recommended.

Creating and Loading Structures and Data

The first step in most simulations is either to load an existing structure or to create a new one. New structures are created through a combination of the `line`, `region`, and `init` commands. The initial mesh is a tensor-product mesh where the density of lines is specified using the `line` command, and the regions are defined by specifying tags in the `line` commands and defined in the `region` command. The initial regions are always defined as axis-aligned rectangles in two dimensions and axis-aligned bricks in three dimensions.

Defining the Structure: The `line` and `region` Commands

The `line` and `region` commands are used together to define the structure. In the `init` command, the structure is actually formed. Care must be taken when creating a structure because there are few checks for errors.

These rules must be followed to obtain a valid structure:

- If this is not the first structure being created in a command file, the command `line clear` must be issued to remove `line` commands and stored mesh ticks.
- Line locations must be given in increasing order.
- The region boundaries are defined by tagged lines. Tagged lines are created with the `line` command where the parameter `tag` has been set (as well as the `location` parameter).
- At least one `region` command must be given to define the substrate.
- Regions must have a material specification, except for the substrate case described below.
- Regions must have the same dimensionality as the `line` commands used (that is, if `line y` is given, a 2D region is expected with `ylo` and `yhi` set in the `region` command).
- The `spacing` parameter of the `line` command is used to create lines between user-defined lines, so that not every line must be specified in the command file. Sentaurus Interconnect smoothly grades the line density between user-defined lines to match as closely as possible the spacing at each user-defined line. In addition, there will be lines at locations given by the `location` parameter of the `line` command. By default, the `spacing` parameter is extremely large, so that if it is not set, only lines given by the `location` parameter will be in the mesh.
- The `*lo` parameter refers to the lowest coordinate value, that is, the location of the line corresponding to the `xlo` tag must be less than the coordinate corresponding to the `xhi` tag.
- The `region` command can be used to tag a region as a substrate in two ways:
 - If the region is being defined with the material name and the parameters `*hi` and `*lo`, the Boolean keyword `substrate` will tag this region as the substrate.

2: Working With the Simulator

Creating and Loading Structures and Data

- If the structure is being loaded from a previously saved file, the command:

```
region name=<c> substrate
```

will tag the region with <c> as the name of the substrate. This is the only occasion when the `region` command will be called *after* the `init` command.

Considerations when creating structures are:

- For 2D and 3D simulations, it is advantageous to create a coarse mesh in the lateral (that is, y- or z-directions) because lines created with the `line` command run all the way through the structure. Often, finer spacing in the y- or z-direction is required near the surface; whereas, further in the bulk, a coarser spacing is required (to minimize the size of the problem).
- When MGOALS is used for etching and deposition, it automatically creates a local refinement near interfaces that does not run the length of the structure.
- To specify refinement boxes, use the `refinebox` command.

Creating the Structure and Initializing Data

The `init` command is used to create the structure. If the `line` and `region` commands have been given to create a structure from the beginning, the `init` command does not require any options. It will take the structure definition and create a new structure.

Many process steps such as etching and deposition require a gas mesh. By default, Sentaurus Interconnect does not add a gas mesh during the `init` command, but delays creating the gas mesh until it is needed. To add the gas mesh immediately, use the command:

```
pdbSet Grid AddGasMesh 1
```

NOTE The parameter must be set before the `init` command to generate the gas mesh during the `init` command.

There are different ways to initialize fields at the time the initial structure is created from `line` and `region` commands:

- To initialize data everywhere in the structure, a field specification can be given in the `init` command.
- To initialize data in one particular region only, a field specification is given in the `region` command.

In both the `init` and `region` commands, the `field` parameter specifies the name of the data field that will be created, and the `concentration` parameter is used to specify the value created. The `init` command must not be used to initialize stress values (use instead either the `stressdata` command or the `select` command).

The `init` command also can read a structure from a file, for example:

```
init tdr= file      ;# Read geometry, data, and PDB parameters from file.tdr
```

The TDR format is the default format and can store all the information necessary to restart a simulation when using ‘splits’ in Sentaurus Workbench. This format stores all `pdb` parameter settings as well as numerous other settings coming from commands (see [Saving a Structure for Restarting Simulations on page 39](#)).

The `init` command specifies the principal wafer orientation (`wafer.orient`) and the lateral crystal orientation of the wafer flat or notch (`notch.direction`).

Defining the Crystal Orientation

Generally, the orientation of a hexagonal crystal system can be described using four Miller indices $\langle ijtk \rangle$; whereas, only three Miller indices $\langle ijk \rangle$ are needed for other systems. In a hexagonal crystal system, the sum of the first three indices equals zero ($i+j+t = 0$); therefore, the third index t is redundant and can be omitted. For simplicity and consistency, the wafer orientation and the flat orientation are specified using three Miller indices for all lattice systems including hexagonal.

The crystal orientation of the wafer is established by specifying the Miller indices of the wafer surface and the wafer flat. The `wafer.orient` and `notch.direction` (originally `flat.orient`) arguments of the `init` command specify the Miller indices of the wafer z-axis and negative y-axis, respectively. The wafer surface orientation (whose surface normal is chosen as the wafer z-axis) is set using `wafer.orient= {<i> <j> <k>}` where $\langle i \rangle$, $\langle j \rangle$, and $\langle k \rangle$ are the crystallographic (Miller) indices of the plane. The notch direction or the flat orientation (a direction that coincides with the wafer negative y-axis) can be set arbitrarily, but it must be orthogonal to the surface normal of the wafer orientation.

The default surface orientation is 100 and the default flat orientation is a 110 direction for all lattice systems.

NOTE The `wafer.orient` and `notch.direction` arguments of the `init` command apply to any crystalline systems in the structure. However, these settings are superseded by the material-specific wafer orientation (`vertical.orient`) and flat orientation (`horizontal.orient`) as specified in the `mater` command.

2: Working With the Simulator

Creating and Loading Structures and Data

Table 3 lists the crystallographic directions of the wafer axes for the most common crystallographic orientations of the wafer as shown in Figure 5 on page 31.

Table 3 Miller indices of wafer axes for each value of wafer.orient (wafer axes are defined in Figure 5)

Wafer orientation	X _w	Y _w	Z _w
100	[1 $\bar{1}$ 0]	[110]	[001]
110	[001]	[1 $\bar{1}$ 0]	[110]
111	[$\bar{1}$ $\bar{1}$ 2]	[1 $\bar{1}$ 0]	[111]

To facilitate simulations of hybrid orientation technology, Sentaurus Interconnect predefines three materials (`Silicon`, `Silicon110`, and `Silicon111`) for crystalline silicon. These materials have exactly the same properties, except for the default crystal orientations that are $\langle 100 \rangle$, $\langle 110 \rangle$, and $\langle 111 \rangle$ for `Silicon`, `Silicon110`, and `Silicon111`, respectively.

Automatic Dimension Control

The maximum dimension of a simulation is determined by the specified `line` commands:

- `line x` commands define the extensions in the vertical direction and are required for 1D, 2D, and 3D simulations.
- If, in addition, `line y` commands are specified, the maximum dimension of the simulation will be at least two dimensions.
- If, in addition, `line z` commands are specified, the maximum dimension of the simulation will be three dimensions.

By default, Sentaurus Interconnect delays the creation of a full-dimensional structure until it becomes necessary. This means that if you specify a 2D structure where all regions span the entire simulation domain in the y-direction, Sentaurus Interconnect will create a 1D structure.

When a 2D or 3D mask is used in an `etch`, a `deposit`, or a `photo` command, Sentaurus Interconnect automatically extrudes the structure and the mesh into the appropriate dimension and copies the data. This delay of creating a full-dimensional structure can be switched off in the `init` command using the option `!DelayFullD`. To increase the dimension manually, use the `grid` command. If a 2D structure is required, that is, both the `line x` and `line y` commands but no `line z` commands have been specified, `grid 2D` or `grid FullD` will cause a 2D structure to be created.

Similarly, if `line x`, `line y`, and `line z` commands have been specified, `grid 2D` can be used to extrude a 1D structure to two dimensions, and a 1D or 2D structure is extruded to three dimensions using `grid 3D` or `grid FullD`. This functionality also can be used to increase the dimension of structures loaded from files. After the structure has been loaded, `line`

commands can be issued and the dimension of the structure will increase automatically when necessary or manually using the `grid` command.

Sentaurus Interconnect does not provide a facility to reduce the dimension of a simulation.

When structures are saved to TDR files (other than TDR restart files), the current maximum dimension as specified with `line` commands is used by default in the file. The dimension of the simulation itself is not affected. To save files in the current dimension, the `!FullD` parameter of the `struct` command can be used (see [Saving Structures on page 38](#)). TDR restart files are always saved in the dimension currently used in the simulation.

Interpolating Field Data From an External Structure File Using the `load` Command

The `load` command can be used to interpolate data from a TDR file onto the current structure (see [load on page 487](#)). For 1D structures, 1D TDR files are allowed. For 3D structures, 3D TDR files are allowed. However, for 2D structures, either a 2D TDR file or a 3D TDR file is allowed.

When loading a 3D file to a 2D structure, the data on the 2D overlap cross-section, between the 3D structure and the 2D structure, is interpolated to the current 2D mesh. The 2D structure is assumed to be located at $z=0$.

Several options can handle the new and old datasets. For example, the `merge` option loads only datasets that do not currently exist in the structure, the `sum` option adds new datasets and sums the matching datasets, the `replace` option adds new datasets and replaces existing datasets with new datasets of the same name, and the `rename` option adds new datasets and renames them by adding the suffix `__load`. These actions also can be applied individually to selected datasets using the `species` and `actions` arguments.

For example, the following command sums `Potential_Saved` and the existing `Potential_Saved` (if available), and replaces the existing `Temperature` data field by the one in the TDR file:

```
load tdr= in species= {Potential_Saved Temperature} actions= {sum replace}
```

The external structure also can be translated or rotated before interpolation with the `transform` argument. This is especially useful when loading 3D data to a 2D structure since the cross section must be at $z=0$ (the 2D structure is assumed to be allocated at $z=0$). For example, the following command loads the 3D data at $z=0.5$ instead of $z=0$:

```
load tdr= source3d transform= { 1 0 0 0 1 0 0 0 1 0. 0. -0.5 }
```

2: Working With the Simulator

Saving Structures

While the following command loads the 3D data at $y=0$ since the 3D structure was rotated, and the y-axis and the z-axis were exchanged:

```
load tdr= source3d transform= { 1 0 0 0 0 1 0 1 0 0. 0. 0. }
```

To ensure the transformation is performed correctly, you may save a TDR file for the transformed structure with the `save.transform` argument.

Loading 1D Profiles: The profile Command

The `profile` command can load a 1D profile into 1D, 2D, or 3D structures (see [profile on page 583](#)). The file to be read must contain one x-coordinate data pair per line. Both linear and logarithmic interpolation are available. For example, profiles are loaded using:

```
profile infile= file.dat name= Boron
```

In this case, Sentaurus Interconnect reads the `file.dat` file and sets the Boron field accordingly.

Saving Structures

Sentaurus Interconnect uses the TDR file format for saving the structure geometry with and without the bulk mesh and data, and with contacts. TDR files contain simply connected regions to operate smoothly with other Synopsys TCAD tools. One important option available for saving files is to omit saving gas regions because this may cause problems for other tools.

The TDR format allows you to save and load geometry and data information along with `pdb` parameters (see [File Types on page 14](#)).

TDR files can be used to split a simulation, and to restart and continue the simulation as if no file save or file load was performed. Besides the simulation grid and data, additional information is stored to facilitate such a restart.

NOTE It is recommended to set the simulation coordinate system using the `math coord.ucs` command, which specifies the use of the unified coordinate system (UCS). When using the UCS, the visualization coordinates are identical to the simulation coordinates.

To select the fields stored in TDR files, use the `SetTDRList` command. Each field name in the `SetTDRList` command is added to the list of fields, which are usually saved (if the field is present in the structure). This command also takes a macro parameter `Solutions`. `Solutions` refers to variables of partial differential equations (PDEs). The solution variables

must be stored in a TDR file if that file is to be used to continue a simulation. By default, TDR files are saved with `Solutions` names in `SetTDRList`. However, this requires many fields to be stored in the TDR files and, sometimes, it is more convenient to have fewer fields.

To do this, set `!Solutions` in `SetTDRList`, which deselects all fields. Then, specify the field names to be stored in the TDR file.

Saving a Structure for Restarting Simulations

When saving files using the TDR format, the current state of the parameter database is, by default, saved in the file. The parameter database contains all of the information necessary to restart a simulation including:

- Model settings
- Parameter settings
- Geometric tolerance settings
- Refinement boxes from the `refinebox` command
- Temperature ramps from the `temp_ramp` command
- Line specifications from the `line` command
- Region specifications from the `region` command
- Specifications for `point`, `polygon`, `polyhedron`
- User materials created with the `mater` command
- Contact definitions created with the `contact` command
- Mask definitions created with the `mask` command
- Solution commands can be optionally stored using the `store` parameter of the `solution` command
- Term commands can be optionally stored using the `store` parameter of the `term` command
- Global Tcl variables can be stored with `fset`
- Tcl procedures can be stored using `fproc`

By default, when loading a TDR file, the changes in the parameter database are read in from the TDR file and are applied. For information about the TDR format, refer to the *Sentaurus™ Data Explorer User Guide*.

When saving a TDR file, the simulation coordinate system used is also included in the file and is used by Sentaurus Visual when opening the file.

2: Working With the Simulator

Saving Structures

The visualization coordinate system is the same as the simulation coordinate system. To change the coordinate system, use the `math` command. For the UCS, use:

```
math coord.ucs
```

NOTE It is recommended to always use the `math coord.ucs` command.

Saving a Structure for Device Simulation

In general, the main steps to saving a structure appropriate for device simulation are:

1. Define contacts.
2. Remesh the structure with appropriate refinement for device simulation.
3. Save the structure with contacts and with Delaunay weights.

Contacts are defined using the `contact` command. To define contacts, you can either:

- Use a *box* where the contact is created at the intersection of a material interface and a box.
- Use a *region* contact in which a region is specified by giving a point inside the region; then all boundaries of this region become a contact.

The contact is given a name and, if the command is executed multiple times with the same contact and the `add` parameter, the contact will include all parts specified. There are also options for creating a contact on the outer boundaries and so on (see [contact on page 381](#)).

Remeshing the structure is needed to create a mesh that is better suited to device simulation. Typically, this means discarding process-based refinements, creating a very fine mesh under the channel, and refining on the p-n junction. A typical sequence of steps is:

- Clear the process mesh:

```
refinebox clear  
line clear
```

- Reset default settings for adaptive meshing:

```
pdbSet Grid AdaptiveField Refine.Abs.Error 1.e37  
pdbSet Grid AdaptiveField Refine.Rel.Error 1e10  
pdbSet Grid AdaptiveField Refine.Target 100.0
```

- Set high-quality Delaunay meshes:

```
pdbSet Grid SnMesh DelaunayType boxmethod
```

- Set mesh spacing near interfaces:

```
pdbSet Grid SnMesh min.normal.size <n>  
pdbSet Grid SnMesh normal.growth.ratio.2d <n>  
pdbSet Grid SnMesh normal.growth.ratio.3d <n>
```

- Set which interfaces will have interface refinement:

```
refinebox interface.materials= {Silicon}
```

- Specify adaptive refinement:

```
pdbSet Grid Adaptive 1
```

- Specify lines if necessary:

```
line y loc= $Ymin+0.001  
line z loc= $Zmin+0.001
```

- Specify refinement boxes, for example:

```
refinebox min= <list> max= <list> xrefine= <list> yrefine= <list>  
zrefine= <list> ;# gate refinement  
  
refinebox refine.fields= NetActive max.asinhdiff= {NetActive= 1.0} \  
refine.min.edge= <list> Silicon ;# adaptive refinement on NetActive
```

- If you use the IC WorkBench Edit/View Plus–TCAD Sentaurus interface, it may be helpful to consider using the mask argument of the `refinebox` command (see [Chapter 13 on page 285](#) and [refinebox on page 591](#)).

To save the structure for device simulation, use the command `struct tdr=<c> !Gas`. This command causes a remesh if necessary, stores any contacts that have been defined previously, and includes fields required for device simulation.

Delaunay weights can be saved in the structure intended for device simulation by setting these parameters before generating the mesh:

```
pdbSet Grid SnMesh StoreDelaunayWeight 1  
pdbSet Grid Contacts.In.Brep 1
```

The first parameter `StoreDelaunayWeight` creates the field variable `Delaunay–Voronoi weight (DelVorWeight)` that is used in the weighted box method in Sentaurus Device. The second parameter `Contacts.In.Brep` switches on a feature that creates contacts in the boundary representation (brep) and prevents changes to the mesh that can locally invalidate the Delaunay weight.

Saving 1D Profiles for Inspect

To store `.plx` files, use the `WritePlx` command. The command `SetPlxList` selects the fields to be stored in the `.plx` file. The `SetPlxList` command is similar to the `SetTDRList` command, except that no fields are selected by default. Only the field names specified in `SetPlxList` are stored in the `.plx` file (see [SetPlxList on page 619](#) and [WritePlx on page 681](#)).

Saving 1D TDR Files From 2D and 3D Simulations, and 2D TDR Files From 3D Simulations

The command `struct` also saves a 1D TDR file if the proper cutting coordinates are specified (see [struct on page 643](#)). In two dimensions, only one cutting coordinate is needed (either `x` or `y`; the `z`-coordinate makes no sense here). In three dimensions, the command saves the intersection of the planes specified by two cutting coordinates (for example, specifying `x` and `z` will save the `y` line containing those `x`- and `z`-coordinates). In addition to storing the mesh and data, these files save any contacts that apply at the cut point, so that the file can be loaded into Sentaurus Device for electrical analysis. This file can be visualized with Sentaurus Visual.

For 3D simulations, the `struct` command also saves 2D TDR files when one cutting coordinate is specified.

For example, in a 2D simulation, the following command picks up all the `x`-coordinates with `y=0.5` and saves them in a 1D TDR file:

```
struct tdr= filename y= 0.5
```

In addition, in a 3D simulation, the following command saves the `y`-coordinates with `x=0.2` and `z=0.1` as a 1D TDR file:

```
struct tdr= filename x= 0.2 z= 0.1
```

The following command saves a 2D cross section of the 3D structure to a 2D TDR file at `x=0.2`:

```
struct tdr= filename x= 0.2
```

The select Command (More 1D Saving Options)

The `select` command is a versatile command for many operations such as viewing results, postprocessing, and initializing or changing datasets. The basic command is:

```
select z=<C>
```

where `<C>` is an Alagator expression (see [Chapter 16 on page 343](#)). A simple example of an expression is the name of a data field such as `Potential` and `vTotal`. The value of the expression is stored in the selected field.

This selected field can be viewed with `print.data` or `print.1d`, for example, or the integrated values can be obtained using the `layers` command. The `select` command can also be used to set an existing data field or create a new data field, for example:

```
select z= 1.0 name= MyDataField ;# create a new data field named MyDataField
```



```
                ;# and set it to 1.0 (everywhere)  
select z= 0.1*Vacancy name= Void store      ;# Set Void equal to 0.1*Vacancy
```

The datexcodes.txt File

The `datexcodes.txt` file is the Synopsys configuration database for materials, doping species, and other quantities that are used in semiconductor process and device simulations. Sentaurus Interconnect uses this file in two ways:

- The "floops" property of each field is read in and is used as a conversion factor between short internal names and proper DATEX names expected by other tools.
- The unit field is read from the `datexcodes.txt` file and is used to convert internal units to those units expected by other tools.

For example, the distributed `datexcodes.txt` file contains the following field definition:

```
VacancyConcentration {  
    label      = "total Vacancy concentration"  
    symbol     = "VTtotal"  
    unit      = "cm^-3"  
    factor     = 1.000E+12  
    precision  = 4  
    interpol  = arsinh  
    material  = All  
    alter1    = vacancy  
    alter2    = 0  
    property("floops") = "VTtotal"  
}
```

From this definition, Sentaurus Interconnect reads `property("floops")` to enable conversion of the internally named field `VTtotal` to `VacancyConcentration` in the TDR file and to save `VacancyConcentration` in cm^{-3} .

References

- [1] B. B. Welch, *Practical Programming in Tcl & Tk*, Upper Saddle River, New Jersey: Prentice Hall PTR, 3rd ed., 2000.

2: Working With the Simulator

References

This chapter discusses the computation of mechanical stress in Sentaurus Interconnect.

Overview

Mechanical stress has an important role in process modeling. It controls the structural integrity of the device, the yield from the process depends on stresses, the mobility of charged carriers is changed by stresses, and leakage currents also are a function of the stress in the system.

In modern process flows, accurate computation of stress is important. However, there is a continual trend toward designing process flows that produce the right types of stress in the device. With appropriate stresses, device performance can be enhanced significantly.

Stress computation simulations are performed in the following distinct steps:

1. Define the equations for mechanics. The equations used in Sentaurus Interconnect define force equilibrium in the quasistatic regime.
2. Define the boundary conditions for these equations. For the elliptic equations that arise from the equations of force equilibrium, boundary conditions are needed on all boundaries. Sentaurus Interconnect allows Dirichlet or Neumann boundary conditions, provided that certain criteria are met. The minimum criterion is to constrain the structure sufficiently so that it has no rigid body modes.
3. Define material properties. This step defines the relationship between stresses and strains. Some materials may hold stresses for a given strain without relaxing; these are *elastic materials*. Other materials may relax the stresses away; these are *viscous or viscoelastic materials*. Sentaurus Interconnect provides viscoelastic constitutive equations for the computation of mechanical stresses. By setting parameters appropriately, the extreme cases of a purely viscous material and a purely elastic material can be simulated as well. The viscoelastic models used in Sentaurus Interconnect provide a choice between the Maxwell model and the standard linear solid model. The viscosity can depend on the local shear stresses, which make the viscosity a locally varying quantity and can lead to nonlinear mechanical behavior. In addition to elastic and viscoelastic materials, there are materials to model irreversible deformation and temperature-dependent volume change. Sentaurus Interconnect provides nonlinear material models for incremental plasticity, deformation plasticity, viscoplasticity and creep, and swelling.

3: Computing Mechanical Stress

Material Models

4. Define the mechanisms that drive the stresses. In Sentaurus Interconnect, this is performed through intrinsic stresses, thermal mismatch, and densification. All these processes are additive in the linear elastic regime. In the nonlinear regime, they must be updated from the available stress history.

Stress is solved in all materials. Parameters describing material behavior, which will be introduced in this chapter, can be found in the parameter database:

```
<material> Mechanics
```

In the following sections, the constitutive equations are discussed in detail. These tensor equations can be split into two components:

- The *dilatational component*, which corresponds to the trace of the tensor, describes the material behavior in the case of a pure volume change.
- The *deviatoric component* describes an arbitrary deformation but without changing the volume.

For example, the strain tensor can be decomposed as follows:

$$\varepsilon_{ij} = \underbrace{\varepsilon'_{ij}}_{\text{deviatoric}} + \underbrace{\frac{1}{3}\left(\sum_k \varepsilon_{kk}\right)\delta_{ij}}_{\text{dilatational}} \quad (1)$$

This decomposition will be used in subsequent equations to discuss the constitutive equation for the dilatational and deviatoric components independently.

Material Models

Sentaurus Interconnect implements the viscous, viscoelastic, and elastic models in a general manner, where the viscous model and elastic model can be derived from the viscoelastic model. The viscous and viscoelastic models use shear stress–dependent viscosity. The elastic model also has anisotropic elasticity where the elastic coefficients depend on the crystal orientation. The plasticity models describe the material behavior beyond yield, independent of the rate of loading. The viscoplasticity and creep models provide rate-dependent plastic behavior, while the swelling material models temperature-dependent volume change.

Viscoelastic Materials

The viscoelastic material response is characterized by elastic and viscous components. The combined response depends on how elastic and viscous stresses or strains are coupled. Sentaurus Interconnect provides two commonly used combinations:

- Maxwell model
- Standard linear solid model

Maxwell Model

The viscoelastic behavior for the Maxwell model is obtained by combining elastic and viscous responses in series. The stress–strain equations are written in terms of dilatational and deviatoric components. The equations for the volumetric component of the stress tensor¹ take the form:

$$\frac{\dot{\sigma}_v}{K} + \frac{\sigma_v}{\eta_v(T, \sigma_s)} = 3\dot{\epsilon}_v \quad (2)$$

$$\sum_k \sigma_{kk} = -3p = 3\sigma_v$$

where η_v is the bulk viscosity. In addition, the relation of the stress and strain tensor to the hydrostatic pressure p is shown. The bulk modulus K can be computed from the Poisson ratio (`PoissRatio`) and Young's modulus (`YoungsMod`) as:

$$K = \frac{\text{YoungsMod}}{3(1 - 2 \cdot \text{PoissRatio})} \quad (3)$$

The deviatoric component of the stress tensor is described by:

$$\frac{\dot{\sigma}'_{ij}}{G} + \frac{\sigma'_{ij}}{\eta'(T, \sigma_s)} = 2\dot{\epsilon}'_{ij} \quad (4)$$

where η' is the shear viscosity. The shear modulus G can be computed from the Poisson ratio and Young's modulus as:

$$G = \frac{\text{YoungsMod}}{2(1 + \text{PoissRatio})} \quad (5)$$

1. The subscripts of vectors and tensors hold for the Cartesian coordinates x , y , and z .

3: Computing Mechanical Stress

Material Models

By default, the viscoelastic response is applied to the deviatoric components. The linear elastic model is used for the pressure–volume response, that is:

$$\sigma_v = K \sum_k \varepsilon_{kk} \quad (6)$$

To apply the viscoelastic response to both the deviatoric components and the volumetric component, use:

```
pdbSet Mechanics NoBulkRelax 0
```

The shear viscosity η' is a function of the shear stress and the temperature T , where:

$$\eta'(T) = \text{Viscosity0} \cdot \exp\left(-\frac{\text{ViscosityW}}{k_B T}\right) \quad (7)$$

Usually, the value of `viscosityW` is negative and, therefore, the shear viscosity η' decreases with increasing temperature. The bulk viscosity has a similar Arrhenius expression defined by the parameters `viscosity0.K` and `viscosityW.K`. The dependency on the shear stress σ_s is discussed in [Shear Stress–Dependent Viscosity on page 50](#).

Standard Linear Solid Model

In the standard linear solid model, the material behavior is modeled by combining the elastic response in parallel with the Maxwell model–based viscoelastic response:

$$\sigma_{ij} = \sigma_{ij}^{\text{el}} + \sigma_{ij}^{\text{ve}} \quad (8)$$

where σ_{ij}^{el} is the elastic stress and σ_{ij}^{ve} is the viscoelastic stress. The difference compared to the Maxwell model allows the total stress to be nonzero even after the viscoelastic stress has relaxed away:

$$\sigma_{ij}(t \rightarrow \infty) = \sigma_{ij}^{\text{el}} \quad (9)$$

The dilatational and deviatoric components of the elastic and viscoelastic stresses are written in the usual form:

$$\begin{aligned} \sigma_v^{\text{el}} &= 3K_{\text{base}} \varepsilon_v \\ \sigma_{ij}^{\text{el}} &= 2G_{\text{base}} \varepsilon'_{ij} \\ \frac{\dot{\sigma}_v^{\text{ve}}}{K} + \frac{\sigma_v^{\text{ve}}}{\eta_v(T, \sigma_s^{\text{ve}})} &= 3\dot{\varepsilon}_v \\ \frac{\dot{\sigma}_{ij}^{\text{ve}}}{G} + \frac{\sigma_{ij}^{\text{ve}}}{\eta'(T, \sigma_s^{\text{ve}})} &= 2\dot{\varepsilon}'_{ij} \end{aligned} \quad (10)$$

where:

- K_{base} and G_{base} are the bulk and shear moduli for the elastic response, respectively.
- K and G are the bulk and shear moduli for the elastic component of the viscoelastic response, respectively.
- η_v and η' are the bulk and shear viscosities, respectively.
- ε_v and ε'_{ij} are the dilatational and the deviatoric components of mechanical strain, respectively.

To enable the standard linear solid model, use:

```
pdbSetSwitch <material> Mechanics ViscoElasticity.Model SLS-Maxwell
```

The default value for the above parameter is `Maxwell` for the Maxwell model.

The elastic response for the standard linear solid model is inactive, by default, so that the material behavior is similar to that of the Maxwell model. The elastic response can be activated by providing nonzero values for the bulk and shear moduli:

```
pdbSetDouble <material> Mechanics BaseBulkModulus <n>
pdbSetDouble <material> Mechanics BaseShearModulus <n>
```

The material parameters for the viscoelastic response are specified in the same way as for the Maxwell model. By default, the dilatational component of viscoelastic stress is assumed to be purely elastic:

$$\sigma_v^{\text{ve}} = 3K\varepsilon_v \quad (11)$$

To activate the viscoelastic response for the dilatational component, use:

```
pdbSet Mechanics NoBulkRelax 0
```

To visualize elastic and viscoelastic responses, this model provides additional output fields. Stresses and strains for the elastic response can be viewed with the `BaseStressEL` and `BaseElasticStrainEL` fields, respectively. Creep strains $\varepsilon_{ij}^{\text{cr}}$ for the viscoelastic response can be viewed with the `CreepStrainEL` field:

$$\varepsilon_{ij}^{\text{cr}} = \int_0^t \left(\frac{\sigma_v^{\text{ve}}}{2\eta_v} + \frac{\sigma'_{ij}}{2\eta'} \right) dt \quad (12)$$

The solution for viscoelastic stress is time dependent. It also becomes nonlinear when viscosity is a function of viscoelastic shear stress. Therefore, the Newton method is used to solve for stresses. At the end of each Newton iteration, a check is made on whether the convergence criteria have been satisfied. More iterations are performed until all the criteria are satisfied within the specified tolerance or until the maximum number of iterations is reached. For details

about convergence criteria and time-stepping for mechanics, see [Time-Step Control for Mechanics on page 338](#).

Purely Viscous Materials

Oxide and nitride, by default, are treated as viscoelastic materials. However, the viscosity is a function of temperature (see [Eq. 7](#)). With increasing temperature, the viscosity decreases, that is, the material becomes increasingly more liquid. When the viscosity reaches a very low value, the first term in [Eq. 4](#) can be neglected:

$$\frac{\sigma'_{jk}}{\eta'(T, \sigma_s)} = 2\dot{\epsilon}'_{jk} \Leftrightarrow \sigma'_{jk} = 2\eta'\dot{\epsilon}'_{jk} \quad (13)$$

[Eq. 13](#) describes the deviatoric component of a purely viscous material. The relaxation time $\tau = \eta'/G$ typically gives a good estimate of the behavior of a viscoelastic material. If τ is much greater than the process time, the material is in the elastic regime. The material behaves viscoelastically if τ is in the range of the process time. If τ is very small, the material is in the viscous regime.

Shear Stress–Dependent Viscosity

For viscous and viscoelastic materials, the viscosity may depend on the temperature and the shear stress σ_s . The temperature dependency is described by [Eq. 7](#). The dependency on the shear stress is given by:

$$\eta(\sigma_s, T) = \eta(T) \cdot \frac{\sigma_s/\sigma_{\text{crit}}}{\sinh(\sigma_s/\sigma_{\text{crit}})} \quad (14)$$

The shear stress σ_s is computed from the local stress distribution based on the second invariant of the deviatoric component of the stress tensor:

$$\sigma_s = \sqrt{\frac{3}{2} \sum_j \sum_k \sigma'_{jk} \sigma'_{kj}} \quad (15)$$

The viscosity breakdown value σ_{crit} can be determined by:

$$\sigma_{\text{crit}} = \frac{2k_B T}{v_{\text{crit}}} \quad (16)$$

where:

$$v_{\text{crit}}(T) = v_{\text{crit}0} \cdot \exp\left(-\frac{V_{\text{crit}W}}{k_B T}\right) \quad (17)$$

By default, oxide and nitride are treated as viscoelastic materials with shear stress–dependent viscosity. The values for V_{crit0} and V_{critW} also are set in the parameter database:

```
pdbSetDouble <material> Mechanics Vcrit0
pdbSetDouble <material> Mechanics VcritW
```

Purely Elastic Materials

If the viscosity in [Eq. 4, p. 47](#) is chosen high enough, the second term on the left can be neglected and the equation reads:

$$\frac{\dot{\sigma}'_{jk}}{G} = 2\dot{\epsilon}'_{jk} \Leftrightarrow \sigma'_{jk} = 2G\epsilon'_{jk} \quad (18)$$

This equation describes the deviatoric component of a purely elastic material. By default, silicon and polycrystalline silicon are treated as purely elastic materials. To achieve this, the viscosity of these materials is set to 1×10^{40} poise.

NOTE K and G are the primary parameters describing elastic materials, and not Young’s modulus and the Poisson ratio. When changing material properties with the `pdb` command, only a change to the primary parameters affects the simulation. To obtain Young’s modulus and the Poisson ratio, use the commands [KG2E on page 477](#) and [KG2nu on page 478](#), respectively.

NOTE When material data is given in terms of Young’s modulus and the Poisson ratio, use the commands [Enu2G on page 411](#) and [Enu2K on page 412](#) to convert them to the shear modulus and the bulk modulus, respectively.

Anisotropic Elastic Materials

The stress and strain relations for anisotropic elastic materials can be described using:

$$\sigma_i = C_{ij}\epsilon_j \quad (19)$$

where σ_i and ϵ_j are the components of the engineering stress and strain, respectively, and C_{ij} is the component of the stiffness matrix. The engineering stress σ_i ($i=1, \dots, 6$) corresponds to the stress-tensor components $\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{yz}, \sigma_{xz}$, and the engineering strain ϵ_j ($j=1, \dots, 6$) corresponds to the strain-tensor components $\epsilon_{xx}, \epsilon_{yy}, \epsilon_{zz}, 2\epsilon_{xy}, 2\epsilon_{yz}, 2\epsilon_{xz}$.

NOTE The engineering shear-strain components differ from the shear-strain tensor components by a factor of 2.

Cubic Crystal Anisotropy

The mechanical responses of a crystalline solid vary along various crystal orientations. For a cubic crystal, the axes of reference are chosen to be parallel to the crystal axes. In a coordinate system with axes aligned along the crystal axes, the symmetric stiffness matrix C has the following nonzero components: $C_{11} = C_{22} = C_{33}$, $C_{12} = C_{23} = C_{13}$, $C_{44} = C_{55} = C_{66}$.

All other components are zeros. The anisotropic stress and strain relation is completely defined when three independent modulus parameters C_{11} , C_{12} , and C_{44} are specified.

The degree of anisotropy for a given material can be measured by the departure from unity of the ratio $A = 2 C_{44} / (C_{11} - C_{12})$. The anisotropic model reduces to the isotropic model if the ratio A is equal to 1. When the simulation coordinate axes do not coincide with the crystal axes, the stiffness matrix C must be transformed accordingly. For this, note that C is actually a rank-4 tensor.

By default, the anisotropic elasticity model is switched off. The following command switches on the model:

```
pdbSet Silicon Mechanics Anisotropic 1
```

The values of these three modulus parameters with respect to the cubic crystal axis can be defined using the following commands, which also show the default values for the crystalline silicon:

```
pdbSet Silicon Mechanics C11 16.57E11  
pdbSet Silicon Mechanics C12 6.39E11  
pdbSet Silicon Mechanics C44 7.96E11
```

The unit for these default values is dyn/cm^2 .

This model depends on the `wafer.orient` and `slice.angle` parameters specified in the `init` command.

Hexagonal Crystal Anisotropy

The mechanical responses of a crystalline solid vary along various crystal orientations. Hexagonal close-packed crystals contain a plane of isotropy. In a coordinate system with axes aligned along the crystal axes, the symmetric stiffness matrix C has the following nonzero components: $C_{11} = C_{22}$, C_{33} , C_{12} , $C_{13} = C_{23}$, $C_{44} = C_{55}$, $C_{66} = (C_{11} - C_{12})/2$.

All other components are zeros. The anisotropic stress and strain relation is completely defined when five independent modulus parameters C_{11} , C_{12} , C_{13} , C_{33} , and C_{44} are specified.

When the simulation coordinate axes do not coincide with the crystal axes, the stiffness matrix C must be transformed accordingly. The transformation depends on the crystal orientation information specified in the `init` command and the `mater` command.

The values of these five modulus parameters with respect to the crystal axis can be defined using the following commands, which also show the default values for crystalline GaN:

```

pdbSet <material> Mechanics C11 39.0E11
pdbSet <material> Mechanics C12 14.5E11
pdbSet <material> Mechanics C13 10.6E11
pdbSet <material> Mechanics C33 39.8E11
pdbSet <material> Mechanics C44 10.5E11

```

The unit for these default values is dyn/cm^2 .

The hexagonal anisotropic elasticity model is applied to certain wurtzite III–V nitride materials. The following command switches on the model:

```

pdbSet <material> Mechanics Anisotropic 1

```

The material must be set to crystalline with hexagonal lattice type, for example:

```

pdbSetBoolean <material> Amorphous 0
pdbSetString <material> LatticeType Hexagonal

```

Orthotropic Model

Orthotropic materials have three planes of symmetry. In a coordinate system with axes aligned along the symmetry planes, the symmetric stiffness matrix C has the following nonzero components: C_{11} , C_{22} , C_{33} , C_{44} , C_{55} , C_{66} , $C_{12} = C_{21}$, $C_{13} = C_{31}$, $C_{23} = C_{32}$. The symmetry planes of the model coincide with the simulation coordinate system, and the axes 1, 2, and 3 become axes X, Y, and Z in the simulation coordinate system, respectively.

Orthotropic material properties can be described by specifying nine independent parameters, namely, the Young's moduli in the symmetry planes (E_x , E_y , E_z), the directional shear moduli (G_{xy} , G_{xz} , G_{yz}), and the directional Poisson ratios (ν_{xy} , ν_{xz} , ν_{yz}). The other directional Poisson ratios are calculated from:

$$\frac{\nu_{ij}}{E_i} = \frac{\nu_{ji}}{E_j} \quad (20)$$

where $i, j = x, y, z$.

3: Computing Mechanical Stress Material Models

The stiffness matrix components are calculated from the specified material properties as:

$$\begin{aligned}C_{11} &= E_x(1 - \nu_{yz}\nu_{zy}) \times \gamma \\C_{22} &= E_y(1 - \nu_{xz}\nu_{zx}) \times \gamma \\C_{33} &= E_z(1 - \nu_{xy}\nu_{yx}) \times \gamma \\C_{12} &= C_{21} = E_x(\nu_{yx} + \nu_{yz}\nu_{zx}) \times \gamma \\C_{13} &= C_{31} = E_z(\nu_{xz} + \nu_{xy}\nu_{yz}) \times \gamma \\C_{23} &= C_{32} = E_y(\nu_{zy} + \nu_{zx}\nu_{xy}) \times \gamma \\ \gamma &= (1 - \nu_{xy}\nu_{yx} - \nu_{yz}\nu_{zy} - \nu_{xz}\nu_{zx} - 2(\nu_{yx}\nu_{xz}\nu_{zy}))^{-1}\end{aligned}\tag{21}$$

$$\begin{aligned}C_{44} &= G_{yz} \\C_{55} &= G_{xz} \\C_{66} &= G_{xy}\end{aligned}\tag{22}$$

By default, the orthotropic model is switched off. It is switched on using the command:

```
pdbSet <material> Mechanics Orthotropic 1
```

The material properties can be specified by the command:

```
pdbSetDouble <material> Mechanics <material_parameter> <n>
```

specifically:

```
pdbSetDouble <material> Mechanics YoungsModulusX <n>
pdbSetDouble <material> Mechanics YoungsModulusY <n>
pdbSetDouble <material> Mechanics YoungsModulusZ <n>
pdbSetDouble <material> Mechanics PoissonRatioXY <n>
pdbSetDouble <material> Mechanics PoissonRatioXZ <n>
pdbSetDouble <material> Mechanics PoissonRatioYZ <n>
pdbSetDouble <material> Mechanics ShearModulusXY <n>
pdbSetDouble <material> Mechanics ShearModulusXZ <n>
pdbSetDouble <material> Mechanics ShearModulusYZ <n>
```

The units for Young's modulus and the shear modulus are dyn/cm^2 .

Orthotropic thermal expansion also is considered in this material model, and different coefficients of thermal expansion can be specified along the three symmetry planes:

```
pdbSetDouble <material> Mechanics ThExpCoeffX <n>
pdbSetDouble <material> Mechanics ThExpCoeffY <n>
pdbSetDouble <material> Mechanics ThExpCoeffZ <n>
```

Temperature-dependent material properties can be specified for all the material parameters specified above. The variation of a property ξ can be specified as:

$$\xi(T) = \xi_{\text{ref}} + \xi(T - T_{\text{ref}}) \quad (23)$$

where the reference value is the material parameter value specified in the command file.

The values can be specified as:

```
pdbSet <material> Mechanics <material parameter>Rate <n>
```

For example:

```
pdbSetDouble FR4 Mechanics ThExpCoeffXRate 0
```

These orthotropic parameter entries exist for silicon, and the shorthand `pdbSet` command can be used. The default values are set to replicate isotropic elastic behavior.

NOTE The older parameters using the 1, 2, 3 indices are deprecated and are replaced with the new parameters using the X, Y, Z indices.

NOTE Anisotropic elastic models and plastic or viscoplastic models must not be switched on simultaneously for the same material.

Plastic Materials

Materials such as metals show linear elastic behavior at lower stresses but undergo permanent deformation at higher stresses. At low temperatures, permanent deformation in these materials is not sensitive to the rate of loading. Such material behavior is defined as plastic or elastic-plastic. Depending on the type of loading, plastic deformations may be computed using incremental plasticity or deformation plasticity.

To switch on the plastic material model, use the command:

```
pdbSet <material> Mechanics IsPlastic <n>
```

Incremental Plasticity

Plastic material behavior under nonmonotonic loading is modeled using incremental formulation.

3: Computing Mechanical Stress

Material Models

Incremental plasticity uses the von Mises yield criterion with associative flow and bilinear hardening. The von Mises yield criterion for isotropic solid materials takes the form:

$$F(\sigma'_{ij}, q_{ij}, \alpha) = \sqrt{(\sigma'_{ij} - q_{ij})(\sigma'_{ij} - q_{ij})} - Y(\alpha) = 0 \quad (24)$$

where:

- q_{ij} is the back stress.
- α is an isotropic hardening variable.
- $Y(\alpha)$ is a function describing the change of yield surface with progressive yielding. This function can be set to linear or exponential with:

```
pdbSetSwitch <material> Mechanics Incremental.Plasticity.Yield.Model
<Linear | Exponential>
```

The linear relation takes the form $Y(\alpha) = \sqrt{\frac{2}{3}}(\sigma_y + H_{\text{iso}}\alpha)$ where σ_y is the yield stress in uniaxial tension, and H_{iso} is the isotropic hardening modulus. σ_y and H_{iso} can be set respectively using the commands:

```
pdbSetDouble <material> Mechanics Incremental.Plasticity.SigmaY <n>
pdbSetDouble <material> Mechanics Incremental.Plasticity.Hiso <n>
```

The exponential relation takes the form $Y(\alpha) = \sqrt{\frac{2}{3}}(\sigma_y + R_{\text{iso}}(1 - \exp(-b_{\text{iso}}\alpha)))$ where R_{iso} and b_{iso} are parameters for nonlinear (exponential) isotropic hardening. To set these parameters, use the commands:

```
pdbSetDouble <material> Mechanics Incremental.Plasticity.Riso <n>
pdbSetDouble <material> Mechanics Incremental.Plasticity.Biso <n>
```

Under a small strain assumption, the strains (and strain rates) are decomposed additively:

$$\epsilon_{ij} = \epsilon_{ij}^e + \epsilon_{ij}^p \quad (25)$$

where ϵ_{ij}^e are the elastic strains, and ϵ_{ij}^p are the plastic strains.

For incremental plasticity, the plastic strains are determined by the plastic flow rule:

$$\dot{\epsilon}_{ij}^p = \dot{\gamma} \frac{\partial Q}{\partial \sigma_{ij}} \quad (26)$$

where $\dot{\gamma} \geq 0$ is the slip rate, and Q is the plastic potential. Plastic flow is assumed to be volume preserving, so that plastic strain is purely deviatoric:

$$\epsilon_{ij}^p \delta_{ij} = 0 \Rightarrow \epsilon_{ij}^p = \epsilon_{ij}^p \quad (27)$$

For associative plastic flow, the plastic potential Q is set equal to the yield function F . The evolution of the isotropic hardening variable and the back-stress variable is given by:

$$\begin{aligned}\dot{\alpha} &= \dot{e}^p = \sqrt{\frac{2}{3}}\dot{\gamma} \\ \dot{q}_{ij} &= \frac{2}{3}\dot{\gamma}\left(H_{\text{kin}}\frac{(\sigma'_{ij}-q_{ij})}{\sqrt{(\sigma'_{kl}-q_{kl})(\sigma'_{kl}-q_{kl})}} - \sqrt{\frac{3}{2}}H_{\text{NLkin}}q_{ij}\right)\end{aligned}\quad (28)$$

where H_{kin} is the linear kinematic hardening modulus, H_{NLkin} is the parameter for nonlinear kinematic hardening, and \dot{e}^p is the equivalent plastic strain rate. The Einstein summation convention is used to define the tensor product in this equation.

To set the kinematic hardening parameters, use the commands:

```
pdbSetDouble <material> Mechanics Incremental.Plasticity.Hkin <n>
pdbSetDouble <material> Mechanics Incremental.Plasticity.HNLkin <n>
```

For linear isotropic hardening, the hardening modulus is interpreted as the slope of the stress versus the plastic strain curve (as obtained from uniaxial tension test) $H_{\text{iso}} = \frac{d\sigma}{de^p}$.

It differs from the elastic-plastic tangent modulus, which is defined as the slope of the stress versus total strain curve $E^{\text{ep}} = \frac{d\sigma}{d\varepsilon}$.

To switch on the incremental plasticity model, use:

```
pdbSet <material> Mechanics Plasticity.Model Incremental
```

The rate equations are discretized using the backward Euler scheme and then are solved using a radial return mapping algorithm (see [1] for more details).

The nonlinear nature of the plasticity model requires Newton iterations to achieve the equilibrium state for each loading step. At the end of each iteration, a check on the satisfaction of convergence criteria is made. More Newton iterations are performed until all the convergence criteria are satisfied within the specified tolerance or until the maximum number of iterations is reached. See [Time-Step Control for Mechanics on page 338](#) for details on convergence criteria and time-stepping for mechanics.

NOTE To define the plastic model, use nonzero values for the isotropic or the kinematic hardening modulus along with yield stress. In the absence of hardening, the numeric simulation of plastic deformation may become unstable.

NOTE The PDB parameters `Hardening.Modulus.Isotropic`, `FirstYield`, and `Hardening.Modulus.Kinematic` are deprecated and are replaced by `Incremental.Plasticity.Hiso`, `Incremental.Plasticity.SigmaY`, and `Incremental.Plasticity.Hkin`, respectively.

Deformation Plasticity

Plastic materials that do not have well-defined yield stress can be modeled using deformation plasticity. This model is based on the Ramberg–Osgood formula [2][3], which is only valid for monotonic loading. It is used mostly for plastic deformation around crack tips since it is well suited to the J-integral calculation.

For one dimension, an additive decomposition of strains under a small strain assumption is given as:

$$\varepsilon = \varepsilon^e + \varepsilon^p = \frac{\sigma}{E} + \alpha \left(\frac{\sigma_y}{E} \right) \left(\frac{\sigma}{\sigma_y} \right)^n \quad (29)$$

where α and n are material parameters, σ is the stress in one dimension, ε is the total strain in one dimension, and E is Young's modulus.

Extending the formula to three dimensions, the strain components can be expressed as:

$$\varepsilon_{ij} = \varepsilon_{ij}^e + \varepsilon_{ij}^p = \overbrace{\frac{\sigma'_{ij}}{2G} + \frac{\sigma_{kk} \delta_{ij}}{9K}}^{\text{elastic}} + \overbrace{\frac{3}{2} \alpha \left(\frac{\sigma_y}{E} \right) \left(\frac{\sigma^{eq}}{\sigma_y} \right)^n \frac{\sigma'_{ij}}{\sigma^{eq}}}^{\text{plastic}} \quad (30)$$

where $\sigma^{eq} = \sqrt{\frac{3}{2} \sigma'_{ij} \sigma'_{ij}}$ is the equivalent stress.

The plastic flow is assumed to be associative and is governed by the von Mises yield criterion.

Under monotonic loading, the total plastic strain can be written as:

$$\varepsilon_{ij}^p = \frac{3}{2} e^p \frac{\sigma'_{ij}}{\sigma^{eq}} \quad (31)$$

where $e^p = \alpha \left(\frac{\sigma_y}{E} \right) \left(\frac{\sigma^{eq}}{\sigma_y} \right)^n$ is the total equivalent plastic strain.

Inverting the plastic strain expression gives:

$$\sigma^{eq} = \sigma_y \left(\frac{E}{\alpha \sigma_y} \right)^m (e^p)^m \quad (32)$$

with $m = \frac{1}{n}$ defining the work hardening exponent.

To set the material parameters, use the commands:

```
pdbSetDouble <material> Mechanics Deformation.Plasticity.sigmay <n>
pdbSetDouble <material> Mechanics Deformation.Plasticity.n <n>
pdbSetDouble <material> Mechanics Deformation.Plasticity.alpha <n>
```

To switch on the deformation plasticity model, use:

```
pdbSet <material> Mechanics Plasticity.Model Deformation
```

The deformation plasticity equations do not require any integration due to total stresses and strains. However, the nonlinear expressions require Newton iterations to achieve the equilibrium state for each loading step. At the end of each iteration, a check on the satisfaction of convergence criteria is made. More Newton iterations are performed until all the convergence criteria are satisfied within the specified tolerance or until the maximum number of iterations is reached. See [Time-Step Control for Mechanics on page 338](#) for details on convergence criteria and time-stepping for mechanics.

NOTE Deformation plasticity must be used only with monotonic loading since the equations are not valid for unloading. This model must be used if the J-integral must be calculated around a crack tip with plastic strains.

Viscoplastic Materials

Materials, such as metals at high temperatures, exhibit rate-dependent plasticity also known as viscoplasticity or creep. There are different ways to model such behavior:

- Anand model
- Power law creep

Anand Model

The Anand model [4][5] is used for rate-dependent plasticity that combines creep and plastic deformation.

Assuming small strains, the strain rates and strains can be decomposed into elastic and viscoplastic components in an additive manner:

$$\begin{aligned}\dot{\epsilon}_{ij} &= \dot{\epsilon}_{ij}^e + \dot{\epsilon}_{ij}^{vp} \\ \epsilon_{ij} &= \epsilon_{ij}^e + \epsilon_{ij}^{vp}\end{aligned}\tag{33}$$

3: Computing Mechanical Stress

Material Models

The elastic strains are evaluated using Hooke's law, while the Anand model is used to evaluate the viscoplastic component. The Anand model assumes that plastic deformation occurs at all values of strain, so instead of a yield function, a constitutive equation is used to relate stresses to viscoplastic strains.

The flow rule for evolution of viscoplastic strains (volume preserving) is assumed to be of the familiar form:

$$\dot{\epsilon}_{ij}^{vp} = \sqrt{\frac{3}{2}} \dot{e}^{vp} \frac{\sigma'_{ij}}{\|\sigma'\|}; \|\sigma'\| = \sqrt{\sigma'_{ij} \sigma'_{ij}} \quad (34)$$

The equivalent viscoplastic strain rate at constant temperature is given by a constitutive equation:

$$\dot{e}^{vp} = f(\|\sigma'\|, s) = A \exp\left(-\frac{Q}{k_B T}\right) \left[\sinh\left(\xi \frac{\|\sigma'\|}{s}\right) \right]^{1/m} \quad (35)$$

where s is the deformation resistance. It is defined in terms of an isotropic hardening function as:

$$\dot{s} = h(s) \dot{e}^{vp} = h_0 \left(1 - \frac{s}{s^*}\right)^a \operatorname{sgn}\left(1 - \frac{s}{s^*}\right) \dot{e}^{vp}; \quad a \geq 1 \quad (36)$$

The saturation value of deformation resistance at a given temperature and strain rate is expressed as:

$$s^* = \tilde{s} \left[\frac{\dot{e}^{vp}}{A} \exp\left(\frac{Q}{k_B T}\right) \right]^n \quad (37)$$

In the above formulation:

- A is a pre-exponential factor (1/s).
- Q is the activation energy in eV.
- k_B is the Boltzmann constant (8.617383e-5 eV/K).
- T is the absolute temperature in kelvin.
- ξ is the stress multiplier (unitless).
- m is the strain rate sensitivity (unitless).
- h_0 is the constant of athermal hardening or softening in dyn/cm^2 .
- a is the exponent of athermal hardening or softening (unitless).
- \tilde{s} is the coefficient for the saturation value of deformation resistance in dyn/cm^2 .
- n is the exponent for the saturation value of deformation resistance (unitless).

Values for the material parameters A , Q , ξ , m , h_0 , a , \tilde{s} , n and the initial value for deformation resistance s_0 are obtained by fitting experimental data for stress–strain (obtained from tension or compression tests conducted at various temperatures and strain rates) to the above equations. For details on how to obtain such data, refer to the literature [4][6][7][8].

To set these parameters, use the commands:

```
pdbSetDouble <material> Mechanics Viscoplasticity.A <n>
pdbSetDouble <material> Mechanics Viscoplasticity.Q <n>
pdbSetDouble <material> Mechanics Viscoplasticity.Xi <n>
pdbSetDouble <material> Mechanics Viscoplasticity.m <n>
pdbSetDouble <material> Mechanics Viscoplasticity.h0 <n>
pdbSetDouble <material> Mechanics Viscoplasticity.a <n>
pdbSetDouble <material> Mechanics Viscoplasticity.stilde <n>
pdbSetDouble <material> Mechanics Viscoplasticity.n <n>
pdbSetDouble <material> Mechanics Viscoplasticity.s0 <n>
```

NOTE For other materials, use the long form of the `pdb` commands to set parameter values.

The material `Solder` in the PDB is used to model viscoplastic behavior. The default values for the above parameters for `Solder` material are based on 96.5Sn3.5Ag solder alloy as reported in [6].

To solve the above nonlinear equations, the rate terms are discretized using the backward Euler method, and the resulting algebraic equations are evaluated locally at every integration point using the Newton–Raphson iterative scheme.

To switch on the viscoplastic material model, use the command:

```
pdbSet <material> Mechanics IsViscoPlastic <n>
```

This flag must be switched on during the simulation if viscoplastic deformation exists. The nonlinear nature of the viscoplasticity model also requires Newton iterations to achieve equilibrium of mechanics equations at each loading step. At the end of each iteration, convergence criteria are checked. More iterations are performed until all the convergence criteria are satisfied within the specified tolerance or until the maximum number of iterations is reached. See [Time-Step Control for Mechanics on page 338](#) for details on convergence criteria and time-stepping for mechanics.

NOTE To avoid convergence problems, use small time steps at the beginning of the analysis. You can increase the number of time steps later, during the analysis, if it does not adversely affect the solution.

Power Law Creep

Power law creep models primary (transient) and secondary (steady-state) creep behavior in metals at high temperatures. Depending on the microstructural processes being modeled, power law creep may be formulated as either Bailey–Norton creep or Mukherjee–Bird–Dorn creep.

To switch on the creep material model, use the command:

```
pdbSet <material> Mechanics IsCreep <n>
```

Bailey–Norton Creep

Bailey–Norton creep [9] assumes creep strain to be of the following form:

$$\dot{\epsilon}^{\text{cr}} = A \exp\left(-\frac{Q}{k_B T}\right) \bar{\sigma}^n \tilde{t}^{(m-1)} \quad (38)$$

where, for multiaxial loading:

- $\dot{\epsilon}^{\text{cr}} = \sqrt{\frac{2}{3}} \epsilon_{ij}^{\text{cr}} \dot{\epsilon}_{ij}^{\text{cr}}$ is the equivalent creep strain rate (1/s).
- A is a pre-exponential factor in $(\text{dyn/cm}^2)^n \cdot \text{s}^{-m}$.
- Q is the activation energy in eV.
- k_B is the Boltzmann constant (8.617383e-5 eV/K).
- T is the absolute temperature in kelvin.
- $\bar{\sigma} = \sqrt{\frac{3}{2}} \sigma'_{ij} \sigma'_{ij}$ is the equivalent stress or the von Mises stress in dyn/cm^2 .
- \tilde{t} is the time (different from physical time) in s.
- n and m are exponents (unitless).

The above form is referred to as a time hardening form. A more commonly used form called the strain hardening form is obtained by eliminating the time variable:

$$\dot{\epsilon}^{\text{cr}} = m \left(A \exp\left(-\frac{Q}{k_B T}\right) \bar{\sigma}^n (\epsilon^{\text{cr}})^{(m-1)} \right)^{\frac{1}{m}} \quad (39)$$

The material parameters A , Q , n , and m are obtained by fitting experimental data. To select this creep model and to set these parameters, use the commands:

```
pdbSet <material> Mechanics Creep.Model PowerLawBN
pdbSet <material> Mechanics Creep.A <n>
pdbSet <material> Mechanics Creep.Q <n>
pdbSet <material> Mechanics Creep.n <n>
pdbSet <material> Mechanics Creep.m <n>
```

NOTE For materials other than `Solder`, use the long form of these commands to set parameter values.

The default values for the above parameters have been added to the PDB to the `Solder` material based on the 96.5Sn3.5Ag solder alloy as reported in [10].

Mukherjee–Bird–Dorn Creep

Mukherjee–Bird–Dorn creep [11][12] assumes creep strain to be of the following form:

$$\dot{\epsilon}^{cr} = \frac{AGb}{k_B T} D_0 \exp\left(-\frac{Q}{k_B T}\right) \left(\frac{b}{d}\right)^p \left(\frac{\bar{\sigma}}{G}\right)^n \quad (40)$$

where, for multiaxial loading:

- $\dot{\epsilon}^{cr} = \sqrt{\frac{2}{3}} \dot{\epsilon}_{ij}^{cr}$ is the equivalent creep strain rate (1/s).
- A is a dimensionless constant (unitless).
- D_0 is the frequency factor for diffusion in cm^2/s .
- G is the shear modulus in dyn/cm^2 .
- k_B is the Boltzmann constant ($8.617383\text{e-}5$ eV/K).
- Q is the activation energy in eV.
- T is the absolute temperature in kelvin.
- b is the magnitude of the Burgers vector in cm.
- d is the grain size/diameter in cm.
- $\bar{\sigma} = \sqrt{\frac{3}{2}} \sigma'_{ij}$ is the equivalent stress or the von Mises stress in dyn/cm^2 .
- n and p are exponents (unitless).

The material parameters A , D_0 , b , Q , n , and p are obtained by fitting experimental data. To select this creep model and to set these parameters, use the commands:

```

pdbSetBoolean <material> Mechanics Creep.Model PowerLawMBD
pdbSetDouble <material> Mechanics Creep.A <n>
pdbSetDouble <material> Mechanics Creep.D0 <n>
pdbSetDouble <material> burgVectMod <n>
pdbSetDouble <material> Mechanics Creep.Q <n>
pdbSetDouble <material> Mechanics Creep.n <n>
pdbSetDouble <material> Mechanics Creep.p <n>

```

3: Computing Mechanical Stress

Material Models

The creep model uses the current grain size when the grain growth model is active (see [Grain Growth Model on page 163](#)). The grain size for the grain growth model is set using the command:

```
pdbSetDouble <material> GrainSize <n>
```

When the grain growth model is not active, the grain size must be specified using `GSize` in the `select` command.

Integrating Creep Equations

Under a small strain assumption, strains (and strain rates) can be decomposed additively as:

$$\epsilon_{ij} = \epsilon_{ij}^e + \epsilon_{ij}^p + \epsilon_{ij}^{cr} \quad (41)$$

with creep strains being distinct from plastic strains.

Creep flow is assumed to be volume preserving ($\epsilon_{ij}^{cr} \delta_{ij} = 0$) and is governed by:

$$\dot{\epsilon}_{ij}^{cr} = \frac{3}{2} \dot{\epsilon}^{cr} \left(\frac{\sigma'_{ij}}{\bar{\sigma}} \right) \quad (42)$$

When incremental plasticity is also active, the creep flow rule is modified to account for hardening:

$$\dot{\epsilon}_{ij}^{cr} = \frac{3}{2} \dot{\epsilon}^{cr} \left(\frac{\sigma'_{ij} - q_{ij}}{\bar{\sigma}} \right) \quad (43)$$

where:

- $\bar{\sigma} = \sqrt{\frac{3}{2}(\sigma'_{ij} - q_{ij})(\sigma'_{ij} - q_{ij})}$ is the equivalent stress.
- q_{ij} is the back stress for kinematic hardening.
- Plastic flow equations are solved simultaneously with creep flow.

To solve the creep equations, the rate terms are discretized using the backward Euler method, and the resulting algebraic equations are evaluated locally at every integration point using the Newton–Raphson iterative scheme.

The nonlinear nature of the creep model also requires global Newton iterations to achieve equilibrium of the mechanics equations at each loading step. At the end of each iteration, convergence criteria are checked. More iterations are performed until all the convergence criteria are satisfied within the specified tolerance or until the maximum number of iterations is reached. See [Time-Step Control for Mechanics on page 338](#) for details on convergence criteria and time-stepping for mechanics.

NOTE To avoid convergence problems, use small time steps at the beginning of the analysis. You can increase the number of time steps later, during the analysis, if it does not adversely affect the solution.

Swelling

Swelling refers to volumetric expansion of material. Swelling material behavior is defined by specifying strain rates at various temperatures that are interpolated linearly. To switch on the model, use the command:

```
pdbSetBoolean <material> Mechanics IsSwelling <n>
```

The swelling strain rate data is specified with a double array:

```
pdbSetArray <material> Mechanics SwellingStrainRate
  Temperature { <temp> {<SSR1> <SSR2> <SSR3>}
               <temp> {<SSR1> <SSR2> <SSR3>} ... }
```

where <temp> is the temperature in degree Celsius, and <SSR1>, <SSR2>, and <SSR3> are strain rates in the x-, y-, and z-direction, respectively, in s^{-1} .

Strain rates can be the same (isotropic) or different (anisotropic) in each of the three directions. For cyclic temperature loading, strain rate data must be given for loading (temperature increment) as well as unloading (temperature decrement). For example:

```
pdbSetArray Mold Mechanics SwellingStrainRate Temperature {
  27 {0.0 0.0 0.0}
  77 {0.001 0.001 0.001}
  127 {0.002 0.002 0.002}
  80 {0.0012 0.0012 0.0012}
  25 {0.0 0.0 0.0}
}
```

If strain rate data is not given for unloading, loading data is used for increasing as well as decreasing temperatures.

Since strains are assumed to be small, swelling strain rates are added to other strain rates:

$$\dot{\epsilon}_{ij} = \dot{\epsilon}_{ij}^e + \dot{\epsilon}_{ij}^{sw} \quad (44)$$

and are integrated over time to give total strains:

$$\epsilon_{ij} = \epsilon_{ij}^e + \epsilon_{ij}^{sw} \quad (45)$$

where $\epsilon_{ij}^{sw} = \dot{\epsilon}_{ij}^{sw} = 0$ for $i \neq j$.

For a given material, only one set of strain rate data can be specified for a solve step. If necessary, different strain rate data may be specified for the same material in a subsequent solve step.

Temperature-Dependent Mechanical Properties

The mechanical properties of materials are different at high temperature from those at room temperature. The elastic modulus of typical materials decreases as temperature rises. Some materials show nonnegligible changes of mechanical properties at different temperatures.

The temperature dependency can be applied to:

- The bulk modulus and the shear modulus for isotropic materials.
- C_{11} , C_{12} , and C_{44} for cubic anisotropic elastic materials.
- C_{11} , C_{12} , C_{13} , C_{33} , and C_{44} for hexagonal anisotropic elastic materials.
- The coefficient of thermal expansion for isotropic, cubic anisotropic, and hexagonal anisotropic elastic materials.

Table 4 Temperature-dependent mechanical parameters

Material model	PDB parameters
Isotropic elasticity	BulkModulus, ShearModulus, ThExpCoeff
Cubic anisotropic elasticity	C11, C12, C44, ThExpCoeff
Hexagonal anisotropic elasticity	C11, C12, C13, C33, C44, ThExpCoeff1, ThExpCoeff2, ThExpCoeff3
Other	Refer to the corresponding sections for related parameters and syntax

The temperature dependency for the above-listed mechanical properties can be defined separately. The available options are:

- The linear dependency is defined with the parameter `<parameter>.T1`, for example:
 - Then, the parameter is calculated using $P(T) = P + P.T1(T - 26.85)$, where the unit of temperature is degree Celsius.
- The piecewise linear dependency can be specified with:

```

pdbSetDoubleArray SiliconGermanium Mechanics ShearModulus.TTable
  {<T1> <v1> ... <Tn> <vn>}

```

The unit of temperature is degree Celsius. The linear dependency defined by `<parameter>.T1` is ignored.

The temperature dependency can be switched off with:

```
pdbSet Mechanics Parameter.Interpolation 0
```

NOTE Do not mix new syntax with old syntax (see [Plane Stress Analysis on page 67](#)). Some of the old syntax might exist in the Advanced Calibration script.

Temperature dependency of yield stress for incremental plasticity (see [Incremental Plasticity on page 55](#)) can be modeled in two ways: linear or exponential.

Linear dependency of yield stress on temperature is modeled as:

$$\sigma_y = \sigma_0 \left(1 - \frac{T}{T_0}\right) \quad (46)$$

where σ_0 is the yield stress at zero absolute temperature ($T = 0$), and T_0 is the reference absolute temperature. These variables are specified, respectively, by the commands:

```
pdbSetDouble <material> Mechanics Incremental.Plasticity.SigmaY <n>  
pdbSetDouble <material> Mechanics Incremental.Plasticity.RefTemp <n>
```

Exponential dependency on temperature is modeled through an Arrhenius expression. The prefactor and exponent of the Arrhenius expression are specified, respectively, by the following commands:

```
pdbSetDouble <material> Mechanics Incremental.Plasticity.SigmaY <n>  
pdbSetDouble <material> Mechanics Incremental.Plasticity.ActEnergy <n>
```

NOTE The linear and exponential models are mutually exclusive.

NOTE The PDB parameter `FirstYieldW` is deprecated and is replaced by `Incremental.Plasticity.ActEnergy`.

Plane Stress Analysis

In 2D problems, the elastic models implemented in Sentaurus Interconnect follow the plane strain formulation by default. Under the plane strain assumption:

$$\varepsilon_{zz} = 0; \sigma_{zz} \neq 0 \quad (47)$$

While this is good for structures where the strain in the third direction is very small compared to the cross section, it would give inaccurate results for thin structures.

3: Computing Mechanical Stress

Equations: Global Equilibrium Condition

Thin plate-like structures where one dimension is very small compared to the other two can be modeled under the plane stress assumption:

$$\epsilon_{zz} \neq 0; \sigma_{zz} = 0 \quad (48)$$

The strain ϵ_{zz} is obtained as a function of other strains, for example, for purely elastic structures:

$$\epsilon_{zz} = -\nu(\epsilon_{xx} + \epsilon_{yy}) \quad (49)$$

The plane stress model can be switched on for a particular region using:

```
pdbSetBoolean <material> Mechanics PlaneStress 1
```

NOTE You must switch on the plane stress model in all regions. Combining plane stress and plane strain formulations within a structure by switching on plane stress in only a few regions is *not* advisable.

Equations: Global Equilibrium Condition

The equations for mechanics in Sentaurus Interconnect are the quasistatic equations of force equilibrium.

The strain rate tensor is related to the symmetric component of the velocity gradient and is given by:

$$\dot{\epsilon}_{jk} = \frac{1}{2} \left(\frac{\partial v_j}{\partial x_k} + \frac{\partial v_k}{\partial x_j} \right) \quad (50)$$

Strain is then related to stresses through any of the models defined in [Material Models on page 46](#). For all models, the global equilibrium condition is given by:

$$\sum_k \frac{\partial \sigma_{jk}(\mathbf{v})}{\partial x_k} = 0 \quad (51)$$

The above equations are solved using the finite-element method. The solution is a vector representing the velocity components at each node. These velocities are used to compute the strain and stresses. The stresses and the boundary conditions determine the mechanical state of the system.

NOTE Stress and strain are derivatives of velocity. They are, therefore, computed at one order of accuracy lower than the solution variable. This also means that they are discontinuous across the elements. When visualized, the stress values may appear poorly converged even if the linear solver has converged.

In addition, the quasistatic mechanics equations are elliptic in nature and, therefore, are prone to high levels of shape dependency. This is most frequently seen at structure corners and crack tips. These equations also exhibit a high sensitivity to the mesh modification algorithms at these corners or crack tips.

NOTE At sharp corners or crack tips, the mechanics equations have a singularity. Therefore, it is not possible to discretize at a corner correctly using regular types of element.

Second-Order Finite-Element Analysis

Second-order finite elements are available for triangles and tetrahedra to perform mechanical stress analysis for linear elastic (isotropic and anisotropic), viscoelastic, elastoplastic, viscoplastic, and creep material models. In addition, they can be used for J-integral calculations and the cohesive zone model.

Figure 8 shows the first-order or linear elements for triangles and tetrahedra. These elements have nodal degrees of freedom, and the figures show the associated shape functions.

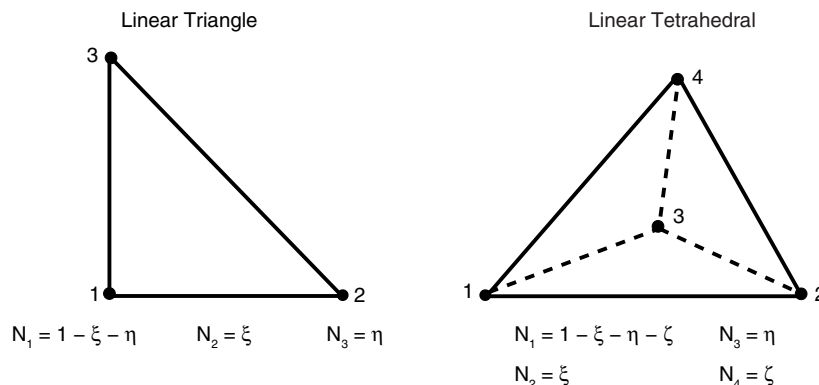


Figure 8 Shape functions and locations of degrees of freedom for linear elements

Second-order or quadratic elements use quadratic shape functions to provide a higher degree of accuracy and, in certain cases, they help avoid locking and convergence issues.

3: Computing Mechanical Stress

Second-Order Finite-Element Analysis

Second-order finite-element analysis is switched on using:

```
pdbSet Mechanics FiniteElementOrder 2
```

If a second-order simulation is performed, second-order elements are used throughout the entire structure and cannot be used regionwise. In addition, second-order finite-element analysis is more memory intensive. Given the higher accuracy of these elements, it is suggested to use fewer mesh points compared to linear elements.

An additional field is available for visualizing second-order elements in Sentaurus Visual:

- `Displacement2` plots the displacement calculated over the nodes and edge centers. This field is quadratic in nature.

NOTE Second-order elements may give inaccurate solutions or may cause convergence difficulties with the cohesive zone model when simulating a uniform pressure distribution on crack surfaces.

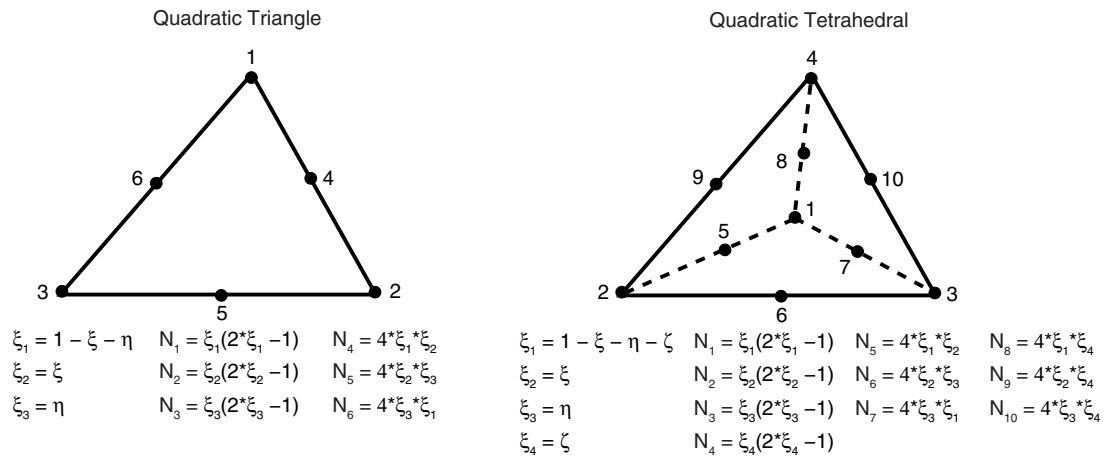


Figure 9 Shape functions and locations of degree of freedom for quadratic elements

Mechanics Simulations on Mixed or Hybrid Meshes

Mechanics simulations can be performed on hybrid meshes including triangular and quadrilateral (axis-aligned) elements in two dimensions, and tetrahedral, brick, pyramid, and prism elements in three dimensions.

The hybrid mesh can be obtained in one of the following ways:

- Use the `line` command to generate axis-aligned meshes.
- Read in a TDR file that contains hybrid elements.

To read in a hybrid mesh from a TDR file, specify the option `mixed.mesh` in the `init` command:

```
init tdr=<c> mixed.mesh
```

Hybrid mesh elements are only available for first-order mechanics simulations. Process steps such as etching and deposition are allowed. However, only mechanics simulations can be performed on these meshes and the results stored. The hybrid meshes support all material models as well as fracture mechanics (including J-integral calculations and cohesive zone modeling).

NOTE Not all postprocessing commands (for example, `slice` and `interface`) are supported in mixed-mesh mode. Data extraction using the `interpolate` command and data integration using the `integrate` command are supported only for nodal data fields. However, you can convert mixed-mesh structures to triangular or tetrahedral structures using either the `!mixed.mesh` option of the `init` command or external tools such as Sentaurus Mesh.

Mechanics Simulations With Composite Solid Shell Element

A composite solid shell element (CSSE) is a solid finite element that exhibits shell behavior in the thickness direction. The CSSE falls between thin shell and conventional solid elements and has the same degree-of-freedom configuration as solid elements. Conventional solid elements lock when subject to bending and require a very fine mesh for thin layers. CSSEs adopt a special constitutive formulation to reduce transverse shear locking, membrane locking, and Poisson thickness locking. It is useful to model shell-like behavior such as thin films subject to bending and multilayered thin films.

The CSSE is implemented with a first-order linear 3D brick element and supports isotropic and orthotropic elastic material models. An axis-aligned mesh with brick elements can be generated using the `line` command and the `mixed.mesh` option of the `init` command (see [Mechanics Simulations on Mixed or Hybrid Meshes on page 70](#)).

NOTE Typical loading types such as displacement loading and force loading are supported. Other stress-causing mechanisms and models except thermal mismatch are not supported.

The CSSE material is defined with the following command and is used to define the material of a CSSE region:

```
mater name=<c> add new.like=CSSEMaterial
```

3: Computing Mechanical Stress

Mechanics Simulations With Composite Solid Shell Element

For example:

```
mater name=CSSEMat add new.like=CSSEMaterial
```

The material behavior of the CSSE is isotropic elastic by default. If orthotropic elastic behavior is used, set:

```
pdbSet <CSSE_Material_Name> Mechanics Orthotropic 1
```

For a CSSE with a single layer, specify the layer material with a relative thickness of 1.0 by setting:

```
pdbSet <CSSE_Material_Name> Mechanics CSSELayers {<material> 1.0}
```

In this case, a CSSE element behaves the same as a typical solid shell element.

In addition, layered regions can be simulated with a single CSSE in the thickness direction through the thickness integration of the layered wall. Up to 20 layers are supported. The material and relative thickness of each layer must be defined layer-wise by setting:

```
pdbSet <CSSE_Material_Name> Mechanics CSSELayers {<material1> <thickness1>
<material2> <thickness2>
...}
```

If the material within each layer is orthotropic, an in-plane rotation angle can be specified with:

```
pdbSetDouble <material1> Mechanics Ortho.InPlane.Rot.Angle.Degree <angle>
```

For a CSSE with multiple layers, the geometry and data fields for layer-by-layer visualization are saved to a TDR file using:

```
struct tdr=<c> csse
```

With the `csse` option, each CSSE is split into multiple layers for visualization, and data fields from layered CSSEs are interpolated to each split layer. In the split CSSE case, each layer can be visualized individually.

Boundary Conditions

Equations for stress equilibrium require boundary conditions to define the system completely.

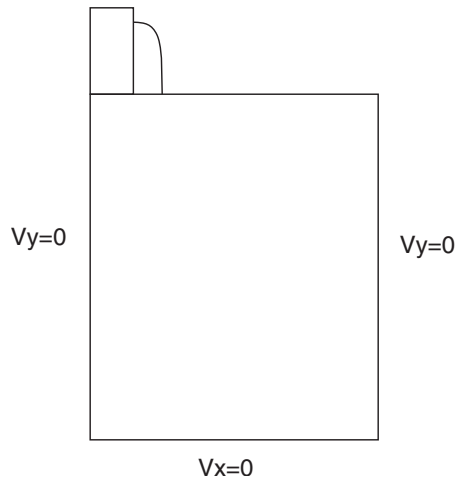


Figure 10 Default mechanics boundary conditions in the unified coordinate system

The default boundary conditions are zero velocities in the direction perpendicular to the boundary planes. Since velocities are set to fixed values along the boundaries, these boundary conditions are referred to as Dirichlet boundary conditions in directions perpendicular to boundary planes. The `HomNeumann` boundary condition is used when the plane must be free.

Sentaurus Interconnect provides a general way to reset the default boundary conditions and specify boundary conditions for stress analysis using the `stressdata` command:

```
stressdata bc.location= Left | Right | Front | Back | Bottom | <c>
          bc.value= {dx=<n> | dy=<n> | dz=<n> | fx=<n> | fy=<n> | fz=<n> |
                    sx=<n> | sy=<n> | sz=<n>}
```

where:

- `dx`, `dy`, and `dz` specify the displacement rates (default unit: cm/s).
- `fx`, `fy`, and `fz` specify the total forces (default unit: dyne).
- `sx`, `sy`, and `sz` specify the distributed forces (default unit: dyn/cm²).

All the displacement rates, total forces, and distributed forces are applied to the area defined by `bc.location`, where `Left`, `Right`, `Front`, `Back`, and `Bottom` refer to the outer boundary surfaces of the simulation domain. If `bc.location` specifies a contact name (`<c>`), the corresponding contact must exist.

3: Computing Mechanical Stress

Boundary Conditions

Sentaurus Interconnect checks whether the following fundamental boundary condition rules are violated in user-defined settings:

- For any degree of freedom, only one boundary condition can be specified.
- At least at one point, the deformation along any coordinate system direction must be fixed through displacement boundary conditions, so that rigid body movement will not become a valid solution.

NOTE To obtain reaction forces at nodes with Dirichlet boundary conditions, specify:

```
pdbSet Boolean Mechanics Save.Reaction.Force 1
```

NOTE Any boundary condition defined with the `stressdata` command invalidates the default boundary conditions. If no boundary condition is defined, free boundary condition is applied by default.

Dirichlet boundary conditions are imposed using the penalty method, by default. To adjust the penalty factor, use the command:

```
pdbSet Mechanics Boundary.Penalty.Factor <n>
```

The default penalty factor is 1.0×10^{12} . The larger this factor, the more accurate the enforcement of Dirichlet boundary conditions. However, using an extremely large penalty factor could lead to an ill-conditioned matrix and, therefore, could slow down the linear equation solver.

Alternatively, you can use the matrix reduction method to impose Dirichlet boundary conditions. To choose the penalty method or matrix reduction method, use the command:

```
pdbSet Mechanics Boundary.Method.Type <model>
```

where `<model>` is `Penalty` or `MatrixReduction`.

NOTE To ensure the structure is bounded by a perfect rectangle, the displacements computed by these general boundary conditions are not applied to the structure. However, they evaluate the stresses correctly. This assumption is consistent with the small deformation assumption within each mechanics time step.

Example: Applying Boundary Conditions

This 2D example simulates silicon covered with oxide, with the right side free to move:

```
line x loc= -0.02 tag= e spacing= 0.005
line x loc= 0      tag= a spacing= 0.005
line x loc= 0.2   tag= b spacing= 0.05

line y loc= 0 tag= c spacing= 0.05
line y loc= 2 tag= d spacing= 0.05

region silicon xlo= a xhi= b ylo= c yhi= d
region oxide xlo= e xhi= a ylo= c yhi= d

init !DelayFullD

pdbSetDouble Mechanics RefThExpCoeff 0
stressdata bc.location= Bottom bc.value= { dx=0 }
stressdata bc.location= Left bc.value= { dy=0 }

pdbSet Oxide Mechanics Viscosity0 1e40
pdbSet Oxide Mechanics ViscosityW 0
mode mechanics

temp_ramp name= tr1 temperature= 600 ramprate= 30<K/min> time= 10<min>
solve temp.ramp= tr1

struct tdr= rampup

solve time= 10 temperature= 900

struct tdr= postout
```

Deprecated Syntax

You can select various boundary conditions using:

```
pdbSet Mechanics <side> BoundaryCondition <model>
```

where:

- <side> is Left, Right, Front, or Back.
- <model> is HomNeumann or Dirichlet.

3: Computing Mechanical Stress

Boundary Conditions

The HomNeumann boundary condition is used when the plane must be free and implies a zero normal stress (shown in Figure 11).

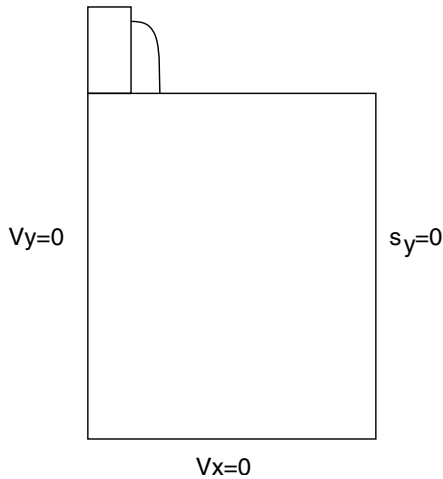


Figure 11 HomNeumann boundary condition on right boundary plane

For example, if you want to set the right plane to be free, use the command:

```
pdbSet Mechanics Right BoundaryCondition HomNeumann
```

NOTE Any boundary condition or loading condition defined with the `stressdata` command invalidates all boundary conditions defined with `pdbSet Mechanics <side> ...`, including the default ones.

Variable Boundary Condition

To define values for displacement rates, total forces, and distributed forces that depend on temperature or time, use the command:

```
stressdata bc.location= Left | Right | Front | Back | Bottom | <c>  
bc.proc= {dxP=<c> | dyP=<c> | dzP=<c> | fxP=<c> | fyP=<c> | fzP=<c> |  
sxP=<c> | syP=<c> | szP=<c>}
```

where:

- dxP , dyP , and dzP specify the procedure names for the displacement rates.
- fxP , fyP , and fzP specify the procedure names for the total forces.
- sxP , syP , and szP specify the procedure names for the distributed forces.

An example of temperature-dependent boundary conditions is:

```
stressdata bc.location= Right bc.proc= {dxP=LookUpTable}
```

```
fproc LookUpTable { args } {
  set CurrTemp [lindex $args 0]           ;# current temperature in C
  set PrevTemp [lindex $args 1]          ;# previous temperature in C
  set Time [lindex $args 2]              ;# previous time in s
  set TimeStep [lindex $args 3]          ;# current time step in s
  set CurrTime [expr $Time + $TimeStep]  ;# current time in s
  set PrevTime $Time
  set dy 0
  if { $CurrTemp < 27 }
  {
    set dy 1.0e-5                         ;# velocity in cm/s
  } elseif { $CurrTemp > 27 && $CurrTemp < 127 } {
    set dy 1.0e-6                         ;# velocity in cm/s
  } else {
    set dy 0                               ;# velocity in cm/s
  }
  return $dy
}
```

Pressure Boundary Condition

The pressure boundary condition is used to apply uniform pressure on the exterior boundary. The direction of the loading depends on the normal of the exterior surface. To apply the pressure boundary condition, use the `stressdata` command, for example:

```
stressdata bc.location= Left | Right | Front | Back | Bottom | <c>
bc.value= {pressure=<n>}
```

Advanced Dirichlet Boundary Condition

A more advanced Dirichlet-type boundary condition can be defined that specifies both the translational and rotational velocities on the boundaries. It is defined using the command:

```
stressdata bc.location= Left | Right | Front | Back | Bottom | <c>
bc.rotation.axis= {xa=<n> | ya=<n> | za=<n>}
bc.value= {dx=<n> | dy=<n> | dz=<n> | rx=<n> | ry=<n> | rz=<n>}
```

where:

- `dx`, `dy`, and `dz` specify the displacement rates (default unit: cm/s).
- `rx`, `ry`, and `rz` specify the rotational velocities (default unit: rad/s).
- `xa`, `ya`, and `za` specify the coordinates of the point around which the rotation occurs (default unit: cm).

3: Computing Mechanical Stress

Boundary Conditions

NOTE To obtain reaction forces and moments at an external point with Dirichlet boundary conditions, specify:

```
pdbSet Boolean Mechanics Save.Reaction.Force 1
```

Advanced Neumann Boundary Condition

A Neumann-type boundary condition can be defined at an external point constrained to a face or contact. It is specified using the command:

```
stressdata bc.location= Left | Right | Front | Back | Bottom | <c>  
  bc.rotation.axis= {xa=<c> | ya=<c> | za=<c>}  
  bc.value= {fx=<n> | fy=<n> | fz=<n> | mx=<n> | my=<n> | mz=<n>}
```

where:

- `fx`, `fy`, and `fz` specify forces (default unit: dyne).
- `mx`, `my`, and `mz` specify moments (default unit: dyn cm).
- `xa`, `ya`, and `za` specify the coordinates of the external point about which the moments are applied (default unit: cm).

Point-Force Boundary Condition

Point-force boundary condition allows force to be applied directly on a bulk node. This boundary condition can be defined using the `stressdata` command:

```
stressdata bc.location=<c> bc.value= {pfx=<n> pfy=<n> pfz=<n>}  
  point.coord= {<n> <n> <n>}
```

where:

- `bc.location` specifies the name of the defined point-force boundary condition.
- `bc.value` specifies the magnitude of the force to be applied in each direction (default unit: dyne).
- `point.coord` is the coordinate of the bulk node (default unit: μm).

The point force is applied to the nearest node to the coordinate specified. The snap tolerance is default to 1 nm and can be changed with:

```
pdbSet Mechanics Point.Snap.Dist <n>
```

If the nearest node is farther than the snap tolerance, the point force is distributed to the two end nodes of the edge containing the point (in two dimensions) or three corner nodes of the face containing the point (in three dimensions). To switch from distributing the point force (default)

to snapping it onto the nearest node when the node is farther than the snap tolerance (nondefault), use:

```
pdbSet Mechanics Point.Force.Distribute 0
```

Unspecified coordinate or force components default to zero. If a new point-force boundary condition is defined with an existing name, the old point-force boundary condition will be replaced.

Point-Displacement Rate Boundary Condition

The point-displacement rate boundary condition allows velocity, that is, displacement rate, to be applied directly to a bulk node. This boundary condition can be defined using the `stressdata` command:

```
stressdata bc.location=<c> bc.value= {pdx=<n> pdy=<n> pdz=<n>}  
point.coord= {<n> <n> <n>}
```

where:

- `bc.location` specifies the name of the defined point-displacement rate boundary condition.
- `bc.value` specifies the magnitude of the displacement rate to be applied in each direction (default unit: cm/s).
- `point.coord` is the coordinate of the bulk node (default unit: μm).

The displacement rate is applied to the nearest node to the coordinate specified. Similar to the point-force boundary condition, the snap tolerance defaults to 1 nm and can be changed with:

```
pdbSet Mechanics Point.Snap.Dist <n>
```

If the nearest node is farther than the snap tolerance, the point displacement rate is distributed to the two end nodes of the edge containing the point (in two dimensions) or three corner nodes of the face containing the point (in three dimensions). To switch from distributing the point displacement rate (default) to snapping it onto the nearest node when the node is farther than the snap tolerance (nondefault), use:

```
pdbSet Mechanics Point.DispRate.Distribute 0
```

If no coordinate is specified, the default value is zero. If the displacement rate component is not specified, it is considered to be unconstrained. If a new point-displacement rate boundary condition is defined with an existing name, the old point-displacement rate boundary condition will be replaced.

Periodic Boundary Condition

The periodic boundary condition is used for structures with a periodically repeating pattern. This condition is used on periodic structures with assigned master and slave boundaries. The slave boundary has the same deformation profile as the master boundary.

In [Figure 12](#), the left and right boundaries are bound together by the periodic boundary condition.

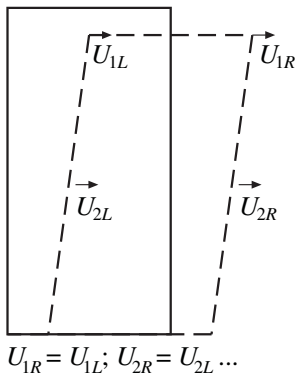


Figure 12 Periodic boundary condition

To apply the periodic boundary condition to the outer bounding surfaces, use the command:

```
pdbSet Mechanics <Left | Right | Front | Back> Periodic 1
```

If this command is specified on a sidewall, the opposite sidewall is defined automatically as a periodic boundary. Conflicts of boundary condition definitions are checked on all sidewalls. For example, to apply a periodic boundary condition on the left and right sidewalls, use one of the following commands:

```
pdbSet Mechanics Left Periodic 1  
pdbSet Mechanics Right Periodic 1
```

Both the periodic and coupling boundary conditions are implemented using the penalty method. To adjust the penalty factor, use the command:

```
pdbSet Mechanics Constraint.Penalty.Factor <n>
```

The default penalty factor is 1.0×10^{13} . The larger this factor, the more accurately the periodic or coupling boundary conditions will be enforced. Using an extremely large penalty factor could lead to an ill-conditioned matrix and, therefore, slow down or even fail the linear equation solver.

NOTE If you choose to apply periodic boundary conditions, all other boundary conditions defined through the old `pdbSet` method will be ignored and must be redefined using the `stressdata` command.

Coupling Boundary Condition

Depending on the structure and loading, certain degrees of freedom must be coupled. One particularly useful case is the coupling boundary condition. The coupling boundary condition on a sidewall enforces an equal displacement component at all nodes of this boundary, as shown in [Figure 13](#).

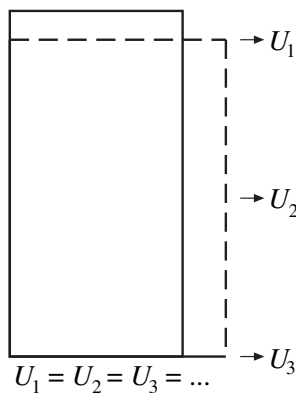


Figure 13 Coupling boundary condition

To switch on the coupling boundary condition, use:

```
pdbSet Mechanics <Left | Right | Back | Front> <Cx | Cy | Cz> 1
```

For example, to apply the coupling boundary condition for the y-component of displacement on the right sidewall, use the command:

```
pdbSet Mechanics Right Cy 1
```

The coupling boundary condition also is implemented through the penalty method.

NOTE If you choose to apply coupling boundary conditions, all other boundary conditions defined through the old `pdbSet` method will be ignored and must be redefined using `stressdata` command.

Time-Step Control

It may be necessary to use time step control when viscous or viscoelastic materials are present in the structure. Usually, when other PDEs are solved, the time steps will be sufficiently limited by the solver. If PDEs are not solved or for materials with a low viscosity, the time step control for the Maxwell model can be switched on with:

```
pdbSet Mechanics Visc.Step.Control 1
```

Time-stepping can be controlled with both the displacement increment and the relative relaxation time. To switch on these two options, use the following commands:

```
pdbSet Mechanics Visc.Step.Limit.Disp 1  
pdbSet Mechanics Visc.Step.Limit.ScaleT 1
```

To see other control parameters, launch the PDB Browser, and see the section **Mechanics > Visc.Step**.

For the standard linear solid model for viscoelasticity, time-step control is activated automatically (see [Time-Step Control for Mechanics on page 338](#)).

Stress-Causing Mechanisms

Every mechanical system needs a set of stress-driving mechanisms to reach a stressed state. The stress-inducing mechanisms in Sentaurus Interconnect are described here.

Densification

A typical densification process uses thermal heating to increase the density of a porous material. As the material density increases, its volume shrinks and the volume shrinkage generates stresses.

The densification-induced stress computation is switched on using the `density.increase` parameter in the `solve` command or the `temp_ramp` command, such as:

```
solve temperature= 1000<C> time= 30<min> \  
  density.increase= {<regionName>=<n> | <material>=<n>}  
  
temp_ramp name= dens time= 1 temperature= 1000 density.increase= {oxide=0.02}  
solve temp.ramp= dens
```


The total amount of density increase can be specified per material or per region for a given `solve` (or `temp_ramp`) step as shown above. A proportional amount of density increase is applied during each time step of the densification process.

The densification operation can be performed for all existing materials, as well as new materials defined using the `mater` command:

```
mater add name= TEOS new.like= oxide
solve time= 1 temperature= 1000 density.increase= {TEOS= 0.03}
```

For densification processes involving large amounts of volume shrinkage, the material boundaries and meshes can be updated using the following settings:

```
pdbSet Grid Modify.Mesh 1
pdbSetDouble TEOS Grid MinimumVelocity 0
```

For a complete densification process that has distinguished density changes, multiple `solve` steps can be used with different density increases for each segment of the process.

Selectively Switching Off Grid Movement

The parameter `MinimumVelocity` can be used to selectively switch off point or interface movement. This can be useful, for example, when a mechanics simulation computes a small amount of boundary movement that is either unwanted or could cause element quality to suffer in the vicinity, and the approximation of no movement is acceptable. In general, the command is:

```
pdbSet <material> Grid MinimumVelocity <speed>
```

If `<material>` is a bulk material (no underscore), the parameter applies to bulk points. If the speed of the bulk points is less than `<speed>` (in cm/s), Sentaurus Interconnect truncates the speed to zero. This truncation is applied to material `Silicon` by default.

On the other hand, if `<material>` is an interface material (with an underscore, such as `PolySilicon_Silicon`), the parameter only applies to points on that interface. This truncation is applied to material `Silicon` by default.

NOTE The moving mesh operations can become unstable for values of `MinimumVelocity` that are neither very large nor zero. Very large values stop all motion, and 0 allows all motion.

Grain Growth

Grain growth means elimination of the grain boundary, which can lead to the evolution of stress. Using the model of spherical grains, the polycrystalline metal is dilated relative to the previous state with smaller grains by [12]:

$$\Delta\varepsilon_{ij} = \text{GISGeomFactor} \cdot \text{GB.Width} \cdot \left(\frac{1}{L_g} - \frac{1}{L_{g0}} \right) \quad (52)$$

where the parameters `GISGeomFactor` and `GB.Width` are defined for a material.

The grain growth-induced stress model applies only to the grain growth model when the parameter `GIS` is switched on:

```
pdbSet Mechanics GIS 1
```

Thermal Mismatch

Temperature changes during the process described by the `temp_ramp` command or the keyword `ramprate` in the `solve` command lead to stress in the structure caused by the different coefficients of thermal expansion of the relevant materials.

When necessary, the stress computation can be switched off using the `stress.relax` flag:

```
solve temperature= 1000<C> time= 30<min> !stress.relax
```

NOTE If viscous or viscoelastic materials are present in the structure, the stress distribution may change even without a change in the temperature due to viscoelastic relaxation.

The coefficient of thermal expansion for certain materials can be found in the parameter database as follows:

```
pdbGet <material> Mechanics ThExpCoeff
```

Thermal expansion only affects the dilatational component of the constitutive equation. For isotropic elasticity, a small temperature change leads to a change in the dilatational stress component according to:

$$\Delta\sigma_{kk} = -3K\alpha_{\text{rel},kk}\Delta T \quad (53)$$

where K is the bulk modulus at the current temperature. The change in the temperature is described by ΔT and $\alpha_{\text{rel}} = \alpha_{\text{mat}} - \alpha_{\text{subs}}$ is the relative coefficient of thermal expansion of a certain material with respect to the coefficient of thermal expansion of the substrate. The

integration of thermal mismatch strain uses the midpoint rule. For anisotropic elasticity, the general tensor expression is used.

For hexagonal anisotropic materials, the coefficient of thermal expansion tensor has a symmetry plane and can be defined with:

```
pdbSet <material> Mechanics ThExpCoeff1 <n>  
pdbSet <material> Mechanics ThExpCoeff2 <n>  
pdbSet <material> Mechanics ThExpCoeff3 <n>
```

The first two coefficients of thermal expansion must be equal for hexagonal anisotropic materials.

In certain examples, like bending, you may want to use absolute coefficients of thermal expansion instead of relative coefficients of thermal expansion. This can be achieved by setting the parameter `RefThExpCoeff` as follows:

```
pdbSetDouble Mechanics RefThExpCoeff 0.
```

All coefficients of thermal expansion are computed with respect to the substrate. This reference value is changed by setting a certain region as the substrate and resetting the coefficient of thermal expansion. A region can be tagged as the substrate in different ways:

- Use the `substrate` argument when defining regions with the `region` command before the `init` command.
- If a saved structure is being loaded into Sentaurus Interconnect, a region is tagged as the substrate with the command:

```
region name=<c> substrate
```

The reference coefficient of thermal expansion can be directly set with:

```
pdbSetDouble Mechanics RefThExpCoeff <n>
```

This command overwrites the reference coefficient of thermal expansion setting from the substrate.

Edge Dislocation

The existence of crystal lattice defects, such as dislocation, affects the channel stress state. The impact of edge dislocation is included by superposing the dislocation-induced stress field for an isotropic infinite medium from elasticity theory. Each edge dislocation can be defined with:

```
stressdata <material> | region=<c>  
  apply.dislocation dislocation.origin= {<n> <n> <n>}  
  para.orient= {<n> <n> <n>} perp.orient= {<n> <n> <n>} stress.relax
```

3: Computing Mechanical Stress

Stress-Causing Mechanisms

where:

- `dislocation.origin` is the location of the dislocation core.
- `para.orient` specifies the direction of the edge dislocation or the direction of the half plane.
- `perp.orient` is the Burgers vector in the perpendicular direction to the half plane. Here, the magnitude of `perp.orient` is the slip distance.
- `stress.relax` switches on the relaxation of stresses after superposing the dislocation-induced stress field.

You must supply either a region name or a material name. If `region` is specified, the stress field is superposed to this region. If a material is specified, the stress field is applied to all regions of crystalline material. To save the edge dislocation geometry information to a TDR file for visualization, specify `saveTDR` with the edge dislocation definition. The edge dislocation is displayed by the half plane of extra atoms. To save the half plane of the missing atoms, use:

```
pdbSet Mechanics Display.Missing.Plane 1
```

The Burgers vector can be stored in the vector data field `BurgersVector` by setting:

```
pdbSet Mechanics Display.Burgers.Vector 1
```

Singularity exists in the analytic solution at the dislocation core. Without using a nonlinear atomistic theory, the stresses in the core region within a few magnitudes of the Burgers vector to the dislocation core are smoothed away. The factor for this core radius can be defined with:

```
pdbSet Mechanics Dislocation.Coresize.Factor 2.0
```

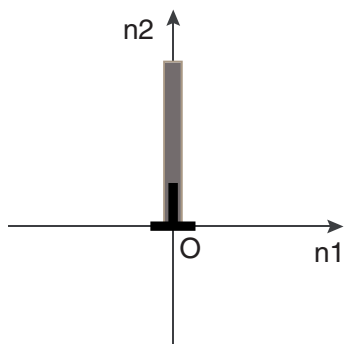


Figure 14 Edge dislocation located at the origin O; n_1 is the Burgers vector and n_2 is the direction of the half plane

A prototype model for positioning the edge dislocations is available by minimizing the elastic strain energy [13]. The stress field from each edge dislocation is superposed. The elastic strain energy is determined after force equilibrium with edge dislocations at their initial locations.

The initial location of edge dislocation serves as the initial guess and can be defined by:

```
stressdata region=<c> !apply.dislocation dislocation.origin= {<n> <n> <n>}  
para.orient= {<n> <n> <n>} perp.orient= {<n> <n> <n>}
```

where:

- `dislocation.origin` is the initial location of the edge dislocation.
- `!apply.dislocation` is specified to delay applying the dislocation-induced stress field.

Multiple edge dislocations with different Burgers vectors can be defined separately with this syntax.

When all the edge dislocations for minimizing the elastic strain energy are specified, you can start the optimization with the command:

```
stressdata origin.max= {<n> <n> <n>} origin.min= {<n> <n> <n>}  
optimize.dislocation
```

where `origin.max` and `origin.min` define the range of dislocation positions in the specified region. Some additional parameters for optimization convergence control also can be defined in this command (see [stressdata on page 634](#)).

The movement of edge dislocations depends on the gradient of the total elastic strain energy computed from a discrete integral over all elements. The target of the optimization is set to -5 multiplied by the absolute value of the starting elastic strain energy. This factor can be changed with:

```
pdbSetDouble Mechanics Energy.Optimization.Factor <n>
```

The coordinates of the edge dislocations after optimization are returned in a Tcl list formatted as:

- `<x1> <y1> <x2> <y2> . . .` for two dimensions.
- `<x1> <y1> <z1> <x2> <y2> <z2> . . .` for three dimensions.

The final stress state remains the same as before the edge dislocations are introduced. The edge dislocations may stop at the local minimum where the elastic strain energy has not reached the global minimum. In such a case, a new optimization step must be started with the initial guess of the edge dislocation positions adjusted based on the previous optimization result. It is also helpful to refine the mesh.

Resolved shear stress can be used to predict the dislocation nucleation. Slip in a grain occurs when the resolved shear stress exceeds the critical value.

3: Computing Mechanical Stress

Stress-Causing Mechanisms

The applied stress is resolved along a slip plane in the slip direction where the normal of the slip plane and the slip direction are defined with:

```
stressdata <material> | region=<c> slip.plane.normal= {<n> <n> <n>}
    slip.direction= {<n> <n> <n>}
```

Both vectors are defined with Miller indices in crystal lattices. The actual calculation is switched on with:

```
stressdata resolved.shear.stress
```

This command will output a scalar field `ResolvedShearStress`. This calculation only applies to regions or materials with both the slip plane normal and the slip direction defined.

Inverse Piezoelectric Effect

The effect of the electric field on stresses can be taken into account in simulations using the inverse piezoelectric effect. To include this effect in a stress simulation, the fields `ElectricField` and `PrevElectricField` are required to be used with only one solve step. These fields may be stored from a previous simulation or stored during the current simulation using the `select` command. Note that in Sentaurus Interconnect the `ElectricField` is obtained from the `Potential` solution.

The contribution of the inverse piezoelectric effect is given by:

$$\sigma_j^{\text{piezo}} = -e_{ij}(E_i^{\text{current}} - E_i^{\text{prev}}) \quad (54)$$

where:

- σ_j^{piezo} are the components of stress due to the inverse piezoelectric effect ($j = 1$ to 6 corresponds to the `xx`, `yy`, `zz`, `xy`, `yz`, and `xz` components of the stress tensor, respectively).
- e_{ij} are the components of the piezoelectric tensor ($i=1, 2, \text{ or } 3$).
- E_i^{current} and E_i^{prev} are the components of the current electric field and the previous electric field, respectively.

To include the effect of electric field on stresses, switch on the following parameter:

```
pdbSet Mechanics InversePiezoEffect 1
```

In addition, the piezoelectric tensor parameters must be set regionwise as:

```
pdbSetDouble <region> Mechanics PiezoElectricTensor<ij> <double>
```

where $i=1, 2, \text{ or } 3$ and $j=1, 2, 3, 4, 5, \text{ or } 6$. These piezoelectric tensor components are defined in the unified coordinate system (UCS). The default value and unit for the piezoelectric tensor components is $0 \text{ dyn}/(\text{V}\cdot\text{cm})$.

NOTE The inverse piezoelectric effect is only included for elastic and viscoelastic materials. It is not available for nonlinear material models.

Effective Surface Tension and Pressure due to Fluid

Residual fluid can cause pattern deformation by unbalanced capillary forces and surface tension. These effects are treated as effective surface tension with the line force boundary condition and effective pressure, respectively.

The line force boundary condition is applied at the gas–fluid–solid material interface. The fluid region must be defined with a fluid material derived from the material template `Fluid`, for example:

```
mater add name=<FluidMaterial> new.like=Fluid
```

The surface tension magnitude and the contact angle can be defined for each fluid material on a list of solid wall materials. Both parameters must be defined for each fluid material (with default units of dyn/cm and degree, respectively). For example:

```
pdbSetDoubleArray <FluidMaterial> SurfaceTension {<mat1> <n> <mat2> <n>}  
pdbSetDoubleArray <FluidMaterial> ContactAngle  
  {<mat1> <angle1> <mat2> <angle2>}
```

A vector data field `TensionForce` is created to save the nodal force due to effective surface tension.

The contact angle specifies the angle between the fluid surface tangent and the solid wall. This angle is used to determine the direction of the effective surface tension loading. The fluid region is not required to match the actual curved fluid surface as long as the intersection with the gas region and the solid wall region is represented correctly. The surface tangent of the fluid region is not used to determine the direction of the effective surface tension.

The fluid pressure is applied as an average pressure to the fluid–solid interface. Average pressure can be defined for each fluid material on a list of solid wall materials, for example:

```
pdbSetDoubleArray <FluidMaterial> AveragePressure {<mat1> <n> <mat2> <n>}
```

NOTE There is no default mechanics boundary condition for simulations with fluid regions. Boundary conditions must be defined with the `stressdata` command.

Intrinsic Stress

Certain process steps require the deposition of materials with intrinsic stresses. Sentaurus Interconnect can be used to model these process steps. The intrinsic stresses (`StressELXX`, `StressELYY`, `StressELZZ`, `StressELXY`, `StressELYZ`, `StressELZX`) can be prescribed in the `deposit` command (see [deposit on page 401](#)). After stress relaxation, the resulting stresses will be less than the prescribed ones by default. You can scale the prescribed stresses so that for a flat surface, the relaxed stress will be the same as the prescribed stress. To scale the stresses, use the command:

```
pdbSet Mechanics StressRelaxFactor 1
```

For deposition in three dimensions, you can specify stresses in specific layers using the `stressdata` command (see [stressdata on page 634](#)). For example, the following command sets the `yy` component of the intrinsic stress in the nitride to 1.4×10^{10} dyn/cm²:

```
stressdata nitride syyi= 1.4e10
```

For interconnect simulations, intrinsic stresses in metal lines can be modeled as width dependent [14] with either a linear relation or a logarithmic relation, using the parameters defined through the `stressdata` command:

- If modeled as a linear relation, the total intrinsic stresses are given by:

$$\sigma_{xx} = \sigma_{xxi} + \sigma_{xx1} \frac{w_x}{w_b} \quad \sigma_{yy} = \sigma_{yyi} + \sigma_{yy1} \frac{w_y}{w_b} \quad \sigma_{zz} = \sigma_{zzi} + \sigma_{zz1} \frac{w_z}{w_b}$$

- If modeled as a natural logarithmic relation, the total intrinsic stresses are given by:

$$\sigma_{xx} = \sigma_{xxi} + \sigma_{xx2} \ln \frac{w_x}{w_b} \quad \sigma_{yy} = \sigma_{yyi} + \sigma_{yy2} \ln \frac{w_y}{w_b} \quad \sigma_{zz} = \sigma_{zzi} + \sigma_{zz2} \ln \frac{w_z}{w_b}$$

where σ_{xxi} , σ_{xx1} , σ_{xx2} , w_b are defined through the parameters `sxxi`, `sxx1`, `sxx2`, and `base` of the `stressdata` command. The other two components (`yy` and `zz`) are defined in the same way, and w is calculated internally with respect to the region (not material) boundaries.

The intrinsic stress introduced during material deposition or insertion can also be predefined per material with:

- `stressdata` command options, for example:

```
stressdata <material> deposit.intrinsic sxxi=<n> syyi=<n> szzi=<n> sxyi=<n>  
syzi=<n> szxi=<n>
```

The specified material must have been defined and will be checked against the list of existing materials.

- PDB parameters, for example:

```
pdbSetDouble <material> Mechanics Deposit.Intrinsic.StressXX <n>  
pdbSetDouble <material> Mechanics Deposit.Intrinsic.StressYY <n>  
pdbSetDouble <material> Mechanics Deposit.Intrinsic.StressZZ <n>  
pdbSetDouble <material> Mechanics Deposit.Intrinsic.StressXY <n>  
pdbSetDouble <material> Mechanics Deposit.Intrinsic.StressYZ <n>  
pdbSetDouble <material> Mechanics Deposit.Intrinsic.StressZX <n>
```

There is no syntax check of the material name or the parameter name. The stress unit here is dyn/cm^2 .

To omit the above materialwise intrinsic stress definition, use:

```
pdbSet Mechanics Deposit.Intrinsic.Stress 0
```

Stress Rebalancing After Etching and Deposition

When materials are removed from or added to a given structure, physical stress distributions generally change with the corresponding geometry and boundary changes. In simulations, a stress-rebalancing step is required to re-establish the stress equilibrium in the structure and to conform the stress distributions to the new boundaries. By default, a stress-rebalancing operation is called after etching or deposition is performed. To omit the stress-rebalancing step, use:

```
pdbSet Mechanics EtchDepoRelax 0
```

By default, an elastic stress rebalancing is performed irrespective of the materials present. To perform stress rebalancing using other material models, the following PDB parameter must be switched on:

```
pdbSet Mechanics Full.Stress.Update 1
```

If stress balancing is switched on with the `Full.Stress.Update` parameter and inelastic materials (for example, viscoelastic, plastic, viscoplastic, and creep) are present, a time of 1.0 s is used for the nonlinear simulation. This is equivalent to switching off the stress balancing step and performing a solve step for 1.0 s immediately after the etching or deposition step. If a time other than 1.0 s is required for nonlinear material stress balancing, switch off `EtchDepoRelax`, and perform a solve step with the time specified.

Automated Tracing of Stress History

Thermal residual stress in a given device structure is a function of its fabrication history, which consists of process steps at various temperatures and temperature ramps in between. To model

3: Computing Mechanical Stress

Saving Stress and Strain Components

stress evolution accurately, all temperature ramps should be traced. When the parameter `StressHistory` is switched on, for example:

```
pdbSet Mechanics StressHistory 1
```

the temperature gaps between process steps such as temperature ramp, deposition, and etching are detected and filled with instant stress rebalancing, solving for thermal mismatch strains and stresses.

When stress history is traced and rebalancing is performed, it is done elastically even if inelastic materials are present.

Saving Stress and Strain Components

By default, stress-tensor components are saved on both elements and nodes. The elastic portions of the strain-tensor components also are saved on both elements and nodes by default. The elastic strains are computed from stresses using isotropic elasticity by default. The anisotropic elasticity also can be used for a given crystalline material when the corresponding `pdb` parameter `Anisotropic` is set. The elastic strain-field computing and saving operation can be omitted using the command:

```
pdbSet Mechanics saveElasticStrain 0
```

The stress tensor can be decomposed and the resulting dilatational and deviatoric stress components can be saved on nodes when the following parameter is switched on:

```
pdbSet Mechanics decomposeStress 1
```

Description of Output Fields

Sentaurus Interconnect assumes that stresses and strains are defined on element Gauss points. Values from the element Gauss points also are interpolated to the element centers using local averaging. However, not all tools can read or visualize element values. For this reason, Sentaurus Interconnect performs an element-to-node interpolation of stresses as a postprocessing step and writes both forms of stresses to output.

Element stresses have the prefix `StressEL`, Gauss point stresses have the prefix `StressGP`, and nodal stresses have the prefix `Stress`. The tensor components are given by the suffixes `XX`, `XY`, `YY`, `YZ`, `ZX`, and `ZZ`.

In history-dependent materials, you cannot create a simple closed-form relation between stresses and strains. It is useful, however, to compute the elastic component. The elastic component of the strain is an indicator of the stored strain energy in the system. In addition, the

elastic component of the strain is the total strain in elastic materials such as silicon and polysilicon.

Pressure is one-third of the negative of the trace of the stress tensor:

$$P = -\frac{1}{3} \sum_i \sigma_{ii} \quad (55)$$

In Sentaurus Interconnect, the `select` command performs Tcl-level and Alagator-level operations. To access the stress components, use the `select` command.

The stresses and strains are represented as symmetric tensors. To access the `xx`, `yy`, and `zz` components of nodal stress values, the variable references for the `select` command are `Stress_xx`, `Stress_yy`, and `Stress_zz`, respectively. To access the `xy`, `yz`, and `zx` components, use `Stress_xy`, `Stress_yz`, and `Stress_zx`, respectively.

For element values, the Boolean keyword `element` of the `select` command must be set to `true`. To access the `xx`, `yy`, and `zz` components of the element stress values, the field references for the `select` command are `StressEL_xx`, `StressEL_yy`, and `StressEL_zz`, respectively. To access the `xy`, `yz`, and `zx` components, use `StressEL_xy`, `StressEL_yz`, and `StressEL_zx`, respectively.

The nodal fields `NodalForce` and `ReactionForce` are not saved in the output files by default. To save the `NodalForce` and `ReactionForce` fields, set the following parameters:

```

pdbSetBoolean Mechanics Save.Nodal.Force 1
pdbSetBoolean Mechanics Save.Reaction.Force 1

```

[Table 5](#) describes the mechanics-related fields that are defined on elements. [Table 6 on page 95](#) describes the mechanics-related fields that are defined on nodes.

Table 5 Fields defined on elements

Field name in Sentaurus Visual	Field name in Sentaurus Interconnect	Description
BaseElasticStrainEL-XX (unitless)	BaseElasticStrainEL_xx (unitless)	XX component of elastic element strain for standard linear solid viscoelasticity model
BaseElasticStrainEL-XY (unitless)	BaseElasticStrainEL_xy (unitless)	XY component of elastic element strain for standard linear solid viscoelasticity model
BaseElasticStrainEL-YY (unitless)	BaseElasticStrainEL_yy (unitless)	YY component of elastic element strain for standard linear solid viscoelasticity model
BaseElasticStrainEL-YZ (unitless)	BaseElasticStrainEL_yz (unitless)	YZ component of elastic element strain for standard linear solid viscoelasticity model
BaseElasticStrainEL-XZ (unitless)	BaseElasticStrainEL_zx (unitless)	ZX component of elastic element strain for standard linear solid viscoelasticity model

3: Computing Mechanical Stress

Saving Stress and Strain Components

Table 5 Fields defined on elements

Field name in Sentaurus Visual	Field name in Sentaurus Interconnect	Description
BaseElasticStrainEL-ZZ (unitless)	BaseElasticStrainEL_zz (unitless)	ZZ component of elastic element strain for standard linear solid viscoelasticity model
BaseStressEL-XX (Pa)	BaseStressEL_xx (dyn/cm ²)	XX component of elastic element stress for standard linear solid viscoelasticity model
BaseStressEL-XY (Pa)	BaseStressEL_xy (dyn/cm ²)	XY component of elastic element stress for standard linear solid viscoelasticity model
BaseStressEL-YY (Pa)	BaseStressEL_yy (dyn/cm ²)	YY component of elastic element stress for standard linear solid viscoelasticity model
BaseStressEL-YZ (Pa)	BaseStressEL_yz (dyn/cm ²)	YZ component of elastic element stress for standard linear solid viscoelasticity model
BaseStressEL-XZ (Pa)	BaseStressEL_zx (dyn/cm ²)	ZX component of elastic element stress for standard linear solid viscoelasticity model
BaseStressEL-ZZ (Pa)	BaseStressEL_zz (dyn/cm ²)	ZZ component of elastic element stress for standard linear solid viscoelasticity model
CreepEnergyDensEL (J/m ³)	CreepEnergyDensEL (erg/cm ³)	Creep strain energy density
CreepEnergyDensRtEL (W/m ³)	CreepEnergyDensRtEL (erg/cm ³ /s)	Creep strain energy rate density
CreepStrainELEQV (unitless)	CreepStrainELEQV (unitless)	Equivalent creep strain
CreepStrainEL-XX (unitless)	CreepStrainEL_xx (unitless)	XX component of creep strain
CreepStrainEL-XY (unitless)	CreepStrainEL_xy (unitless)	XY component of creep strain
CreepStrainEL-YY (unitless)	CreepStrainEL_yy (unitless)	YY component of creep strain
CreepStrainEL-YZ (unitless)	CreepStrainEL_yz (unitless)	YZ component of creep strain
CreepStrainEL-XZ (unitless)	CreepStrainEL_zx (unitless)	ZX component of creep strain
CreepStrainEL-ZZ (unitless)	CreepStrainEL_zz (unitless)	ZZ component of creep strain
PseudoEnergyDensEL (J/m ³)	PseudoEnergyDensEL (erg/cm ³)	Pseudo strain energy density
StressEL-XX (Pa)	StressEL_xx (dyn/cm ²)	XX component of element stress

Table 5 Fields defined on elements

Field name in Sentaurus Visual	Field name in Sentaurus Interconnect	Description
StressEL-XY (Pa)	StressEL_xy (dyn/cm ²)	XY component of element stress
StressEL-YY (Pa)	StressEL_yy (dyn/cm ²)	YY component of element stress
StressEL-YZ (Pa)	StressEL_yz (dyn/cm ²)	YZ component of element stress
StressEL-XZ (Pa)	StressEL_zx (dyn/cm ²)	ZX component of element stress
StressEL-ZZ (Pa)	StressEL_zz (dyn/cm ²)	ZZ component of element stress
SwellingStrainEL-XX (unitless)	SwellingStrainEL_xx (unitless)	XX component of swelling strain
SwellingStrainEL-XY (unitless)	SwellingStrainEL_xy (unitless)	XY component of swelling strain
SwellingStrainEL-YY (unitless)	SwellingStrainEL_yy (unitless)	YY component of swelling strain
SwellingStrainEL-YZ (unitless)	SwellingStrainEL_yz (unitless)	YZ component of swelling strain
SwellingStrainEL-XZ (unitless)	SwellingStrainEL_zx (unitless)	ZX component of swelling strain
SwellingStrainEL-ZZ (unitless)	SwellingStrainEL_zz (unitless)	ZZ component of swelling strain
ViscoPlasticEnerDensRtEL (W/m ³)	ViscoPlasticEnerDensRtEL (erg/cm ³ /s)	Viscoplastic strain energy rate density

Table 6 Fields defined on nodes

Field name in Sentaurus Visual	Field name in Sentaurus Interconnect	Description
DeformationResistance (Pa)	DeformationResistance (dyn/cm ²)	Deformation resistance
Displacement-X (um)	Displacement_x (cm)	X-component of displacement
Displacement-Y (um)	Displacement_y (cm)	Y-component of displacement
Displacement-Z (um)	Displacement_z (cm)	Z-component of displacement
ElasticEnergyDens (J/m ³)	ElasticEnergyDens (erg/cm ³)	Elastic strain energy density
ElasticStrain-XX (unitless)	ElasticStrain_xx (unitless)	XX component of elastic strain

3: Computing Mechanical Stress

Saving Stress and Strain Components

Table 6 Fields defined on nodes

Field name in Sentaurus Visual	Field name in Sentaurus Interconnect	Description
ElasticStrain-XY (unitless)	ElasticStrain_xy (unitless)	XY component of elastic strain
ElasticStrain-YY (unitless)	ElasticStrain_yy (unitless)	YY component of elastic strain
ElasticStrain-YZ (unitless)	ElasticStrain_yz (unitless)	YZ component of elastic strain
ElasticStrain-XZ (unitless)	ElasticStrain_zx (unitless)	ZX component of elastic strain
ElasticStrain-ZZ (unitless)	ElasticStrain_zz (unitless)	ZZ component of elastic strain
MisesStress (Pa)	MisesStress (dyn/cm ²)	von Mises stress
NodalForce-X (N)	NodalForce_x (dyne)	X-component of the sum of forces at node
NodalForce-Y (N)	NodalForce_y (dyne)	Y-component of the sum of forces at node
NodalForce-Z (N)	NodalForce_z (dyne)	Z-component of the sum of forces at node
PlasticEnergyDens (J/m ³)	PlasticEnergyDens (erg/cm ³)	Plastic strain energy density
PlasticStrainEQV (unitless)	PlasticStrainEQV (unitless)	Equivalent plastic strain
PlasticStrain-XX (unitless)	PlasticStrain_xx (unitless)	XX component of plastic strain
PlasticStrain-XY (unitless)	PlasticStrain_xy (unitless)	XY component of plastic strain
PlasticStrain-YY (unitless)	PlasticStrain_yy (unitless)	YY component of plastic strain
PlasticStrain-YZ (unitless)	PlasticStrain_yz (unitless)	YZ component of plastic strain
PlasticStrain-XZ (unitless)	PlasticStrain_zx (unitless)	ZX component of plastic strain
PlasticStrain-ZZ (unitless)	PlasticStrain_zz (unitless)	ZZ component of plastic strain
Pressure (Pa)	Pressure (dyn/cm ²)	Pressure
PseudoDisplacement-X (um)	PseudoDisplacement_x (cm)	X-component of pseudo displacement
PseudoDisplacement-Y (um)	PseudoDisplacement_y (cm)	Y-component of pseudo displacement
PseudoDisplacement-Z (um)	PseudoDisplacement_z (cm)	Z-component of pseudo displacement

Table 6 Fields defined on nodes

Field name in Sentaurus Visual	Field name in Sentaurus Interconnect	Description
ReactionForce-X (N)	ReactionForce_x (dyne)	X-component of reaction force at node with Dirichlet boundary condition
ReactionForce-Y (N)	ReactionForce_y (dyne)	Y-component of reaction force at node with Dirichlet boundary condition
ReactionForce-Z (N)	ReactionForce_z (dyne)	Z-component of reaction force at node with Dirichlet boundary condition
Stress-XX (Pa)	Stress_xx (dyn/cm ²)	XX component of node stress
Stress-XY (Pa)	Stress_xy (dyn/cm ²)	XY component of node stress
Stress-YY (Pa)	Stress_yy (dyn/cm ²)	YY component of node stress
Stress-YZ (Pa)	Stress_yz (dyn/cm ²)	YZ component of node stress
Stress-XZ (Pa)	Stress_zx (dyn/cm ²)	ZX component of node stress
Stress-ZZ (Pa)	Stress_zz (dyn/cm ²)	ZZ component of node stress
ViscoPlasticEnergyDens (J/m ³)	ViscoPlasticEnergyDens (erg/cm ³)	Viscoplastic strain energy density
ViscoPlasticStrainEQV (unitless)	ViscoPlasticStrainEQV (unitless)	Equivalent viscoplastic strain
ViscoPlasticStrain-XX (unitless)	ViscoPlasticStrain_xx (unitless)	XX component of viscoplastic strain
ViscoPlasticStrain-XY (unitless)	ViscoPlasticStrain_xy (unitless)	XY component of viscoplastic strain
ViscoPlasticStrain-YY (unitless)	ViscoPlasticStrain_yy (unitless)	YY component of viscoplastic strain
ViscoPlasticStrain-YZ (unitless)	ViscoPlasticStrain_yz (unitless)	YZ component of viscoplastic strain
ViscoPlasticStrain-XZ (unitless)	ViscoPlasticStrain_zx (unitless)	ZX component of viscoplastic strain
ViscoPlasticStrain-ZZ (unitless)	ViscoPlasticStrain_zz (unitless)	ZZ component of viscoplastic strain

NOTE The stresses and strains in the output file are according to the unified coordinate system (UCS).

3: Computing Mechanical Stress

Saving Stress and Strain Components

Table 7 lists the mechanics-related fields available for use in the select command.

Table 7 Fields available in select command

Field type	Field name		
Displacement	Displacement_x	Displacement_y	Displacement_z
	StepDisplacement_x	StepDisplacement_y	StepDisplacement_z
Stress	StressEL_xx	StressEL_yy	StressEL_zz
	StressEL_xy	StressEL_yz	StressEL_zx
	MisesStress	Pressure	
	PrincipalStress1	PrincipalStress2	PrincipalStress3
	Stress_xx	Stress_yy	Stress_zz
	Stress_xy	Stress_yz	Stress_zx
Strain	ElasticStrainEL_xx	ElasticStrainEL_yy	ElasticStrainEL_zz
	ElasticStrainEL_xy	ElasticStrainEL_yz	ElasticStrainEL_zx
	PlasticStrainEL_xx	PlasticStrainEL_yy	PlasticStrainEL_zz
	PlasticStrainEL_xy	PlasticStrainEL_yz	PlasticStrainEL_zx
	ViscoPlasticStrainEL_xx	ViscoPlasticStrainEL_yy	ViscoPlasticStrainEL_zz
	ViscoPlasticStrainEL_xy	ViscoPlasticStrainEL_yz	ViscoPlasticStrainEL_zx
	ElasticStrain_xx	ElasticStrain_yy	ElasticStrain_zz
	ElasticStrain_xy	ElasticStrain_yz	ElasticStrain_zx
	PlasticStrain_xx	PlasticStrain_yy	PlasticStrain_zz
	PlasticStrain_xy	PlasticStrain_yz	PlasticStrain_zx
	ViscoPlasticStrain_xx	ViscoPlasticStrain_yy	ViscoPlasticStrain_zz
	ViscoPlasticStrain_xy	ViscoPlasticStrain_yz	ViscoPlasticStrain_zx
Strain	PlasticStrainEQV	ViscoPlasticStrainEQV	
	PrincipalStrain1	PrincipalStrain2	PrincipalStrain3

Tracking Maximum Stresses

During a typical process flow, the maximum stresses may be reached in a process step and, subsequently, the stresses may fall. If the material is prone to failure through delamination or nucleation of dislocations, the failure may occur when the maximum stress is reached.

To always track the maximum stresses, set the following parameter:

```
pdbSet Mechanics SaveMaxStress 1
```

The `StressMaxGP` field is updated when the current stress is greater than the stored stress. In this way, the maximum is maintained throughout the process flow. The maximum element stresses and the von Mises stress are computed and stored.

Using the `stressdata` command (see [stressdata on page 634](#)), a list of maximum stresses (hot spots) and their locations can be obtained. The hot spots can be evaluated by one of the six stress components (`sxx`, `syy`, `szz`, `sxy`, `syx`, and `szx`), the von Mises stress, the principal stress, the principal strain, or the hydrostatic stress (negative pressure value or the pressure). The command returns a list of maximum stress values (largest magnitude, largest tensile, largest compressive) and the corresponding location coordinates.

Principal Stresses

Principal stresses are normal stresses that act on the planes where shear stresses are zero, that is:

$$(\sigma - \sigma_p \mathbf{I}) \hat{\mathbf{n}} = 0 \quad (56)$$

which implies that the determinant of the Cauchy relation in matrix form is zero:

$$\sigma_p^3 - I_1 \sigma_p^2 + I_2 \sigma_p - I_3 = 0 \quad (57)$$

where the coefficients are known as the three stress invariants:

$$\begin{aligned} \sigma_1 &= \sigma_{ii} \\ \sigma_2 &= \frac{1}{2}(\sigma_{ii}\sigma_{jj} - \sigma_{ij}\sigma_{ji}) \\ \sigma_3 &= \det(\sigma_{ij}) \end{aligned} \quad (58)$$

The three principal stresses are ordered from the largest tensile (positive) stress to the largest compressive (negative) stress, and are named:

- First principal stress: $\sigma_1 = \max(\sigma_{p1}, \sigma_{p2}, \sigma_{p3})$
- Second principal stress: $\sigma_2 = I_1 - \sigma_1 - \sigma_3$
- Third principal stress: $\sigma_3 = \min(\sigma_{p1}, \sigma_{p2}, \sigma_{p3})$

The first and third principal stresses are also algebraically the largest and the smallest normal stresses that can be found at this point. Therefore, principal stresses are often adopted as measures to evaluate the integrity of structures.

3: Computing Mechanical Stress

Saving Stress and Strain Components

To calculate the principal stresses, use:

```
pdbSet Mechanics Calculate.Principal.Stress 1
```

The resulting fields are:

- `PrincipalStress1` for the first principal stress.
- `PrincipalStress2` for the second principal stress.
- `PrincipalStress3` for the third principal stress.

If the maximum-stress tracking flag is switched on:

```
pdbSet Mechanics SaveMaxStress 1
```

Sentaurus Interconnect also tracks the maximum principal stresses that occurred during the entire simulation process and stores them as:

- `PrincipalStress1Max`
- `PrincipalStress2Max`
- `PrincipalStress3Max`

Principal Strains

Principal strains are normal strains that act on the planes where shear strains are zero, that is:

$$(\boldsymbol{\varepsilon} - \varepsilon_p \mathbf{I})\hat{\mathbf{n}} = 0 \quad (59)$$

which implies that the determinant of the Cauchy relation in matrix form is zero:

$$\varepsilon_p^3 - I_1 \varepsilon_p^2 + I_2 \varepsilon_p - I_3 = 0 \quad (60)$$

where the coefficients are known as the three strain invariants:

$$\begin{aligned} \varepsilon_1 &= \varepsilon_{ii} \\ \varepsilon_2 &= \frac{1}{2}(\varepsilon_{ii}\varepsilon_{jj} - \varepsilon_{ij}\varepsilon_{ji}) \\ \varepsilon_3 &= \det(\varepsilon_{ij}) \end{aligned} \quad (61)$$

The three principal strains are ordered from the largest tensile (positive) strain to the largest compressive (negative) strain, and are named:

- First principal strain: $\varepsilon_1 = \max(\varepsilon_{p1}, \varepsilon_{p2}, \varepsilon_{p3})$
- Second principal strain: $\varepsilon_2 = I_1 - \varepsilon_1 - \varepsilon_3$
- Third principal strain: $\varepsilon_3 = \min(\varepsilon_{p1}, \varepsilon_{p2}, \varepsilon_{p3})$

The first and third principal strains are also algebraically the largest and the smallest normal strains that can be found at this point.

To calculate the principal strains, use:

```
pdbSet Mechanics Calculate.Principal.Strain 1
```

The resulting fields are:

- `PrincipalStrain1` for the first principal strain.
- `PrincipalStrain2` for the second principal strain.
- `PrincipalStrain3` for the third principal strain.

If the maximum-stress tracking flag is switched on:

```
pdbSet Mechanics SaveMaxStress 1
```

Sentaurus Interconnect also tracks the maximum principal strains that occurred during the entire simulation process and stores them as:

- `PrincipalStrain1Max`
- `PrincipalStrain2Max`
- `PrincipalStrain3Max`

Nodal Stress and Strain at Like-Material Interface

Nodal stress and strain data fields are interpolated regionwise from the corresponding element data fields. They are discontinuous across the interfaces of different materials. At the interface of like materials, the nodal fields `Pressure` and `ElasticStrain` are specially interpolated to have a continuous distribution. This list of nodal data fields can be changed with:

```
pdbSet Mechanics Like.Material Boundary Continuous {<list of fields>}
```

Each like-material interface can have its own list of continuous nodal data fields, for example:

```
pdbSetString <mat1_mat2> Mechanics Boundary Continuous {<list of fields>}
```

Submodeling

Submodeling is a technique to obtain accurate stress in a local region with mesh refinements, while the full model can be simulated with a coarser mesh. This technique extracts the displacement field from the global structure, interpolates the field onto the local refined model, and applies the displacement on the outer surfaces of the local model as boundary conditions.

3: Computing Mechanical Stress

Submodeling

Submodeling is based on St. Venant's principle [15], which states that the stress and strain fields of the original system and the statically equivalent system on parts of the body far away from the loading regions are approximately the same. The principle implies that accurate results can be obtained in the submodel if the boundaries of the submodel are far away from the stress concentration.

To initiate a submodeling analysis, use the command:

```
stressdata global.model=<c>
```

or equivalently, use:

```
pdbSet Mechanics Submodel <c>
```

where <c> is the name of the file that has both grid and field data of the global model. If submodeling is specified through either the `stressdata` command or `pdb` parameter, all other mechanics boundary condition definitions will be ignored. The global model should contain a step displacement field (`StepDisplacement`). In addition, the local model must be cuboid shaped, and all six boundaries must be within the domain of the global model.

The local model with a cuboid shape also can be created by cutting the global structure with user-specified bounding box dimensions. To define the cuboid region for submodeling, use the command:

```
stressdata align.cut=<n> global.model=<c> min= {<n> <n> <n>} max= {<n> <n> <n>}
```

NOTE Sometimes, due to round-off issues, the specified coordinates used to create the local model may deviate slightly from the coordinates already specified in the masks used to build the structure. This could lead to very short edges and failures in the simulation. To snap the local model coordinates to the existing mask edges, use the `align.cut` argument.

When the tool interpolates the step displacement field from the global model to the boundaries of the local model, it does not require the two materials at the same location of the global and local models to match. This facilitates choosing approximate materials for the global model as a representative unit cell (RUC) to perform a multiscale simulation [16]. When performing an RUC simulation, iterations between the global model approximation and the local model are necessary.

If the local model has a different material compared to the global model, for example, elastoplastic instead of linear elastic, multiple time steps may be needed to solve for stresses. In such cases, boundary conditions on the outer surfaces of the local model are scaled so that the total displacements at the end of the multiple-step analysis are the same as the displacements in the global model.

When simulating stresses due to thermal expansion, the local model can be analyzed in two ways. The first option is to use the `temp_ramp` command to specify the same temperature profile as in the global model for a multiple-step analysis. The second option is to perform a single-step analysis after interpolating temperatures from the global model to the local model using the command:

```
pdbSet Mechanics Submodel.Add.Global.Temperature 1
```

NOTE For the second option to give the same thermal strains in the local model as the global model, it is important that the initial temperature in the local model is the same as the global model, and the analysis must be limited to a single time step.

NOTE Use absolute coefficients of thermal expansion instead of relative coefficients if the substrate is removed from the local model. This can be achieved by setting the parameter `RefThExpCoeff` as follows:

```
pdbSetDouble Mechanics RefThExpCoeff 0
```

NOTE When using the `temp_ramp` command in the local model, you must first use `temp_ramp clear` to remove any `temp_ramp` commands defined in the global model.

NOTE In submodeling, the step displacements (see [Principal Stresses on page 99](#)) *not* the total displacements are extracted and used as boundary conditions for the local model.

Crack Analysis

Crack analysis refers to introducing pre-existing cracks in structures and performing stress analysis. If no cohesive zone model (CZM) is used (see [Cohesive Zone Modeling on page 106](#)), the crack surfaces remain traction free. Depending on the type of loading on the structure, a crack opens up in one or a combination of the three failure modes: Mode I, Mode II, and Mode III.

In Sentaurus Interconnect, a crack can be defined at the interface of two materials for 2D or 3D structures. For a crack within a single material, an interface must be defined at the location of the crack, with one of the materials defined using the `new.like` parameter in the `mater` command.

A crack is defined using the `crack` command (see [crack on page 389](#)). The `normal` parameter defines the normal direction of the crack surfaces, as \vec{n} shown in [Figure 20 on page 115](#). The `CZM` parameter specifies the name of the CZM law defined using the `CZM` command. The CZM law name `none` is reserved to set an initial crack without any traction on crack surfaces.

3: Computing Mechanical Stress

Crack Analysis

When the CZM is used for a crack, tractions are assumed to be nonzero over the entire length of the crack. To apply the CZM over certain parts of the crack, use the `CZM.box` parameter to provide the minimum and the maximum coordinates of the bounding boxes. In addition, use `CZM.polarity` to specify whether the CZM is active inside or outside the bounding box.

A bounding box is used to define the location of the crack along the interface. In a 2D structure, the intersection of the bounding box (rectangle) with the interface (line) defines the crack tip (point). The crack extends along the interface from one crack tip to the other crack tip or the boundary. In a 3D structure, the intersection of the bounding box (cuboid) with the interface (surface) defines the crack front (line). The crack extends along the interface bounded by the crack front or the boundary. If the interface is smaller than the bounding box, the crack tip or crack front lies at the edge of the interface. If the interface contains a bubble-shaped crack, the crack front becomes a closed loop that lies completely inside the bounding box.

Figure 15 shows how a crack is defined in 2D and 3D structures. The 2D structure has the same material on both sides of the interface. The 3D structure has different materials on the top and bottom. The top block also has an interface between two prismatic halves that have the same material. For the crack definition, materials for only one of the prisms and the bottom block are used for the `crack` command. Even though the bounding box extends over the entire interface between the top and bottom blocks, the crack is only defined in the triangular portion under the selected prism with the material specified in the `crack` command. The longer side of the triangle forms the crack front.

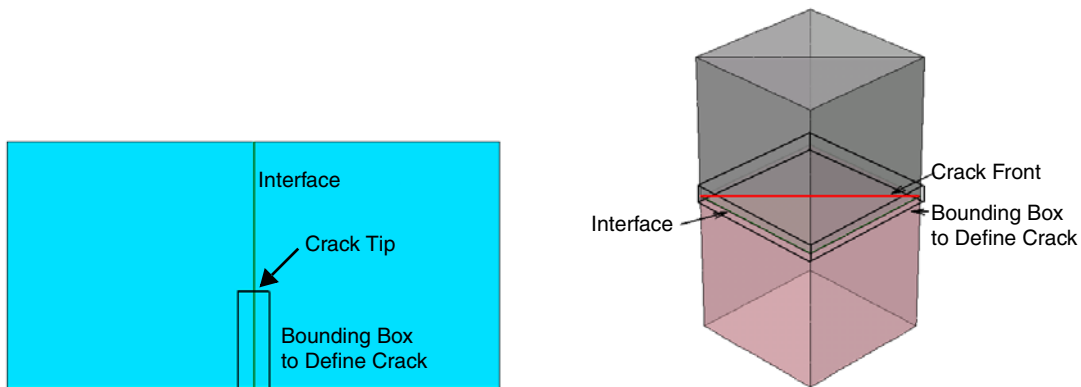


Figure 15 Crack definition in (*left*) 2D structure and (*right*) 3D structure

To initiate a crack simulation, use:

```
pdbSet Mechanics Model Crack
```

V-Notch Cracks

The notched specimen has been extensively used to study crack initiation. For structures with a V-shaped notch (see [Figure 16](#)), the same syntax for crack definition can be applied. This will not affect stress analysis, but it is required for the J-integral calculation at the V-notch crack front.

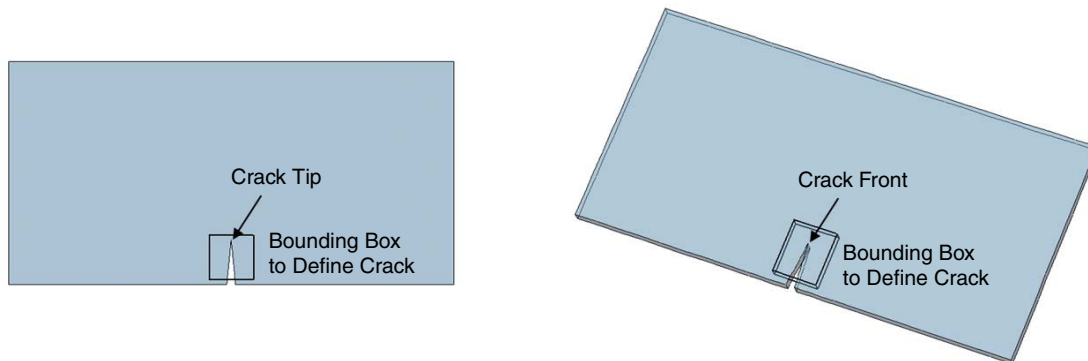


Figure 16 V-notch crack definition in (*left*) 2D structure and (*right*) 3D structure

A bounding box intersecting the gas–material interface is used to define the crack surface. The crack front is set automatically to be where the sharpest angle is on the crack surface.

Bulk Cracks

A bulk crack is an alternative way of defining a crack in bulk material without creating an interface. In two dimensions, a bulk crack is defined using one or more line segments specified in terms of point coordinates. Multiple connected line segments define a segmented crack path.

In three dimensions, a bulk crack is defined using one or more previously specified polygons. Multiple connected polygons define a faceted crack surface. [Figure 17 on page 106](#) shows 2D and 3D cracks defined in bulk material.

Use `segments` and `polygons` in the `crack` command to define 2D and 3D bulk cracks, respectively. Use `mat.1` in the `crack` command to specify the name of the bulk material containing the crack.

NOTE Bulk cracks require boundary representation for contacts and 2D structures. The following parameters must be used for bulk cracks; if not defined, they will be switched on automatically:

```
pdbSet Grid Contacts.In.Brep 1
pdbSet Grid MGoals use.brep.2d 1
```

3: Computing Mechanical Stress

Cohesive Zone Modeling

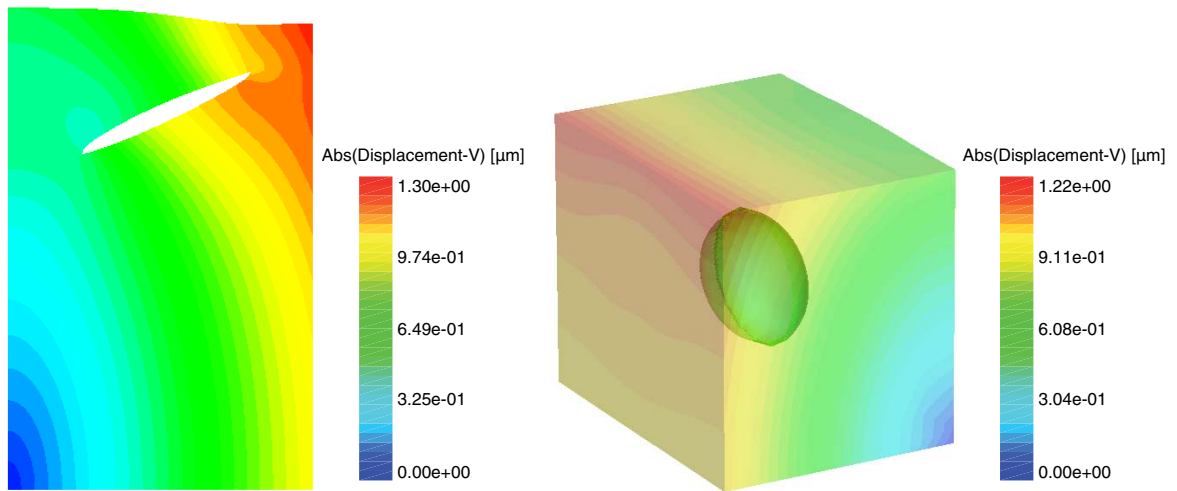


Figure 17 Deformed configurations of (left) 2D bulk crack and (right) 3D bulk crack

Cohesive Zone Modeling

Cohesive zone modeling is a way to model propagation of a crack along a predefined path, for example, delamination or separation of material interfaces. It involves defining a CZM law to model the traction–separation behavior of cohesive surfaces. Sentaurus Interconnect provides two CZM laws: exponential law and triangular law.

To define a CZM law, use (see [CZM on page 396](#)):

```
CZM name=<c> (law= exponential | triangular) normal.stress.max=<n>
shear.stress.max=<n> normal.toughness=<n> shear.toughness=<n>
mat.1=<c> mat.2=<c>
```

Exponential Law

The exponential constitutive law for a cohesive surface is defined by the interface potential [17]:

$$\phi(\delta) = \phi_{n0} + \phi_{n0} e^{-\frac{\delta_n}{\delta_{n0}}} \left[\left(1 - r + \frac{\delta_n}{\delta_{n0}}\right) \cdot \frac{1 - q}{r - 1} - \left(q + \frac{r - q}{r - 1} \cdot \frac{\delta_n}{\delta_{n0}}\right) e^{-\frac{\delta_n}{\delta_{n0}}} \right] e^{-\frac{\delta_s^2}{\delta_{s0}^2}} \quad (62)$$

where:

- $q = \phi_{t0}/\phi_{n0}$ is the ratio of the work of separation along the tangent direction over the normal direction.
- $r = \delta_n^*/\delta_{n0}$ is the ratio of the opening distance along the normal direction after complete shear separation over the maximum opening distance in the normal direction.
- δ_n and δ_t are the opening distances along the normal and tangent directions of a CZM surface, respectively.
- δ_{n0} and δ_{t0} are the maximum opening distances along the normal and tangent directions of a CZM surface, respectively.

The ratios r and q in Eq. 62 define the coupling parameters for the normal and tangential directions. In Sentaurs Interconnect, the parameter q is set to 1, so the work of separation along the tangent direction is assumed to be equal to the work of separation along the normal direction. The exponential law also assumes the constitutive behavior of the cohesive surfaces to be elastic; therefore, loading and unloading occur along the same path.

The tractions at a cohesive surface are defined accordingly as:

$$T = \frac{\partial \phi}{\partial \delta} \quad (63)$$

whose normal and shear components are shown in Figure 18.

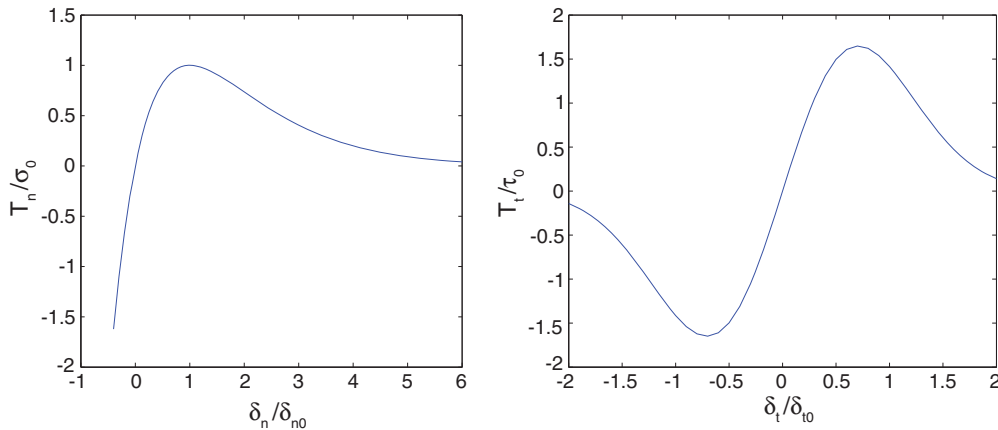


Figure 18 Cohesive zone modeling: Exponential law

Triangular Law

The triangular law for cohesive surfaces is defined by a bilinear curve with a specified peak. The constitutive behavior is assumed to be elastic but without any coupling between the normal

3: Computing Mechanical Stress

Cohesive Zone Modeling

and the tangential responses. The normal and shear components of traction for the triangular law can be expressed directly as:

$$T_n(\delta_n) = \begin{cases} 2\left(\frac{\phi_{n0}}{\delta_{n0}}\right)\left(\frac{\delta_n}{\delta_{nc}}\right) & , (0 \leq \delta_n \leq \delta_{nc}) \\ 2\left(\frac{\phi_{n0}}{\delta_{n0}}\right)\left(\frac{\delta_{n0} - \delta_n}{\delta_{n0} - \delta_{nc}}\right) & , (\delta_{nc} \leq \delta_n \leq \delta_{n0}) \end{cases}$$

$$T_t(\delta_t) = \begin{cases} 2\left(\frac{\phi_{t0}}{\delta_{t0}}\right)\left(\frac{\delta_t}{\delta_{tc}}\right) & , (|\delta_t| \leq \delta_{tc}) \\ 2\left(\frac{\phi_{t0}}{\delta_{t0}}\right)\left(\frac{\delta_{t0} - \delta_t}{\delta_{t0} - \delta_{tc}}\right) & , (\delta_{tc} \leq \delta_t \leq \delta_{t0}) \\ -2\left(\frac{\phi_{t0}}{\delta_{t0}}\right)\left(\frac{\delta_{t0} + \delta_t}{\delta_{t0} - \delta_{tc}}\right) & , (-\delta_{t0} \leq \delta_t \leq -\delta_{tc}) \end{cases} \quad (64)$$

where:

- ϕ_{n0} and ϕ_{t0} are the work of separation along the normal and tangent directions of a CZM surface, respectively.
- δ_{n0} and δ_{t0} are the maximum opening distances along the normal and tangent directions of a CZM surface, respectively.
- δ_{nc} and δ_{tc} are the critical opening distances corresponding to the peak along the normal and tangent directions of a CZM surface, respectively.

Figure 19 shows the traction–separation curves for the triangular law with the peak in the middle.

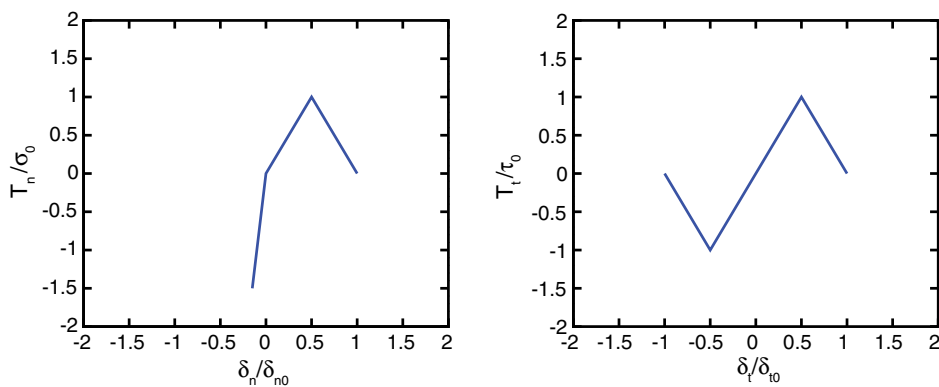


Figure 19 Cohesive zone modeling: Triangular law

When the cohesive surfaces close up on each other after separation, contact constraints are needed to prevent penetration. In Sentaurus Interconnect, these constraints are enforced along the normal direction using the penalty method; frictionless conditions are assumed for the tangent direction. The normal component of traction for the exponential law and the triangular law are defined as:

$$T_n(\delta_n) = \begin{cases} (-a)\left(\frac{\phi_{n0}}{\delta_{n0}}\right)\left(\frac{\delta_n}{\delta_{n0}}\right), & \text{exponential} \\ (-2a)\left(\frac{\phi_{n0}}{\delta_{n0}}\right)\left(\frac{\delta_n}{\delta_{nc}}\right), & \text{triangular} \end{cases} \quad (65)$$

where a negative sign indicates compression, and a is an antipenetration factor that can be defined using the command:

```
pdbSet Mechanics Anti.Penetration.Factor <n>
```

The default value for a is 5.0. You can specify a larger value to reduce penetration, provided it does not severely affect the convergence of the nonlinear solution.

The CZM name `default` is reserved to define the CZM law that will be applied to any interfaces where no CZM law is applicable. If you do not define the default CZM law, Sentaurus Interconnect automatically defines one, for example:

```
CZM name= default law= exponential normal.stress.max= 1e10 \  
shear.stress.max= 1e10 normal.toughness= 1e6 shear.toughness= 1e6
```

A CZM simulation is invoked using:

```
pdbSet Mechanics Model CZM
```

NOTE A CZM simulation may give an inaccurate solution or may cause convergence difficulties with second-order finite elements when modeling uniform pressure distribution on crack surfaces. First-order finite elements should be used in such cases.

Damage Variable

The damage variable is an indicator of the extent of material failure based on CZM element separation. Normal and shear damage variables are calculated independently and range between 0 (no damage) and 1 (complete damage). For both triangular law and exponential law, material failure starts when the maximum normal or shear stresses are reached.

3: Computing Mechanical Stress

Cohesive Zone Modeling

For triangular law, complete material failure occurs at the maximum opening and the damage variable takes the form:

$$\begin{aligned}
 d_n(\delta_n) &= \begin{cases} 0, & (0 \leq \delta_n \leq \delta_{nc}) \\ \frac{\delta_n - \delta_{nc}}{\delta_{n0} - \delta_{nc}}, & (\delta_{nc} \leq \delta_n \leq \delta_{n0}) \\ 1, & (\delta_{n0} \leq \delta_n) \end{cases} \\
 d_t(\delta_t) &= \begin{cases} 0, & (|\delta_t| \leq \delta_{tc}) \\ \frac{\delta_t - \delta_{tc}}{\delta_{t0} - \delta_{tc}}, & (\delta_{tc} \leq \delta_t \leq \delta_{t0}) \\ -\frac{\delta_t + \delta_{tc}}{\delta_{t0} - \delta_{tc}}, & (-\delta_{t0} \leq \delta_t \leq -\delta_{tc}) \\ 1, & (\delta_{t0} \leq |\delta_t|) \end{cases} \tag{66}
 \end{aligned}$$

For exponential law, complete material failure is set to six times the maximum normal opening and three times the maximum tangential opening. The damage variable takes the form:

$$\begin{aligned}
 d_n(\delta_n) &= \begin{cases} 0, & (0 \leq \delta_n \leq \delta_{n0}) \\ \frac{\delta_n - \delta_{n0}}{5\delta_{n0}}, & (\delta_{n0} \leq \delta_n \leq 6\delta_{n0}) \\ 1, & (6\delta_{n0} \leq \delta_n) \end{cases} \\
 d_t(\delta_t) &= \begin{cases} 0, & (|\delta_t| \leq \delta_{t0}) \\ \frac{\delta_t - \delta_{t0}}{2\delta_{t0}}, & (\delta_{t0} \leq \delta_t \leq 3\delta_{t0}) \\ -\frac{\delta_t + \delta_{t0}}{2\delta_{t0}}, & (-3\delta_{t0} \leq \delta_t \leq -\delta_{t0}) \\ 1, & (3\delta_{t0} \leq |\delta_t|) \end{cases} \tag{67}
 \end{aligned}$$

Normal and shear damage variables are calculated and stored on the interface elements that coincide with CZM elements. The internal field names are `DamageVariableNEL` and `DamageVariableSEL`, respectively. The damage variable fields defined on interface nodes are interpolated from interface elements and are written to TDR files as `DamageVariableN` and `DamageVariableS`.

References

- [1] J. C. Simo and T. J. R. Hughes, *Computational Inelasticity*, vol. 7, New York: Springer, 1998.
- [2] W. Ramberg and W. R. Osgood, *Description of Stress-Strain Curves by Three Parameters*, National Advisory Committee for Aeronautics, Technical Note No. 902, Washington, DC, USA, July 1943.
- [3] J. Lubliner, *Plasticity Theory*, New York: Macmillan Publishing Company, 1990.
- [4] S. B. Brown, K. H. Kim, and L. Anand, “An Internal Variable Constitutive Model for Hot Working of Metals,” *International Journal of Plasticity*, vol. 5, no. 2 pp. 95–130, 1989.
- [5] G. G. Weber *et al.*, “An Objective Time-Integration Procedure for Isotropic Rate-Independent and Rate-Dependent Elastic-Plastic Constitutive Equations,” *International Journal of Plasticity*, vol. 6, no. 6, pp. 701–744, 1990.
- [6] G. Z. Wang *et al.*, “Applying Anand Model to Represent the Viscoplastic Deformation Behavior of Solder Alloys,” *Journal of Electronic Packaging*, vol. 123, no. 3, pp. 247–253, 2001.
- [7] Q. Wang *et al.*, “Anand Parameter Test for Pb-Free Material SnAgCu and Life Prediction for a CSP,” in *8th International Conference on Electronic Packaging Technology (ICEPT)*, Shanghai, China, pp. 1–9, August 2007.
- [8] J. Wilde *et al.*, “Rate Dependent Constitutive Relations Based on Anand Model for 92.5Pb5Sn2.5Ag Solder,” *IEEE Transactions on Advanced Packaging*, vol. 23, no. 3, pp. 408–414, 2000.
- [9] H. J. Frost and M. F. Ashby, *Deformation-Mechanism Maps, The Plasticity and Creep of Metals and Ceramics*, Oxford: Pergamon Press, 1982.
- [10] S. Wiese, F. Feustel, and E. Meusel, “Characterisation of constitutive behaviour of SnAg, SnAgCu and SnPb solder in flip chip joints,” *Sensors and Actuators A*, vol. 99, no. 1–2, pp. 188–193, 2002.
- [11] B. Wilshire and C. J. Palmer, “Grain size effects during creep of copper,” *Scripta Materialia*, vol. 46, no. 7, pp. 483–488, 2002.

3: Computing Mechanical Stress

References

- [12] M. P. Phaniraj, M. J. N. V. Prasad, and A. H. Chokshi, "Grain-size distribution effects in plastic flow and failure," *Materials Science and Engineering: A*, vol. 463, no. 1–2, pp. 231–237, 2007.
- [13] R. Gatti *et al.*, "Dislocation engineering in SiGe heteroepitaxial films on patterned Si (001) substrates," *Applied Physics Letters*, vol. 98, no. 12, p. 121908, 2011.
- [14] Y.-C. Joo, J.-M. Paik, and J.-K. Jung, "Effect of Microstructure and Dielectric Materials on Stress-Induced Damages in Damascene Cu/Low-k Interconnects," in *MRS Symposium Proceedings, Materials, Technology and Reliability of Advanced Interconnects*, vol. 863, San Francisco, CA, USA, p. B7.6/O11.6, March 2005.
- [15] S. Timoshenko and D. H. Young, *Engineering Mechanics*, New York: McGraw-Hill Book Company, 3rd ed., 1951.
- [16] V. Fiori *et al.*, "Chip-Package Interactions: Some Combined Package Effects on Copper/Low-k Interconnect Delaminations," in *2nd Electronics System-Integration Technology Conference*, Greenwich, UK, pp. 713–718, September 2008.
- [17] X.-P. Xu and A. Needleman, "Numerical Simulations of Fast Crack Growth in Brittle Solids," *Journal of the Mechanics and Physics of Solids*, vol. 42, no. 9, pp. 1397–1434, 1994.

This chapter introduces the stress analysis–related postprocessing and reliability evaluation capabilities of Sentaurus Interconnect.

Stress-Induced Mobility Enhancement

The mobility enhancement induced by the stress field is defined through piezoresistance change [1]:

$$\frac{\Delta\rho_i}{\rho} = \pi_{ik}\sigma_k \quad (68)$$

where π_{ik} is the piezoresistance coefficient tensor component, and σ_k is the stress component in compact form. Because of the cubic symmetry of silicon crystals, Sentaurus Interconnect only uses the three independent coefficients π_{11} , π_{12} , and π_{44} . If the axes of the simulation coordinate system do not align with the crystal axes, a coordinate system transformation is performed to obtain the piezoresistance coefficients in the simulation coordinate system.

The piezoresistance coefficients P11/P12/P44 (unit: %/GPa) are defined through the following parameters:

```
pdbSetDouble <material> Mechanics <P11 | P12 | P44> <n>
```

For example:

```
pdbSet Silicon Mechanics P11 -102.2
```

NOTE Here, `pdbSet` is used instead of `pdbSetDouble` because P11 | P12 | P44 are predefined for the material silicon in the Parameter Database Browser.

The calculation of stress-induced mobility enhancement can be invoked during the solving process using a `pdb` Boolean parameter:

```
pdbSet Mechanics Calculate.Mobility 1
```

The resulting field is saved as `MobilityEL` (element data) and its unit is percent.

4: Mechanics Postprocessing

Displacement Field

The default device model is planar type. For FinFET-type 3D devices, set:

```
pdbSetBoolean Mechanics FinFET.Mobility 1
```

With this model, the channel orientation can be specified as either 1 for a (110) orientation of the fin or 2 for a (100) orientation of the fin with:

```
pdbSetDouble <material> Mechanics FinFET.Channel.Orientation <1 | 2>
```

The carrier type can be specified as either 0 for electrons or 1 for holes with:

```
pdbSetDouble <material> Mechanics FinFET.Carrier.Type <0 | 1>
```

Alternatively, it can be performed as a postprocess step after simulation using the `mobility` command (see [mobility on page 516](#)), for example:

```
mobility device.model=planar region.name=Silicon \  
  piezo.coefficients= {P11=-6.6, P12=1.1, P44=-138.1}
```

This command calculates the stress-induced mobility enhancement in the material `Silicon` using the `psilicon` material parameters (P11, P12, P44) and stores it in the field named `MobilityEL`.

The different device models available are planar-type and FinFET-type 3D devices. Details are available from [mobility on page 516](#).

Displacement Field

The stress analysis kernel in Sentaurus Interconnect solves the velocity at grid points of the given structure. The deformation is obtained by integrating the solution over time. By default, the displacement field is saved in the result file. Saving this field can be switched off by:

```
pdbSet Mechanics Save.Displacement.Field 0
```

Two displacement fields are saved:

- Total displacement for the structure simulation – The displacement solution is stored as a vector field named `Displacement`. This displacement is the incremental total of displacement from all process steps carried out during the simulation. Saving this field can be switched off by:

```
pdbSet Mechanics Save.Total.Displacement 0
```

- Incremental displacement for the last process step – This solution is stored as `StepDisplacement`. Saving this field can be turned off by:

```
pdbSet Mechanics Save.Step.Displacement 0
```


If there are displacement fields in the result file, you can obtain a deformation plot of the current structure using the `struct` command.

You also can use this command to generate files for viewing the crack opening status after a crack simulation (see `struct` on page 643).

J-Integral

J-integral is an expression to compute the strain-energy release rate, based on the distribution of stresses around a crack tip. It is defined as [2][3]:

$$J = \lim_{\Gamma \rightarrow 0} \int_{\Gamma} \left(W \delta_{1j} - \sigma_{ij} \frac{\partial u_i}{\partial x_1} \right) n_j ds \quad (69)$$

where:

- W is the strain energy density.
- δ_{ij} is the Kronecker operator.
- 1 is the direction of the crack propagation (this is orthogonal to the plane containing the crack normal and the crack front as shown in Figure 20).

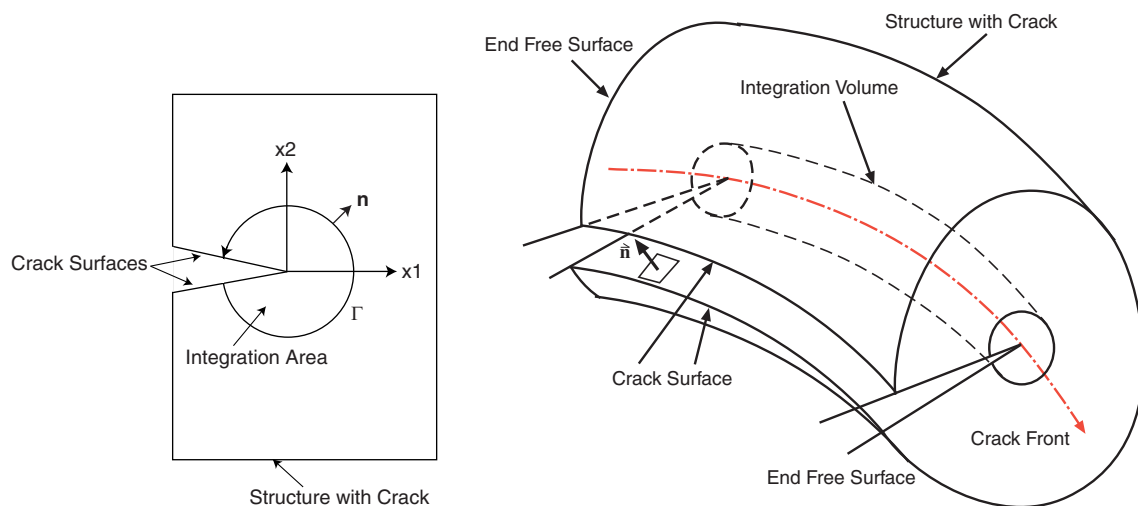


Figure 20 Schematic of a crack definition: (left) two dimensions and (right) three dimensions

It can be easily shown that, for homogenous linear or nonlinear elastic materials, the J-integral value is independent of the integration path around the crack tip. In addition, by Gauss theorem, the contour integral can be converted to an area or a volume integral without any loss of accuracy. In finite-element models where element stress values are more accurate compared to

4: Mechanics Postprocessing

J-Integral

nodal values, J-integral is generally computed over the area or volume around the crack tip or crack front. In 3D structures, J-integral is computed at each node along the crack front.

As shown in [Figure 20 on page 115](#), a crack has a fractured front (*crack tip* or *crack front*), which is the end of two open cracked surfaces. If the stress intensity exceeds the material toughness, the cracked surfaces extend, thereby releasing the strain energy.

When the J-integral value reaches the threshold value:

$$J = J_c \quad (70)$$

the crack becomes unstable and starts to propagate. The critical energy release rate J_c is usually obtained through experiments.

The J-integral value gives a measure of stress intensity at the crack tip, or crack front, for linear elastic materials and elastic-plastic materials under small scale yielding: The size of the plastic zone around the crack tip or crack front is much smaller than the size of the crack [\[4\]\[5\]](#). If J-integral must be calculated for elastic-plastic materials with cracks, use [Deformation Plasticity on page 58](#) to model plastic material behavior.

The J-integral value changes with time for viscoelastic and viscoplastic materials. At a shorter time after application of load, the value near the crack tip is close to that of purely elastic material. As stresses and strains change with time, the J-integral value also changes depending on the solve time for stress analysis. For viscoelastic materials, the reference elastic value at any time can be obtained by computing the J_v -integral (see [Jv-Integral on page 118](#)), which differs from the initial elastic value due to relaxation of stresses. For viscoplastic materials, the $C(t)$ -integral (see [C\(t\)-Integral on page 118](#)) should be used to obtain a time-dependent value that becomes path independent after relaxation of stresses.

Use the `j_integral` command (see [j_integral on page 475](#)) to calculate the J-integral value for cracks in linear elastic materials in 2D and 3D structures under thermal or mechanical loading. This command calculates the J-integral at the crack tip of the specified crack and returns its value in units of N/m. The return value can be used in a Tcl expression. Use this command after a simulation or after loading a TDR file that contains the crack simulation results.

NOTE J-integral calculation requires the normal direction vector of the crack. Use the `normal` parameter to specify the crack surface normal in the `crack` command before specifying the `j_integral` command.

Calculation of J-integral depends on the value of stress and strain-energy density in the region around the crack tip. As shown in [Figure 21 on page 117](#), the region around the crack tip has very high stresses and stress gradients. Analytic solutions for stress fields around the crack tip show that stresses increase asymptotically in the radial direction as you approach the crack tip.

To model such high stress gradients in Sentauros Interconnect, a fine mesh must be used around the crack tip.

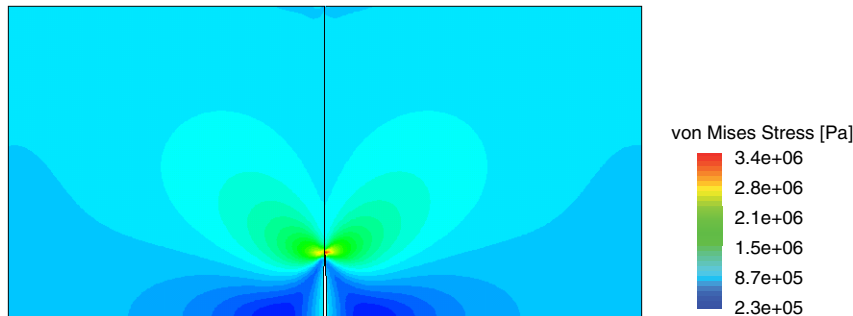


Figure 21 von Mises stress in a 2D structure with crack

Stress gradients in the immediate neighborhood of the crack tip are generally too large to be modeled accurately with a refined mesh. Therefore, the J-integral calculation over elements close to the crack tip are not very accurate. More accurate values are obtained over larger domains that include elements farther away from the crack tip where the stresses are smoother. This is possible due to the path independency of the J-integral expression.

Use the number parameter of the `j_integral` command to specify the number of domains over which the J-integral value will be calculated. The first domain includes the elements surrounding the crack tip. The second domain includes the first domain and the elements surrounding the first domain. Each subsequent domain is built in the same way as the second domain.

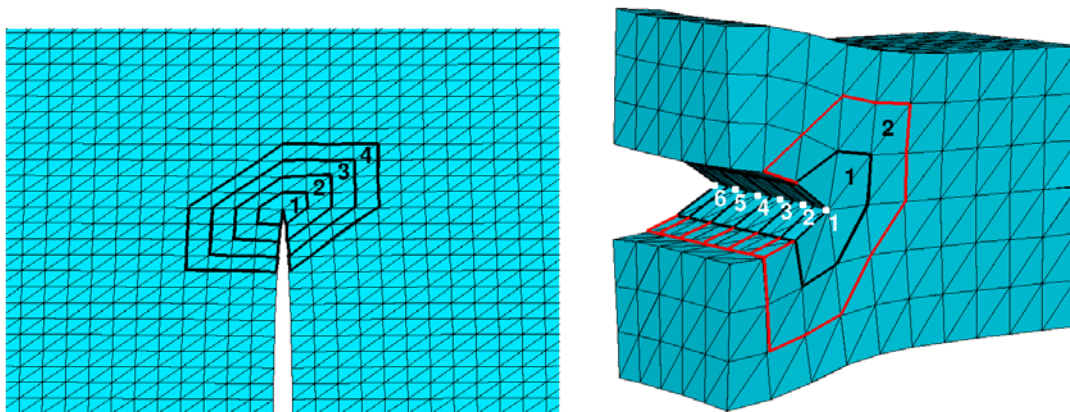


Figure 22 Domains for J-integral calculation in a (left) 2D structure and (right) 3D structure

Figure 22 shows the domains around the crack tip and crack front in 2D and 3D structures. In 3D structures, J-integral values are computed at each node on the crack front. Use the `x`, `y`, and `z` parameters to specify the coordinates of the node on the crack front. As seen in Figure 22, domains in a 3D structure, for nodes in the interior of the crack front, include half the element thickness on either side of the node. Domains for nodes on either end of the crack front include

4: Mechanics Postprocessing

C(t)-Integral

only half the element thickness on one side. An average value of J-integral over all the nodes on the crack front also is computed for each domain and is reported along with the nodal value.

By default, J-integral is computed for the first domain. When the `number` parameter is set to a value greater than 1, J-integral is calculated for all the domains starting from the first domain.

C(t)-Integral

For viscoplastic materials (see [Viscoplastic Materials on page 59](#)), a time-dependent contour integral called *C(t)-integral* is used to characterize the amplitude of stresses around crack tips [6][7][8]. It is obtained by replacing strains with strain rates and displacements with velocities in the expression for J-integral:

$$C(t) = \lim_{\Gamma \rightarrow 0} \int_{\Gamma} \left(\dot{W} \delta_{1j} - \sigma_{ij} \frac{\partial \dot{u}_i}{\partial x_j} \right) n_j ds \quad (71)$$

where $\dot{W} = \int_0^{\dot{\epsilon}_{ij}} \sigma_{ij} d\dot{\epsilon}_{ij}$ is strain energy rate density.

For a short time after load is applied, creep strains dominate other strains in a small region around the crack tip. During this time C(t)-integral is path dependent. Its value changes at points away from the neighborhood of crack tip. Over time, as the stresses relax, creep strains spread through the material away from the crack tip. As time $t \rightarrow \infty$, steady-state conditions prevail everywhere so C(t)-integral becomes path independent. The path independent, constant value is referred to as C^* .

C(t)-integral is calculated in a manner similar to J-integral, so a fine mesh must be defined around the crack tip to capture the high stress gradient. Use the `CtIntegral` parameter of the `j_integral` command to calculate the C(t)-integral value in units of N/ms. In addition, use the `number` parameter to specify the number of domains over which to calculate the value. At a shorter time after loading, the value should be computed over domains close to the crack tip; at a longer relaxation time, more domains should be used to obtain a path-independent value.

Jv-Integral

For viscoelastic material (see [Viscoelastic Materials on page 47](#)), the J-integral value changes with time. At any time after applying the load, the value depends on stresses and strains at that time.

To obtain the value in a reference elastic body with the same stresses, use the correspondence principle [8][9]:

$$J_v = \lim_{\Gamma \rightarrow 0} \int_{\Gamma} \left(W^e \delta_{1j} - \sigma_{ij} \frac{\partial u_i^e}{\partial x_j} \right) n_j ds \quad (72)$$

where:

- $W^e = \int_0^{\varepsilon_{ij}^e} \sigma_{ij} d\varepsilon_{ij}^e$ is the pseudo-strain energy density.
- $\varepsilon_{ij}^e = \frac{\sigma_{ij}}{E_R} = \frac{1}{E_R} \int_0^t E((t-\tau), t) \frac{\partial \varepsilon_{ij}}{\partial \tau} d\tau$ is the pseudo-strain.
- $u_i^e = \frac{1}{E_R} \int_0^t E((t-\tau), t) \frac{\partial u_i}{\partial \tau} d\tau$ is the pseudo-displacement.
- $E(t)$ is the relaxation modulus.
- E_R is the reference elastic modulus.

J_v -integral is calculated in a manner similar to J-integral with a fine mesh around the crack tip to capture the high stress gradient. Use the `JvIntegral` parameter of the `j_integral` command to calculate the J_v -integral value in units of N/m. In addition, use the `number` parameter to specify the number of domains over which to calculate the value. At a shorter time after loading, the value should be computed over domains close to the crack tip; at a longer relaxation time, more domains should be used to obtain a path-independent value.

References

- [1] Y. Kanda, "A Graphical Representation of the Piezoresistance Coefficients in Silicon," *IEEE Transactions on Electron Devices*, vol. ED-29, no. 1, pp. 64–70, 1982.
- [2] C. F. Shih, B. Moran, and T. Nakamura, "Energy release rate along a three-dimensional crack front in a thermally stressed body," *International Journal of Fracture*, vol. 30, no. 2, pp. 79–102, 1986.
- [3] J. R. Rice, "A Path Independent Integral and the Approximate Analysis of Strain Concentration by Notches and Cracks," *Journal of Applied Mechanics*, vol. 35, no. 2, pp. 379–386, 1968.
- [4] J. W. Hutchinson, "Singular Behaviour at the End of a Tensile Crack in a Hardening Material," *Journal of the Mechanics and Physics of Solids*, vol. 16, no. 1, pp. 13–31, 1968.
- [5] J. R. Rice and G. F. Rosengren, "Plane Strain Deformation Near a Crack Tip in a Power-Law Hardening Material," *Journal of the Mechanics and Physics of Solids*, vol. 16, no. 1, pp. 1–12, 1968.

4: Mechanics Postprocessing

References

- [6] H. Riedel and J. R. Rice, “Tensile Cracks in Creeping Solids,” *Fracture Mechanics: Twelfth Conference*, ASTM STP 700, American Society for Testing Materials, pp. 112–130, 1980.
- [7] J. L. Bassani and F. A. McClintock, “Creep Relaxation of Stress Around a Crack Tip,” *International Journal of Solids and Structures*, vol. 17, pp. 479–492, 1981.
- [8] T. L. Anderson, *Fracture Mechanics*, 3rd Edition, CRC Press, 2005.
- [9] R. A. Schapery, “Correspondence principles and a generalized J integral for large deformation and fracture analysis of viscoelastic media,” *International Journal of Fracture*, vol. 25, pp. 195–223, 1984.

This chapter discusses the thermal analysis capability of Sentaurus Interconnect.

Overview

In a backend simulation tool, thermal analysis is a key requirement because the temperature distribution in a structure is an important entity. Moreover, many parameters used in other physical models depend on local temperature.

Thermal analysis, however, is optional in Sentaurus Interconnect. When thermal analysis is switched off during a simulation, the temperature does not evolve during the simulation but remains in whatever initial state it was at the start of the simulation.

To switch on thermal analysis, use (see [mode on page 518](#)):

```
mode thermal
```

Thermal Model

The thermal analysis available in Sentaurus Interconnect is based on the time-dependent nonlinear heat equation with distributed heat sources:

$$\rho C_p \frac{\partial}{\partial t} T - \nabla \cdot (\kappa \nabla T) = P_f + P_{JH} \quad (73)$$

where T is the temperature in kelvin. The first source term on the right side is a fixed power density term, and the second term is a power density term associated with Joule heating in the current-carrying regions.

The thermal conductivity, κ , is by default a simple floating-point number, although you can override this value with a general expression of solution variables for a given material. ρ and C_p are the mass density and specific heat capacity:

```
pdbSet <material> MassDensity <n>
```

```
pdbSet <material> SpecificHeatCapacity <string_expression>
```

5: Thermal Analysis

Thermal Model

The material parameter of main relevance for the state thermal analysis is the isotropic thermal conductivity, κ . For each material type, a default value for this parameter has been provided in the parameter database. To modify this parameter, use the command:

```
pdbSet <material> Temperature Conductivity <string_expression>
```

For example:

```
pdbSet Copper Temperature Conductivity { 4.01 - 0.02*(Temperature-300) }
```

If the thermal conductivity is anisotropic, it can be set with the command:

```
pdbSet <material> Temperature Aniso.Conductivity { X <string_expression>
                                                Y <string_expression>
                                                Z <string_expression> }
```

where:

- X, Y, and Z are the simulation domain x, y, and z, respectively.
- <string_expression> is any valid Alagator thermal conductivity expression in that direction.

If thermal conductivity is set to zero in any direction, the value available for isotropic thermal conductivity is used in that direction.

Individual components of the anisotropic thermal conductivity can be set by:

```
pdbSet <material> Temperature Aniso.Conductivity X <string_expression>
pdbSet <material> Temperature Aniso.Conductivity Y <string_expression>
pdbSet <material> Temperature Aniso.Conductivity Z <string_expression>
```

Setting any component of the anisotropic conductivity automatically activates the anisotropic model. To switch off the anisotropic conductivity model, all of its components must be set to zero.

NOTE For thermal analysis to converge, each noncontact material type in the structure must have a well-defined thermal conductivity.

The Joule heating term, P_{JH} , is expressed as:

$$P_{JH} = \sigma|E|^2 \quad (74)$$

where σ is the material-specific electrical conductivity, and E is the electrical field in a current-carrying region. To modify the electrical conductivity, use the command:

```
pdbSet <material> Potential Conductivity <string_expression>
```


The Joule heating term is switched off by default. To switch it on, use:

```
pdbSet <material> Temperature JouleHeating 1
```

When the Joule heating term is switched on, the current Laplace equation (see [Eq. 83, p. 131](#)) also must be solved. To solve the thermal equation and the current Laplace equation simultaneously, use (see [mode on page 518](#)):

```
mode thermal current
```

For more details on configuring electrical current analysis, see [Chapter 6 on page 131](#).

To define a fixed power-density term for any noncontact region, use the `supply` command (see [supply on page 647](#)), for example:

```
supply region.name= Copper_1 power.density= 2.0e8<W/cm3>
```

NOTE The steady-state heat equation is solved if there are no time-dependent boundary conditions (for example, voltage ramp), or if Joule heating or circuit analysis is switched off.

Boundary Conditions

To ensure that the heat equation has a well-defined steady-state solution, the problem must be properly constrained. This is done by applying thermal boundary conditions to a set of contacts (thermodes). To establish a fixed temperature reference in the system, at least one of the boundary conditions must be of constant temperature type. Additional boundary conditions can be either the constant temperature type or the constant heat flux type.

For all noncontact outer boundaries, by default simple reflective boundary conditions apply, while flux-conserving continuous boundary conditions apply at material interfaces:

$$T_1(x \rightarrow i) = T_2(i \leftarrow x) \quad (75)$$

$$\kappa_1 \nabla T|_{n1} = \kappa_2 \nabla T|_{n2} \quad (76)$$

where:

- Indices 1 and 2 indicate the two sides of the interface i .
- n indicates the component of the temperature gradient normal to the interface.
- κ is the thermal conductivity.

Noncontact boundary conditions are set up automatically by the system. The outer boundary conditions are user configurable and can be set to a nonreflective boundary condition.

5: Thermal Analysis

Thermal Model

Bottom Boundary Condition

At the bottom, the boundary condition depends on whether the thermal resistor is attached. The thermal resistor can be switched on using the command:

```
pdbSet <material> AttachThermalResistor 1
```

where <material> is the bottom material name. If the thermal resistor is switched on, the emission flux at bottom is calculated by:

$$F = -\left(\frac{\kappa}{t_w - x_{bot}}\right)(T - T_0) \quad (77)$$

where:

- t_w is the wafer thickness.
- x_{bot} is the bottom coordinate of a simulation structure.
- T_0 is the environment temperature.

To set the wafer thickness and environment temperature, use:

```
pdbSet WaferThickness <n>  
pdbSet Env.Temp <n>
```

If the thermal resistor is not switched on, the emission flux at the bottom becomes:

$$F = -H_T(T - T_0) \quad (78)$$

where H_T is the heat sink transfer rate.

To specify the heat sink transfer rate, use:

```
pdbSet <material> HeatSinkTransfer <n>
```

By default, `HeatSinkTransfer` is set to zero for all materials.

Side Boundary Condition

At the sides, the flux is calculated by:

$$F = -H_{ST}(T - T_0) \quad (79)$$

where H_{ST} is the side heat transfer rate.

To specify the side heat transfer rate, use:

```
pdbSet <material> SideHeatTransfer <n>
```

By default, `SideHeatTransfer` is set to zero for all materials.

Top Boundary Condition

At the top surface, that is, the gas interface, the boundary condition depends on whether the thermal resistor is attached. The thermal resistor can be switched on using the command:

```
pdbSet Gas AttachThermalResistor 1
```

If the thermal resistor is switched on, the emission flux at the top is calculated by:

$$F = -\left(\frac{\kappa}{t_w - x_{\text{top}}}\right)(T - T_0) \quad (80)$$

where x_{top} is the top coordinate of a simulation structure.

At the top surface, if the radiation heat transfer boundary condition is selected using:

```
pdbSet Gas RadiationHeatTransfer 1
```

the heat emission flux from the top material is given by:

$$F = -5.6703 \cdot 10^{-12} \cdot \text{Emissivity} \cdot (T^4 - T_0^4) \quad (81)$$

where `Emissivity` is the emissivity rate. To specify the emissivity rate, use:

```
pdbSet <material> Emissivity <n>
```

At the top interface, if the convective heat transfer boundary condition is selected using:

```
pdbSet Gas ConvectiveHeatTransfer 1
```

the emission flux from the top material is given by:

$$F = -H_{\text{TT}}(T - T_0) \quad (82)$$

where H_{TT} is the top heat transfer rate.

To specify the top heat transfer rate, use:

```
pdbSet <material> TopHeatSinkTransfer <n>
```

By default, `TopHeatSinkTransfer` is set to zero for all materials.

5: Thermal Analysis

Thermal Model

It is also possible to set the heat emission flux boundary condition (see [Eq. 78, p. 124](#)) at any thermal contact using the `supply` command (see [supply on page 647](#)).

For example, the following command will set a heat emission boundary condition at the contact named `therm1` with a rate of $1 \times 10^5 \text{ W/cm}^2\text{K}$ and the environment temperature T_0 of 400 K. If the environment temperature is not given, it will be read from the PDB:

```
supply contact.name= therm1 heat.rate= 1e5 temperature= 400<K>
```

Defining Thermal Contacts

To define thermal contacts, use the `contact` command (see [contact on page 381](#)), for example:

```
contact box xlo=2<um> xhi=3<um> ylo=0<um> yhi=0<um> Silicon name=therm1 \  
  sidewall  
contact box xlo=7<um> xhi=7<um> ylo=0<um> yhi=10<um> Silicon name=therm2 \  
  bottom
```

Applying Boundary Conditions to Thermal Contacts

To apply boundary conditions to the thermal contacts, use the `supply` command (see [supply on page 647](#)), for example:

```
supply contact.name = therm1 temperature=285<C>  
supply contact.name = therm2 power=1.0e-5<W>
```

The first command sets a fixed temperature on contact `therm1`, and the second command injects a constant power at contact `therm2`.

NOTE To establish an absolute temperature reference in the system, you must specify at least one constant temperature boundary condition for thermal analysis.

Ramped-up temperature sources can be specified using the `temp_ramp` command (see [temp_ramp on page 655](#)), for example:

```
temp_ramp name=Tramp1 time=0.06<s> temp=700<V> ramprate=(300.0/0.06)<C/s>  
temp_ramp name=Tramp1 time=0.1<s> temp=1000<C>  
supply contact.name=elect1 temp.ramp=Tramp1
```

The first two commands set up a temperature ramp where the temperature increases from 700°C to 1000.0°C in $6 \times 10^{-2} \text{ s}$ and remains at 1000.0°C for $1 \times 10^{-1} \text{ s}$. The third command uses the specified temperature ramp on the contact `elect1`.

Thermal Submodeling

Thermal submodeling is a technique to obtain accurate thermal results in a local region with mesh refinements, while the full model can be simulated with a coarser mesh. This technique extracts the temperature from the global structure, interpolates the field on to the local refined model, and applies the temperature to the outer surfaces of the local model as boundary conditions using the `mode` command (see [mode on page 518](#)):

```
mode thermal.global.model=<c>
```

where `<c>` is the name of the global TDR file. The temperature field from `<c>` will be applied to the outer surfaces of the local model as boundary conditions during the execution of the next `solve` command. If the global model is not provided, the submodeling simulation will continue using the current temperature profile at the execution of the next `solve` command. This allows users to define temperature profiles using other commands such as `select`. For example:

```
mode thermal.global.model=Chip
```

will load the temperature field from a global model file named `Chip` as the outer boundary condition during the next `solve` command, and the thermal submodeling equations will be solved.

The following command will not load the temperature field but will use the current temperature field as the outer boundary condition at the next `solve` command, and the thermal submodeling equations will be solved:

```
mode thermal.global.model
```

NOTE In thermal submodeling mode, no thermal contacts for the outer boundary can be defined. In addition, no bottom, top, or sidewall boundary conditions with heat transfer are allowed.

Thermal Resistance Matrix

Sentaurus Interconnect calculates the thermal resistance matrix for the listed contacts. If no contact information is given, conductors are treated as contacts. The resistances are calculated where resistances may be required. The mathematical relationship between the thermal resistances expressed in the schematic and corresponding short-circuit thermal resistances calculated by Sentaurus Interconnect.

Sentaurus Interconnect calculates the thermal resistance matrix among electrodes.

5: Thermal Analysis

Thermal RC Network

To calculate thermal resistance matrix, the `mode` command (see [mode on page 518](#)) must be used, for example:

```
mode thermal.resistance
```

See [Appendix B on page 683](#) for detailed explanation.

NOTE Thermal resistance matrix analysis requires a Raphael™ license.

Thermal RC Network

Sentaurus Interconnect performs transient thermal analysis to calculate the thermal RC networks for the listed contacts.

To calculate the values of the elements of the thermal RC networks, the `mode` command must be used (see [mode on page 518](#)), for example:

```
mode thermal.RC.network= {"excitation.contact" "ambient.contact" }
```

See [Appendix B on page 683](#) for detailed explanation.

NOTE Thermal RC network analysis requires a Raphael license.

Output Data Fields

[Table 8](#) lists the data fields pertinent to thermal analysis.

Table 8 Data fields associated with thermal analysis

Field name	Element/Node	Description	Unit
Temperature	Node	Final temperature profile	K
PrevTemperature	Node	Previous temperature profile	K
TemperatureGradient	Node	Final temperature gradient	K/cm
HeatFlux	Node	Heat flux	W/cm ²
TotalHeat	Node	Right hand side of Eq. 73	W/cm ³
JouleHeat	Node	Joule heating	K/cm ³

In the trivial case where thermal analysis is switched off, the temperature profiles for `Temperature` and `PrevTemperature` are identical.

Example: 2D Thermal Analysis With Joule Heating

The following self-contained example runs a thermal analysis on a simple structure containing a current-carrying straight copper wire surrounded by oxide (see [Figure 23 on page 130](#)). Fixed temperature boundary conditions are applied to the sidewall and bottom contacts, and the Joule heating model is switched on to allow the wire to generate heat. Finally, the example demonstrates how to create a thermal hotspot map in a new data field that is exported to a file together with other simulations results.

```
# Set thermal conductivities [W/cm*K] (:approx value)
pdbSet Copper Temperature Conductivity 3.7e4

# Define thermal contacts and supplies
contact box xlo=10<um> xhi=10<um> ylo=0<um> yhi=10<um> Oxide name=thermBot \
  bottom
contact box xlo=0<um> xhi=3<um> ylo=0<um> yhi=0<um> Oxide name=thermSide \
  sidewall
supply contact.name=thermBot temperature=200<C>
supply contact.name=thermSide temperature=200<C>

# Define electrical contacts and supplies
contact box xlo=4.5<um> xhi=5.5<um> ylo=0<um> yhi=0<um> \
  Copper name=anode sidewall
contact box xlo=4.5<um> xhi=5.5<um> ylo=10<um> yhi=10<um> \
  Copper name=cathode sidewall
supply contact.name=anode voltage=2.0e-2<V>
supply contact.name=cathode voltage=0<V>

# Define the structure
line clear
line x location=0 spacing=0.1 tag=OxideTop
line x location=4.5 spacing=0.1 tag=wireTop
line x location=5.5 spacing=0.1 tag=wireBot
line x location=10 spacing=0.1 tag=OxideBot

line y location=0 spacing=0.1 tag=OxideLeft
line y location=10 spacing=0.1 tag=OxideRight

region Oxide xlo=OxideTop xhi=wireTop ylo=OxideLeft yhi=OxideRight
region Copper xlo=wireTop xhi=wireBot ylo=OxideLeft yhi=OxideRight
region Oxide xlo=wireBot xhi=OxideBot ylo=OxideLeft yhi=OxideRight

init !DelayFullD

# Switch on both electrical current analysis and thermal analysis as
# well as the Joule heating model, which will couple the heat equation
# and the current Laplace equation.
mode thermal current
pdbSet Copper Temperature JouleHeating 1
```

5: Thermal Analysis

References

```
# Set initial temperature in degree Celsius
SetTemp [expr 300.0-273.156]

# Start simulation
solve info=2

# Set up a thermal failure criterion and store the hotspot areas in a
# separate field. Temperature unit is K.
select z="Temperature > 527" name=MaxTemperatureExceeded store

struct tdr=joule_heating_2d_out !Gas
```

The final temperature distribution for the example is shown in [Figure 23](#).

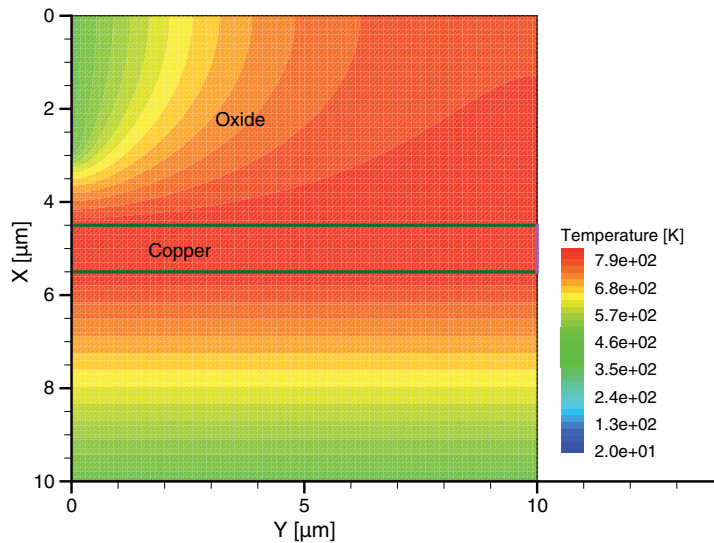


Figure 23 Final temperature distribution for example with Joule heating source

References

- [1] J. R. Cannon, *The One-Dimensional Heat Equation*, Encyclopedia of Mathematics and Its Applications, vol. 23, Menlo Park, California: Addison-Wesley, 1984.

CHAPTER 6 Electrical Current Analysis

This chapter discusses the electrical current analysis capability of Sentaurus Interconnect.

Overview

In an interconnect reliability analysis tool, electrical analysis is a key requirement because the electric potential and the electric field are important entities in their own right.

The types of electrical analysis available in Sentaurus Interconnect are:

- Electrical current analysis
- Electrostatic analysis (see [Chapter 7 on page 143](#))

Both analyses are performed as steady-state problems and are mutually exclusive; in other words, only one of these two types of analysis can be performed in one simulation.

Electrical current analysis calculates the electric potential and the electric field in and around the current-carrying conducting regions biased between two or more contacts.

To switch on electrical current analysis, use (see [mode on page 518](#)):

```
mode current
```

Electrical Current Model

Electrical current analysis in Sentaurus Interconnect is based on the continuity equation for conducting media [1]:

$$\nabla \cdot (\sigma \nabla \phi) = \nabla \cdot J_D \quad (83)$$

$$J_D = -\frac{\partial}{\partial t}(\epsilon \nabla \phi) \quad (84)$$

6: Electrical Current Analysis

Electrical Current Model

where:

- ϕ is the electric potential
- J_D is the displacement current.
- ϵ is the permittivity (see [Electrostatic Model on page 143](#)).

The displacement currents are only added when there is ramping boundary conditions (see [current_ramp on page 393](#) and [voltage_ramp on page 679](#)). σ is the electrical conductivity, usually dependent on the local temperature. A simple temperature derating model is used for most materials by default:

$$\sigma(T) = \frac{\sigma_{300}}{(1.0 + \alpha[T - 300])} + \sigma_{qmc} \quad (85)$$

where:

- σ_{300} is the electrical conductivity at 300 K.
- α is a linear temperature coefficient.
- σ_{qmc} represents the quantum-mechanical correction. It is optional and is zero by default. See [Wire Size–Dependent Electrical Conductivity on page 133](#).

These parameters can be modified with the respective commands:

```
pdbSet <material> Potential SigmaRef <n>
pdbSet <material> Potential Alpha <n>
```

The two parts in [Eq. 85](#) can be overridden with the following commands:

```
pdbSet <material> Potential Conductivity <string_expression>
pdbSet <material> Potential Conductivity.QMC <string_expression>
```

NOTE For current analysis to converge, the material type of each noncontact region in the structure for which the current analysis is switched on must have a well-defined electrical conductivity.

For efficiency reasons, the current Laplace equation (see [Eq. 83](#)) is only solved in metal regions by default. Whenever you want to solve for the electric potential and the electric field in dielectric or semiconducting regions, switch on the solution in such material types with:

```
pdbSet <material> Conductor 1
```

Wire Size–Dependent Electrical Conductivity

Two semiempirical conductivity models for narrow wires are available to calculate the quantum-mechanical correction σ_{qmc} :

- Proximity field–based model
- Nearest distance–based model

Proximity Field–Based Model

This model is based on the proximity effect. In this model, the resistivity ρ_i at the i -th node of a conductor is expressed as:

$$\frac{1}{\sigma_{qmci}} = \rho_i = \rho_b \cdot \frac{\sum_{k \text{ } |r_k| < \beta \lambda_q} V_k \cdot e^{-\frac{|r_k|}{\lambda_q}}}{\sum_k \frac{V_k}{\rho_{ck}} \cdot e^{-\frac{|r_k|}{\lambda_q}}} = \text{ProximityField} \quad (86)$$

where:

- σ_{qmci} is the quantum-mechanical correction at the i -th node.
- V_k is the volume surrounding the k -th node.
- r_k is the distance from the k -th node to the i -th node.
- ρ_b is the bulk resistivity correction. The default is 28.5.
- λ_q is a control parameter for the range of the proximity effect. The default is 3.75.
- ρ_{ck} is set to 1 if the node is inside the conductor, or it is set to the boundary resistivity correction ρ_{bc} if the node is outside the conductor. The default for ρ_{bc} is 1000.
- β is the proximity range factor that controls the size of the summation. The default is 5.

These parameters must be calibrated with the known empirical resistivity model or experimental data. You can modify the values for ρ_b , ρ_{bc} , λ_q , and β with their respective commands:

```

pdbSetDouble <material> Potential Bulk.Resistivity.Correction <n>
pdbSetDouble <material> Potential Boundary.Resistivity.Correction <n>
pdbSetDouble <material> Potential Proximity.Range <n>
pdbSetDouble <material> Potential Proximity.Range.Factor <n>

```

6: Electrical Current Analysis

Electrical Current Model

For example, to set these values for copper, you add the following settings to the command file:

```
pdbSetDouble Copper Potential Bulk.Resistivity.Correction 28.5
pdbSetDouble Copper Potential Boundary.Resistivity.Correction 1000
pdbSetDouble Copper Potential Proximity.Range 3.75
pdbSetDouble Copper Potential Proximity.Range.Factor 5.0
```

To activate the proximity field-based model, you must specify the keyword `ProximityField` for the `Conductivity.QMC` parameter. A field named `ProximityField` will be calculated according to [Eq. 86, p. 133](#) for each node. For example:

```
pdbSet Copper Potential Conductivity.QMC "1e7/ProximityField"
```

The default units of all variables in this model are extracted assuming lengths in nanometers. Therefore, `1e7` must be used to convert the units from nm to cm.

The total conductivity is the sum of `Conductivity` and `Conductivity.QMC` ([Eq. 85, p. 132](#)). By default, `Conductivity` is not 0 and `Conductivity.QMC` is 0. Therefore, when using `Conductivity.QMC` to define the model, you must set `Conductivity` properly.

In addition, if the total conductivity is defined only by `Conductivity.QMC`, then set `Conductivity` to 0 to overwrite its default value:

```
pdbSet <material> Potential Conductivity 0
```

The `ProximityField` is calculated according to regions. For each conductor region, any node outside of the conductor region is considered to be in a dielectric region. Therefore, adjacent conductor regions are not considered to be conductors.

In certain cases, such as when the conductor region is close to or on the boundary, the conductor region may need to be buried in dielectric to achieve the correct proximity effect. To do this, the structure can be first extended to ensure the conductor regions have a surrounding dielectric region of at least $\beta\lambda_q$ thickness. The structure will be restored after the field `ProximityField` is calculated. To enable this extension, specify the following:

```
pdbSetBoolean Copper Potential Extend.Structure 1
```

By default, `Extend.Structure` is false (0), and the summation in [Eq. 86](#) is calculated only up to the boundary when a conductor is near the boundary.

Nearest Distance–Based Model

This model is based on the nearest distance to the conductor surface [2]. In this model, the resistivity is calculated using:

$$\frac{1}{\sigma_{\text{qmc}}(d)} = \rho(d) = \rho_b + \frac{\rho_q}{\cosh\left(\frac{d}{\lambda_q}\right)} \quad (87)$$

where:

- $\sigma_{\text{qmc}}(d)$ is the quantum-mechanical correction.
- d is the nearest distance to the conductor surface from a point within the conductor.
- ρ_b is the bulk resistivity.
- ρ_q is a control parameter for the quantum-mechanical effect.
- λ_q is a control parameter for the range of the quantum-mechanical effect.

This model is activated using the `pdb` parameter `Conductivity.QMC`. When this parameter is set, a `NearestDistance` field (in cm) will be calculated for each conductor region. You can then use this field to define the nearest distance–based model according to Eq. 87. For example:

```
proc Cosh1 { a } {
  return "(2*exp(-$a)/(1+exp(-2*$a)))"
}
pdbSet Copper Potential Conductivity.QMC
      "1e7/(30.5+52.5*[Cosh1 (NearestDistance*1e7/3.5)])"
```

Again, the default units of all variables in this model are extracted assuming lengths in nanometers. Therefore, `1e7` must be used to convert the units from nm to cm.

In addition, if the total conductivity is defined only by `Conductivity.QMC`, then set `Conductivity` to 0 to overwrite its default value:

```
pdbSet <material> Potential Conductivity 0
```

Grain Size–Dependent Electrical Conductivity

Mayadas and Shatzkes [3] proposed a model of electron scattering due to a distribution of planar potentials, that is, grain boundaries, so that smaller grains cause more electron scattering, leading to greater resistivity:

$$\sigma' = f_{MS} \cdot \sigma_0 \quad (88)$$

6: Electrical Current Analysis

Electrical Current Model

$$f_{MS} = 3\left(\frac{1}{3} - \frac{\alpha}{2} + \alpha^2 - \alpha^2 \ln\left(1 + \frac{1}{\alpha}\right)\right) \quad (89)$$

$$\alpha = \frac{l_0 R}{L_g(1-R)} \quad (90)$$

where:

- R is the coefficient for the reflection from a single plane.
- l_0 is the background mean free path that is calculated by:

$$l_0 = \sigma_0 \Omega m_e e^{-2} c \sqrt{\frac{2 \cdot E_F}{m_e c^2}} \quad (91)$$

where Ω is the atomic volume, and E_F is the Fermi energy. Ω is $1/\text{LatticeDensity}$.

R and E_F are specified on a material as follows:

```

pdbSet <material> GB.Reflectivity <n> ;# unitless
pdbSet <material> Fermi.Energy <n> ;# eV

```

Boundary Conditions

To ensure that the current Laplace equation (see [Eq. 83, p. 131](#)) has a well-defined steady-state solution, the problem must be properly constrained. This is performed by applying electrical boundary conditions to two or more contacts defined on the regions that constitute the current-carrying path.

To establish a fixed reference potential in the system, at least one of these boundary conditions on the contacts must be of constant voltage (Dirichlet) type. Additional boundary conditions on the contacts can be either of constant current type, constant current density type, or constant voltage type.

For all noncontact outer boundaries, simple reflective boundary conditions apply, while the following charge flux-conserving continuous boundary conditions apply at noncontact material interfaces:

$$\sigma_1 E_{1n} = \sigma_2 E_{2n} \quad (92)$$

$$E_{1t} = E_{2t} \quad (93)$$

where E_n and E_t designate the normal component and the tangential component of the electric field, respectively.

Noncontact boundary conditions are established automatically by the system and are not user configurable.

Defining Electrical Contacts

Use the `contact` command to define electrical contacts (see [contact on page 381](#)), for example:

```
contact box xlo=2<um> xhi=3<um> ylo=0<um> yhi=0<um> Copper name=elect1 \  
    sidewall  
  
contact box xlo=7<um> xhi=7<um> ylo=0<um> yhi=10<um> Copper name=elect2 \  
    bottom
```

Applying Boundary Conditions to Electrical Contacts

Use the `supply` command to apply boundary conditions for electrical current analysis to contacts (see [supply on page 647](#)), for example:

```
supply contact.name=elect1 voltage=1.0e-3<V>  
supply contact.name=elect2 current.density=1.0e5<A/cm2>  
supply contact.name=elect3 current=1.0e-4<A>
```

- The first command specifies a constant voltage type boundary condition on contact `elect1`.
- The second command specifies a constant current density type boundary condition on contact `elect2`.
- The third command specifies a constant current type boundary condition on contact `elect3`.

NOTE To establish an absolute reference potential in the system, at least one constant voltage (Dirichlet) type boundary condition must be specified when doing current analysis. The presence of a constant voltage boundary condition on at least one contact also ensures Kirchhoff compliance, absorbing any excess charge injected into the system by constant current or fixed current density boundary conditions set on the other contacts.

6: Electrical Current Analysis

Resistance Matrix

NOTE When applying the constant current density–type boundary condition to regional contacts, you should use the `adjacent.material` option of the `contact` command (see [contact on page 381](#)) to specify the type of material interfaces that the current is allowed to traverse. For example, to specify that current is to be injected only into copper regions that abut the regional contact affiliated with region `Metal_1`, define the contact as:

```
contact region=Metal_1 adjacent.material=Copper
```

Ramped-up current and voltage sources can be specified using `voltage_ramp` command (see [voltage_ramp on page 679](#)) and `current_ramp` command (see [current_ramp on page 393](#)), for example:

```
voltage_ramp name=Vramp1 time=1e-12<s> voltage=0<V> ramprate=(8.0/1e-12)<V/s>  
voltage_ramp name=Vramp1 time=1e-08<s> voltage=8<V>  
supply contact.name=elect1 voltage.ramp=Vramp1
```

The first two commands set up a voltage ramp where the voltage increases from 0 V to 8.0 V in 1×10^{-12} s and remains at 8.0 V for 1×10^{-8} s. The third command uses the specified voltage ramp on contact `elect1`.

NOTE Whenever a voltage ramp or current ramp is applied to a contact, the displacement currents are added automatically.

Resistance Matrix

Sentaurus Interconnect calculates the resistance matrix among electrodes. To calculate the resistance matrix, use the `mode` command (see [mode on page 518](#)), for example:

```
mode resistance
```

For a detailed explanation, see [Appendix B on page 683](#).

NOTE The resistance matrix analysis requires a Raphael user license.

Output Data Fields

Table 9 lists the data fields relevant to electrical current analysis and whether the field applies to elements or nodes.

Table 9 Data fields pertinent to electrical current analysis

Field name	Element/Node	Description	Unit
ElectrostaticPotential	Node	Electric potential	V
ElectricField	Node	Electric field	V/cm
ConductionCurrentDensity	Node	Current density	A/cm ²
ElectricalConductivity	Node	Electrical conductivity	A/Vcm

Example: 2D Electrical Current Analysis

The following self-contained example demonstrates electrical current analysis on a simple structure containing a copper T-junction embedded in oxide (see Figure 24 on page 141). Two constant current boundary conditions and one (mandatory) constant voltage boundary condition are applied to the sidewall and bottom contacts, respectively. The example demonstrates how to report currents and voltages on contacts as well as the maximum electric field and current density in the structure to the log file. Finally, the example shows how to generate a current-crowding hotspot map in a new data field that is exported to a file together with other simulations results.

```
# Comment out the next line to override the default
# electrical conductivity model for copper, for example:
# pdbSet Copper Potential Conductivity 0.25e5

# Define contacts
contact box xlo=4<um> xhi=5<um> ylo=0<um> yhi=0.0<um> sidewall name=C1 Copper
contact box xlo=4<um> xhi=5<um> ylo=10<um> yhi=10<um> sidewall name=C2 Copper
contact box xlo=10<um> xhi=10<um> ylo=5<um> yhi=6<um> bottom name=C3 Copper

# Bias the contacts: Two current BCs and one (mandatory) Dirichlet BC
supply contact.name=C1 current=5e-3<A>
supply contact.name=C2 current=2e-5<A>
supply contact.name=C3 voltage=0<V>

# Define and initialize the structure
line clear
line x location=0.0 spacing=0.2 tag=StructTop
line x location=4.0 spacing=0.2 tag=Wire1Top
line x location=5.0 spacing=0.2 tag=Wire1Bot
```

6: Electrical Current Analysis

Example: 2D Electrical Current Analysis

```
line x location=10.0 spacing=0.2 tag=StructBot
line y location=0.0 spacing=0.2 tag=StructLeft
line y location=5.0 spacing=0.2 tag=Wire2Left
line y location=6.0 spacing=0.2 tag=Wire2Right
line y location=10.0 spacing=0.2 tag=StructRight

region Oxide xlo=StructTop xhi=Wire1Top ylo=StructLeft yhi=StructRight
region Copper xlo=Wire1Top xhi=Wire1Bot ylo=StructLeft yhi=StructRight
region Oxide xlo=Wire1Bot xhi=StructBot ylo=StructLeft yhi=Wire2Left
region Copper xlo=Wire1Bot xhi=StructBot ylo=Wire2Left yhi=Wire2Right
region Oxide xlo=Wire1Bot xhi=StructBot ylo=Wire2Right yhi=StructRight

init !DelayFullD

# Specify current analysis
mode current

# Run the simulation
solve info=2

# Set up a current crowding failure criterion and store the
# hotspot areas in a separate field. The suffix '_mag' calculates
# the absolute value of the vector field 'ConductionCurrentDensity'.
select z="ConductionCurrentDensity_mag > 6.9e5" name=CurrDensHotspots store

# Save results
struct tdr=current_bc !Gas !interfaces

# Write max E-field and max current density to log file
set max_E_field [select name=ElectricField max]
set max_curr_dens [select name=ConductionCurrentDensity max]

LogFile "MAX(E-field) = $max_E_field V/cm"
LogFile "MAX(curr_dens) = $max_curr_dens A/cm^2"

# Write measured voltages and currents at contacts to log file
set v1 [contact Potential name=C1 voltage]
set c1 [contact Potential name=C1 current]
set v2 [contact Potential name=C2 voltage]
set c2 [contact Potential name=C2 current]
set v3 [contact Potential name=C3 voltage]
set c3 [contact Potential name=C3 current]

LogFile "MEASUREMENTS @ CONTACTS:"
LogFile "$v1\n$c1\n$v2\n$c2\n$v3\n$c3"
```

The electric potential and current density distributions for the example are shown in [Figure 24](#) and [Figure 25](#), respectively.

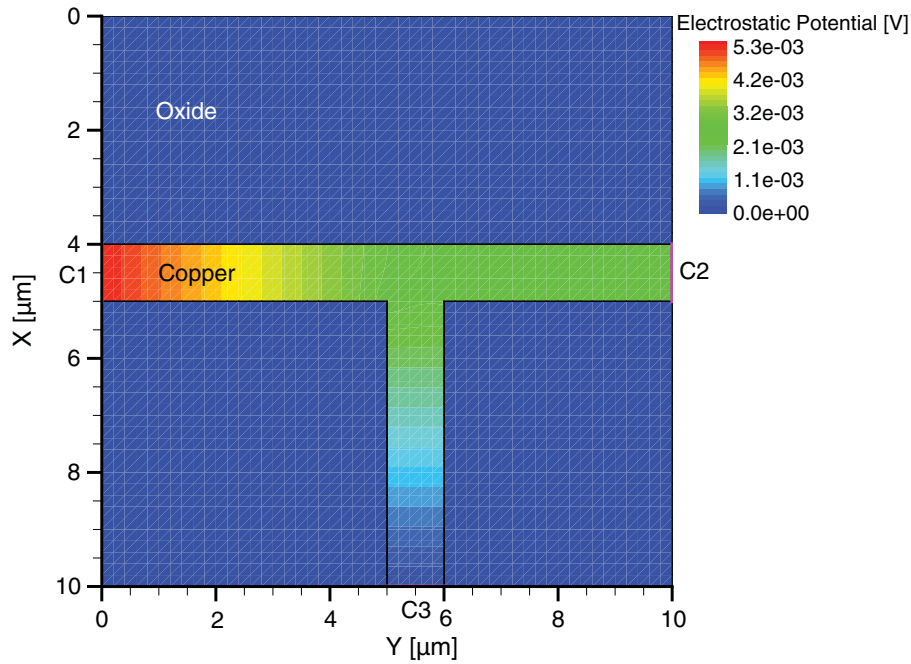


Figure 24 Electric potential distribution

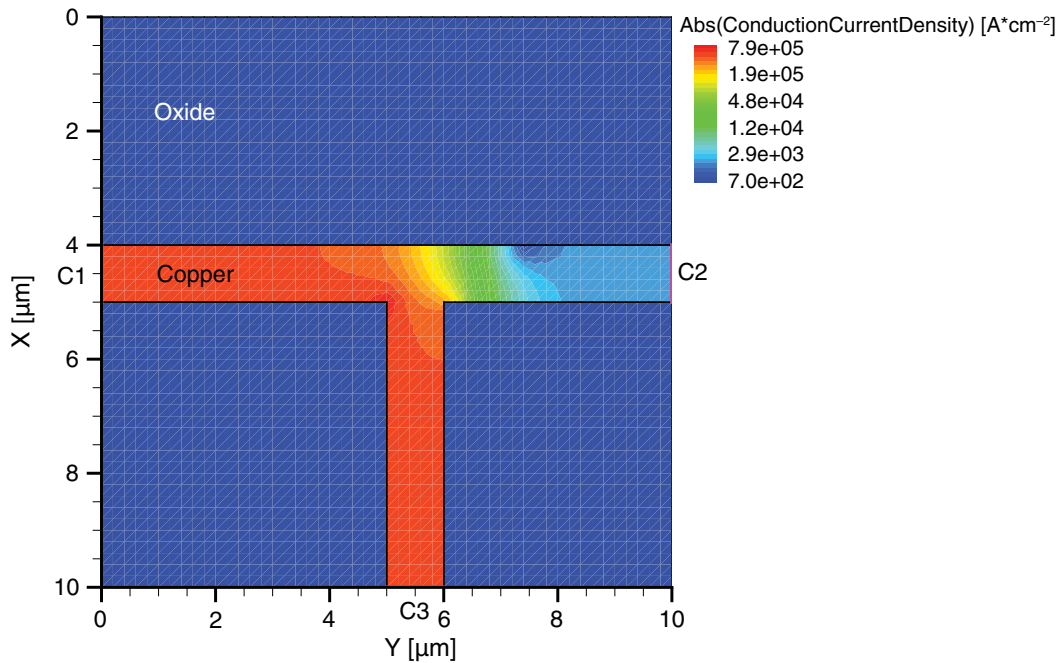


Figure 25 Current density distribution

References

- [1] J. R. Reitz, F. J. Milford, and R. W. Christy, *Foundations of Electromagnetic Theory*, Chapters 3–4, Reading, Massachusetts: Addison-Wesley, 4th ed., 1993.
- [2] I. Ciofi *et al.*, “Impact of Wire Geometry on Interconnect *RC* and Circuit Delay,” *IEEE Transactions on Electron Devices*, vol. 63, no. 6, pp. 2488–2496, 2016.
- [3] A. F. Mayadas and M. Shatzkes, “Electrical-Resistivity Model for Polycrystalline Films: the Case of Arbitrary Reflection at External Surfaces,” *Physical Review B*, vol. 1, no. 4, pp. 1382–1389, 1970.

CHAPTER 7 **Electrostatic Analysis**

The chapter discusses the electrostatic analysis capability of Sentaurus Interconnect.

Overview

In an interconnect reliability analysis tool, electrical analysis is a key requirement because the electric potential and the electric field are important entities in their own right.

The types of electrical analysis available in Sentaurus Interconnect are:

- Electrostatic analysis
- Electrical current analysis (see [Chapter 6 on page 131](#))

Both analyses are performed as steady-state problems and are mutually exclusive; in other words, only one of these two types of analysis can be performed in one simulation.

Electrostatic analysis calculates the static electric potential and the electric field in the dielectric regions (including gas) between a set of biased metal electrodes.

To switch on electrostatic analysis, use (see [mode on page 518](#)):

```
mode electrostatic
```

NOTE The calculation of electrostatic potential can be switched off in gas regions using the following command:

```
mode electrostatic !gas
```

Electrostatic Model

Electrostatic analysis in Sentaurus Interconnect is based on the Poisson equation [1]:

$$\nabla \cdot (\epsilon \nabla \phi) = -\rho_f \tag{94}$$

7: Electrostatic Analysis

Electrostatic Model

where:

- ϕ is the electric potential.
- $\epsilon = \epsilon_r \epsilon_0$ is the permittivity, which, in turn, is a product of the relative permittivity ϵ_r and the electric constant ϵ_0 .

The relative isotropic permittivity is a double-precision floating-point number that can be modified with the command:

```
pdbSet <material> Potential Permittivity <n>
```

If the permittivity is anisotropic, it can be set with the command:

```
pdbSet <material> Potential Aniso.Permittivity { X<n> Y<n> Z<n> }
```

where:

- X, Y, Z are the simulation domain x, y, and z respectively.
- <n> is the permittivity in that direction.

If permittivity is set to zero in any direction, the value available for isotropic permittivity is used in that direction. Individual components of the anisotropic permittivity can be set by:

```
pdbSet <material> Potential Aniso.Permittivity X <n>  
pdbSet <material> Potential Aniso.Permittivity Y <n>  
pdbSet <material> Potential Aniso.Permittivity Z <n>
```

Setting any component of the anisotropic permittivity automatically activates the anisotropic model. To switch off the anisotropic permittivity model, all of its components must be set to zero.

It is also possible to define a prefactor for the permittivity using the command:

```
pdbSet <material> Potential Permittivity.Factor <String Expression>
```

where <String Expression> can be any valid Alagator expression.

For example, the following command sets the isotropic permittivity to be 1x, 1.2x, and 3x in the x-, y-, and z-directions, respectively:

```
pdbSet Oxide Potential Permittivity.Factor "diag(1.0,1.2, 3.0)"
```

NOTE For electrostatic analysis to converge, the material type of each nonelectrode region in the structure must have a well-defined relative permittivity.

The source term ρ_f is an optional fixed-charge density term that can be set on a region with the `supply` command, for example:

```
supply region.name=Oxide_1 charge.density=1.0e-3<C/cm3>
```

Boundary Conditions

To ensure that the Poisson equation has a well-defined steady-state solution, the problem must be properly constrained. This is done by applying boundary conditions to a set of electrodes, one boundary condition per electrode.

For all nonelectrode outer boundaries, simple reflective boundary conditions apply, while following E-flux conserving continuous boundary conditions apply at nonelectrode material interfaces:

$$D_{2n} - D_{1n} = \rho_s \quad (95)$$

$$E_{2t} = E_{1t} \quad (96)$$

where:

- D_n is the normal component of the electric displacement
- ρ_s is the surface charge density.
- E_t is the tangential component of the electric field.

Nonelectrode boundary conditions are established automatically by the system and are not user configurable.

Defining Electrodes

To define electrodes, use the `contact` command (see [contact on page 381](#)), for example:

```
contact box xlo=2<um> xhi=3<um> ylo=0<um> yhi=0<um> Copper name=elect1 \
    sidewall
contact box xlo=7<um> xhi=7<um> ylo=0<um> yhi=10<um> Copper name=elect2 bottom
```

Applying Boundary Conditions to Electrodes

To apply boundary conditions for electrostatic analysis to electrodes, use the `supply` command (see [supply on page 647](#)), for example:

```
supply contact.name = elect1 voltage=1.0e-3<V>
supply contact.name = elect2 charge=1e-18<Coulomb>
```

7: Electrostatic Analysis

Capacitance Matrix

The first command specifies a constant voltage-type boundary condition on contact `elect1` and sets it to 1×10^{-3} V. The second command specifies a constant charge-type boundary condition on contact `elect2` and sets it to 1×10^{-18} C.

Capacitance Matrix

Sentaurus Interconnect calculates the capacitance matrix among electrodes. To calculate the capacitance matrix, use the `mode` command (see [mode on page 518](#)), for example:

```
mode capacitance
```

For a detailed explanation, see [Appendix B on page 683](#).

NOTE The capacitance matrix analysis requires a Raphael user license.

Output Data Fields

[Table 10](#) lists the data fields pertinent to electrostatic analysis, and whether the field applies to elements or nodes.

Table 10 Data fields pertinent to electrostatic analysis

Field name	Element/Node	Description	Unit
ElectrostaticPotential	Node	Electric potential	V
ElectricField	Node	Electric field	V/cm

Example: Electrostatic Analysis of Strip Between Two Ground Planes

The following self-contained example demonstrates electrostatic analysis on a simple structure containing a straight copper wire embedded in oxide between two metal ground planes (see [Figure 26 on page 148](#)). The wire and the ground planes are biased at 1.33 V and 0 V, respectively. The example demonstrates how to report the maximum magnitude of the electric field to the log file and how to generate a hotspot map of the electric field in a new data field that is exported to a file together with other simulations results.

```
# Override default relative permittivity for Oxide in database
pdbSet Oxide Potential Permittivity 4.1

# Define electrodes
```



```

contact region=Copper_1 name=electrode1 !replace
contact region=Metal_1 name=groundPlane1 !replace
contact region=Metal_2 name=groundPlane2 !replace

# Set boundary conditions on electrodes
supply contact.name=electrode1 voltage=1.33<V>
supply contact.name=groundPlane1 voltage=0<V>
supply contact.name=groundPlane2 voltage=0<V>

# Define the structure
line clear
line x location=0.0 spacing=0.25 tag=StructTop
line x location=0.5 spacing=0.25 tag=GrPl1Bot
line x location=2.7 spacing=0.1 tag=StripTop
line x location=3.3 spacing=0.1 tag=StripBot
line x location=5.5 spacing=0.25 tag=GrPl2Top
line x location=6.0 spacing=0.25 tag=StructBot

line y location=0.0 spacing=0.5 tag=StructLeft
line y location=4.0 spacing=0.1 tag=StripLeft
line y location=6.0 spacing=0.1 tag=StripRight
line y location=10.0 spacing=0.5 tag=StructRight

line z location=0.0 spacing=2.0 tag=front
line z location=20.0 spacing=2.0 tag=back
region Metal xlo=StructTop xhi=GrPl1Bot ylo=StructLeft yhi=StructRight \
zlo=front zhi=back
region Metal xlo=GrPl2Top xhi=StructBot ylo=StructLeft yhi=StructRight \
zlo=front zhi=back

region Oxide xlo=GrPl1Bot xhi=StripTop ylo=StructLeft yhi=StructRight \
zlo=front zhi=back
region Oxide xlo=StripBot xhi=GrPl2Top ylo=StructLeft yhi=StructRight \
zlo=front zhi=back
region Oxide xlo=StripTop xhi=StripBot ylo=StructLeft yhi=StripLeft \
zlo=front zhi=back
region Oxide xlo=StripTop xhi=StripBot ylo=StripRight yhi=StructRight \
zlo=front zhi=back
region Copper xlo=StripTop xhi=StripBot ylo=StripLeft yhi=StripRight \
zlo=front zhi=back

init !DelayFullD

# Specify electrostatic analysis
mode electrostatic

# Start simulation
solve info=2

# Set up a dielectric breakdown criterion for all oxide regions in
# units of V/cm and store the hotspot areas in a separate field. The suffix
# '_mag' calculates the absolute value of the vector field 'ElectricField'.

```

7: Electrostatic Analysis

References

```
select z="ElectricField_mag > 1.2e4" Oxide name=Max_E_FieldExceeded store  
  
# Save results  
struct tdr=strip !Gas !interfaces !contacts  
  
# Print maximum E-Field value to logfile  
set max_E_field [select name=ElectricField max]  
LogFile "MAX(E-field) = $max_E_field V/cm"
```

Figure 26 shows the electrostatic potential distribution for the example.

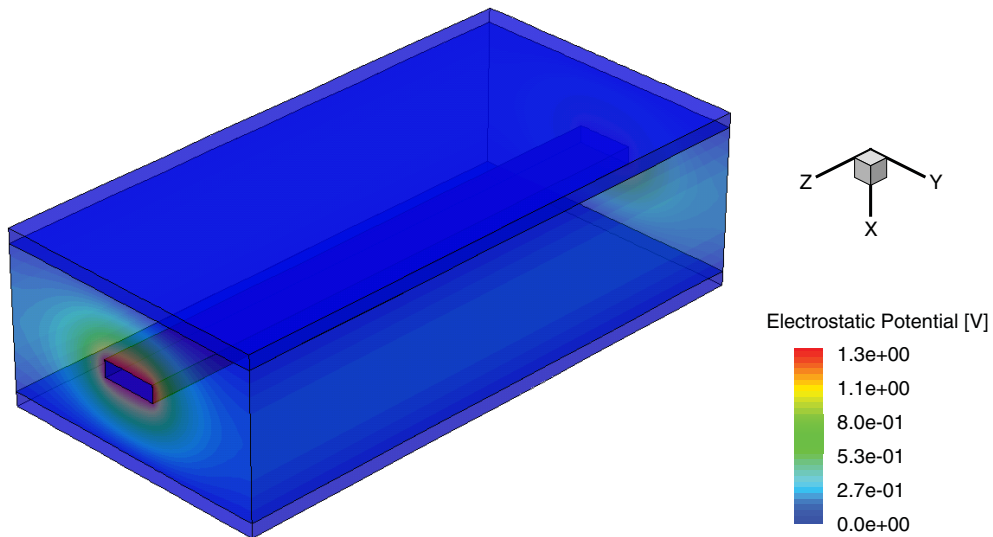


Figure 26 Electrostatic potential for a conducting strip embedded in an oxide region between two ground planes

References

- [1] J. R. Reitz, F. J. Milford, and R. W. Christy, *Foundations of Electromagnetic Theory*, Chapters 3–4, Reading, Massachusetts: Addison-Wesley, 4th ed., 1993.

CHAPTER 8 Mixed-Mode Analysis

This chapter discusses the mixed-mode analysis capability of Sentaurus Interconnect.

Overview

Sentaurus Interconnect is an interconnect reliability analysis simulator with circuit simulator capabilities in mixed mode. For a mixed-mode simulation, the command file must include specifications of the mesh, the contacts, and the model parameters for the interconnect structure (physical device). A circuit netlist must be defined forming a whole system.

To solve the whole system of interconnect structure and circuits, use the `solve` command. Initialization of circuit nodes is automatic and can be switched on or off with the `isolve` parameter of the `solve` command (see [solve on page 629](#)). A command file can have only one interconnect structure.

To create a netlist and connect the circuit elements and the interconnect structure, the `System` section (see [System Section on page 155](#)) or the `circuit` command (see [circuit on page 377](#)) is required.

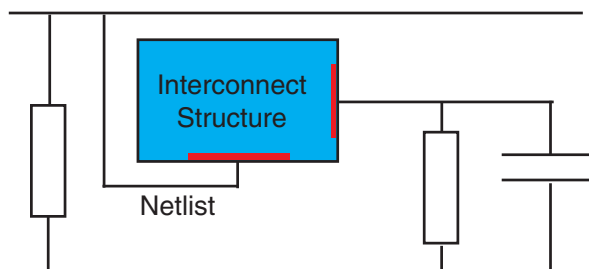


Figure 27 In mixed-mode simulation, an interconnect structure is connected to a circuit netlist

Sentaurus Interconnect also provides a number of compact models for use in mixed mode.

Compact Models

In Sentaurus Interconnect, the compact models comprise three levels:

- Device

This describes the basic properties of a compact model and includes the names of the model, electrodes, and internal variables; and the names and types of the internal states and parameters. The devices are predefined for SPICE models and built-in models. For user-defined models, you must specify the devices.

- Parameter set

Each parameter set is derived from a device. The set defines default values for the parameters of a compact model. Usually, a parameter set defines parameters that are shared among several instances. Most SPICE and built-in models provide a default parameter set, which can be directly referenced in a circuit description. For more complicated models, such as MOSFETs, you can introduce new parameter sets.

- Instances

Instances correspond to the elements in the Sentaurus Interconnect circuit. Each instance is derived from a parameter set. If necessary, an instance can override the values of its parameters.

For SPICE and built-in models, you can define parameter sets and instances. For user-defined models, it is possible (and required) to introduce new devices. This is described in the *Compact Models User Guide*.

The parameter sets and instances in a circuit simulation are specified in external SPICE circuit files (see [SPICE Circuit Files on page 154](#)). These files are recognized by the extension `.scf` and are parsed by Sentaurus Interconnect at the beginning of a simulation.

[Table 11](#) lists the built-in models, and [Table 12 on page 151](#) presents an overview of the SPICE models.

Table 11 Built-in models

Model	Device	Default parameter set
Parameter interface	Param_Interface_Device	Param_Interface

Table 12 SPICE models

Model	Device	Default parameter set
Resistor	Resistor	Resistor_pset
Capacitor	Capacitor	Capacitor_pset
Inductor	Inductor	Inductor_pset
Coupled inductors	mutual	mutual_pset
Voltage-controlled switch	Switch	Switch_pset
Current-controlled switch	CSwitch	CSwitch_pset
Voltage source	Vsource	Vsource_pset
Current source	Isource	Isource_pset
Voltage-controlled current source	VCCS	VCCS_pset
Voltage-controlled voltage source	VCVS	VCVS_pset
Current-controlled current source	CCCS	CCCS_pset
Current-controlled voltage source	CCVS	CCVS_pset
Junction diode	Diode	Diode_pset
Bipolar junction transistor	BJT	BJT_pset
Junction field effect transistor	JFET	JFET_pset
MOSFET	Mos1	Mos1_pset
	Mos2	Mos2_pset
	Mos3	Mos3_pset
	Mos6	Mos6_pset
	BSIM1	BSIM1_pset
	BSIM2	BSIM2_pset
	BSIM3	BSIM3_pset
	BSIM4	BSIM4_pset
B3SOI	B3SOI_pset	
GaAs MESFET	MES	MES_pset
HSPICE Level 1	HMOS_L1	
HSPICE Level 2	HMOS_L2	
HSPICE Level 3	HMOS_L3	
HSPICE Level 28	HMOS_L28	

8: Mixed-Mode Analysis

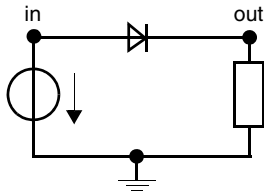
Overview

Table 12 SPICE models

Model	Device	Default parameter set
HSPICE Level 49	HMOS_L49	
HSPICE Level 53	HMOS_L53	
HSPICE Level 54	HMOS_L54	
HSPICE Level 57	HMOS_L57	
HSPICE Level 59	HMOS_L59	
HSPICE Level 61	HMOS_L61	
HSPICE Level 62	HMOS_L62	
HSPICE Level 64	HMOS_L64	
HSPICE Level 68	HMOS_L68	
HSPICE Level 69	HMOS_L69	
HSPICE Level 73	HMOS_L73	
HSPICE Level 76	HMOS_L76	

Example: Compact Models

Consider the following simple rectifier circuit:



The circuit comprises three compact models and can be defined in the file `rectifier.scf` as:

```
PSET D1n4148
  DEVICE Diode
  PARAMETERS
    is = 0.1p // saturation current
    rs = 16 // Ohmic resistance
    cjo = 2p // junction capacitance
    tt = 12n // transit time
    bv = 100 // reverse breakdown voltage
    ibv = 0.1p // current at reverse breakdown voltage
  END PSET

  INSTANCE v
```

```

PSET Vsource_pset
ELECTRODES in 0
PARAMETERS sine = [0 5 1meg 0 0]
END INSTANCE

INSTANCE d1
PSET D1n4148
ELECTRODES in out
PARAMETERS temp = 30
END INSTANCE

INSTANCE r
PSET Resistor_pset
ELECTRODES out 0
PARAMETERS resistance = 1000
END INSTANCE

```

The parameter set D1n4148 defines the parameters shared by all diodes of type 1n4148. Instance parameters are usually different for each diode, for example, their operating temperature.

NOTE A parameter set must be declared before it can be referenced by an instance.

The *Compact Models User Guide* further explains the SPICE parameters in this example. The command file for this simulation can be:

```

Grid1D
pdbSet Circuit SPICE.Model.Path "/spice/lib"
mode current
solve time=40e-9<s>
LogFile [circuit node Potential print]

```

The SPICE.Model.Path is assigned a directory name for SPICE files. The directory is scanned for .scf files (SPICE circuit files).

Check the log file of Sentaurus Interconnect to see which circuit files were found and used in the simulation.

The solve command (see [solve on page 629](#)) denotes the interconnect and circuit equations to be solved. The circuit command (see [circuit on page 377](#)) prints the voltage values at the circuit nodes.

8: Mixed-Mode Analysis

SPICE Circuit Files

The instances in a circuit also can appear directly in the `System` section of the command file, for example:

```
System {  
    Vsource_pset v (in 0) {sine = (0 5 1meg 0 0)}  
    Dln4148 d1 (in out) {temp = 30}  
    Resistor_pset r (out 0) {resistance = 1000}  
}
```

SPICE Circuit Files

Compact models can be specified in an external SPICE circuit file, which is recognized by the extension `.scf`. The declaration of a parameter set can be:

```
PSET pset  
    DEVICE dev  
    PARAMETERS  
    parameter0 = value0  
    parameter1 = value1  
    ...  
END PSET
```

This declaration introduces the parameter set `pset` that is derived from the device `dev`. It assigns default values for the given parameters. The device `dev` should have already declared the parameter names. Furthermore, the values assigned to the parameters must be of the appropriate type.

[Table 13](#) lists the possible parameter types.

Table 13 Parameters in SPICE circuit files

Parameter type	Example	Parameter type	Example
char	c = 'n'	char[]	cc = ['a' 'b' 'c']
int	i = 7	int[]	ii = [1 2 3]
double	d = 3.14	double[]	dd = [1.2 -3.4 5e6]
string	s = "hello world"	string[]	ss = ["hello" "world"]

Similarly, the circuit instances can be declared as:

```
INSTANCE inst
  PSET pset
  ELECTRODES e0 e1 ...
  PARAMETERS
    parameter0 = value0
    parameter1 = value1
    ...
END INSTANCE
```

According to this declaration, the instance `inst` is derived from the parameter set `pset`. Its electrodes are connected to the circuit nodes `e0`, `e1`. This instance also defines or overrides parameter values.

See [Compact Models User Guide, Syntax of Compact Circuit \(.ccf\) Files on page 174](#) for the complete syntax of the input language for SPICE circuit files.

NOTE The tool `spice2sdevice` is available to convert Berkeley SPICE and HSPICE circuit files (extension `.cir`) to Sentaurus Interconnect circuit files (extension `.scf`) (see [Utilities User Guide, Chapter 3 on page 13](#)).

System Section

The `System` section (see [System on page 651](#)) defines the netlist of the physical device and the circuit elements to be solved. The netlist is connected through circuit nodes. By default, a circuit node is electrical. Each electrical node is associated with a voltage variable. Node names are numeric or alphanumeric. The node `0` is predefined as the ground node (`0V`).

Compact models can be defined in SPICE circuit files (see [SPICE Circuit Files on page 154](#)). However, instances of compact models also can appear directly in the `System` section of the command file:

```
parameter-set-name instance-name (node0 node1 ...) {
  <attributes>
}
```

The order of the nodes in the connectivity list corresponds to the electrodes and thermodes in the SPICE device definition (refer to the *Compact Models User Guide*).

The connectivity list is a list of `contact-name=node-name` connections, separated by space. `Contact-name` is the name of the contact from the grid file of the given device, and `node-name` is the name of the circuit netlist node as previously defined in the definition of a circuit element.

The connectivity list of a physical device explicitly establishes the connection between a contact and node. For example, the following defines two circuit resistors:

```
Resistor_pset r1(1 0) {...} # circuit resistor
Resistor_pset r2(1 GC){...} # resistor connected to device and circuit node
```

Both resistors have one of their terminals connected to node 1. Resistor `r1` has its other terminal connected to ground (0), and resistor `r2` has its other terminal connected to one of the following:

- The physical device contact `GC`. The name `GC` of the contact is defined in the grid file of the physical device. For the complete syntax of the input language for SPICE circuit files, see [Compact Models User Guide, Syntax of Compact Circuit \(.ccf\) Files on page 174](#).
- The node `GC`, which in turn is connected to the physical device contact by the `circuit interconnect` command (see [Temperature Dependency on page 157](#)).

To plot voltage and current values at circuit nodes, use the `option` and `WritePlx` commands (see [option on page 526](#) and [WritePlx on page 681](#)).

NOTE A physical device, even if it is not part of the circuit netlist, must exist in order to solve circuit equations.

Circuit Devices

SPICE instances can be declared in SPICE circuit files as discussed in [SPICE Circuit Files on page 154](#). They also can appear directly in the `System` section of the command file, for example:

```
pset inst (e0 e1 ... t0 t1 ...) {
  parameter0 = value0
  parameter1 = value1
  ...
}
```

This declaration in the command file provides the same information as the equivalent declaration in the SPICE circuit file.

Array parameters must be specified with parentheses, not braces, for example:

```
dd = (1.2 -3.4 5e6)
ss = ("hello" "world")
```

Certain SPICE models have internal nodes. For example, a SPICE inductance creates an internal node `branch`, which represents the current through the instance. Therefore, the expression `l.branch` can be used to gain access to the current through the inductor `l`. This is

useful for plotting internal data (refer to the *Compact Models User Guide* for a list of the internal nodes for each model).

In addition, instances of compact models can appear outside the `System` section of the command file using the `circuit` command (see [circuit on page 377](#)):

```
circuit add name=<instance name> model.name = <parameter set name> \  
    electrodes = <nodes> instance.parameters = {<attributes>}
```

The order of the nodes in the connectivity list corresponds to the electrodes and thermodes in the SPICE device definition as in the `System` section (see [System Section on page 155](#)).

Temperature Dependency

You can perform coupled electrothermal analysis in compact models through the definition of thermal nodes in the circuit. Compact model instances are declared as before, however, the nodes associated with these instances can be specified as thermal nodes or thermodes.

As for electrical nodes, boundary conditions can be specified on the thermodes, and they can be connected to contacts. The only difference is that these nodes are solved for temperature, analogous to potential for electrical nodes.

Thermodes can be defined using the `System` command:

```
System {  
    Thermal (t0 t1 ...)  
    ...  
}
```

or the `circuit` command:

```
circuit thermodes = {t0 t1 ...}
```

A thermal instance also can be added as:

```
circuit add name=<instance name> model.name= <parameter set name> \  
    thermodes= <nodes> instance.parameters= {<attributes>}
```

NOTE An instance can be attached to both electrodes and thermodes simultaneously.

8: Mixed-Mode Analysis

SPICE Circuit Models

You can connect electrical and thermal nodes to the contacts defined on the interconnect (TDR) structure using the `circuit` command. For example, the following command connects the contact `MetalContact` to the thermode (or electrode) `t1`:

```
circuit interconnect= {"MetalContact t1"}
```

This allows the connection of an electrode and a thermode to the same contact.

NOTE The `circuit interconnect` command must be specified explicitly to make a connection between the structure or contact and the circuit nodes. The contacts cannot be specified in the `System` command. Therefore, in the absence of the `circuit interconnect` command, a contact will not *see* the node to which it should be attached.

In addition, boundary conditions can be set on the thermodes using the `System` or `circuit` command (see [circuit](#) on page 377 and [System](#) on page 651).

SPICE Circuit Models

Sentaurus Interconnect supports SPICE circuit models for mixed-mode simulations. These models are based on Berkeley SPICE 3 Version 3F5. Several frequently used HSPICE models also are available. For a detailed description of the SPICE models, see the *Compact Models User Guide*.

SPICE Circuit Parameters

The SPICE circuit parameters, model path, model file extension, minimum conductivity, temperature, and nominal temperature can be specified using the `pdb` commands:

```
pdbSet Circuit SPICE.Model.Path <string>
pdbSet Circuit SPICE.Model.Extension <string>
pdbSet Circuit Potential SPICE.Minimum.Conductivity <n>
pdbSet Circuit Temperature SPICE.Temperature <n>
pdbSet Circuit Temperature SPICE.Nominal.Temperature <n>
```

The value of `SPICE.Temperature` denotes the global SPICE circuit temperature. Its default value is 300.0 K. The value of `SPICE.Nominal.Temperature` denotes the nominal SPICE circuit temperature. Its default value is 300.0 K.

The parameter `SPICE.Minimum.Conductivity` refers to the minimum conductance in SPICE. The default value is $10^{-12} \Omega^{-1}$.

User-Defined Circuit Models

Sentaurus Interconnect provides a compact model interface (CMI) for user-defined circuit models. The models are implemented in C++ by users and are linked to Sentaurus Interconnect at runtime. Access to the source code of Sentaurus Interconnect is not required.

To implement a new user-defined model:

1. Provide a corresponding equation for each variable in the compact model. For electrode voltages, compute the current flowing from the device into the electrode. For an internal model variable, use a model equation.
2. Write a formal description of the new compact model. Sentaurus Interconnect reads this compact circuit file before the model is loaded.
3. Implement a set of interface subroutines C++. Sentaurus Interconnect provides a runtime environment.
4. Compile the model code into a shared object file, which is linked at runtime to Sentaurus Interconnect. A `cmi` script executes this compilation.
5. Use the variable `CMI.Model.Path` in the command file to define a search path.
6. Reference user-defined compact models in compact circuit files (with the extension `.ccf`) or directly in the `System` section of the command file.

The CMI model path and the model file extension can be specified using the commands:

```
pdbSet Circuit CMI.Model.Path <string>
pdbSet Circuit CMI.Model.Extension <string>
```

Example: Mixed-Mode Electrical Current Analysis

The following self-contained example demonstrates mixed-mode electrical current analysis on a simple structure containing a copper wire, three resistors, and a sinusoidal voltage source (see [Figure 28 on page 160](#)). The example demonstrates how to include circuit components with the physical device. Finally, the example shows how to generate time-versus-voltage plots at circuit nodes and device contacts.

```
# Define contacts
contact box xlo=0<um> xhi=0<um> ylo=0.0<um> yhi=6.0<um> Copper \
  name=MetalContact sidewall width = 1.0<cm> depth = 3.0<cm>
contact box xlo=8<um> xhi=8<um> ylo=0.0<um> yhi=6.0<um> Copper \
  name=GoldContact sidewall width = 1.0<cm> depth = 3.0<cm>
```

8: Mixed-Mode Analysis

Example: Mixed-Mode Electrical Current Analysis

```
# Define and initialize the structure
line clear
line x location=0 tag=t1 spacing=4
line x location=8 tag=t4 spacing=4
line y location=0 tag=l1 spacing=.5
line y location=5 tag=l2 spacing=.5
line z location=0 tag=front spacing=.2
line z location=3 tag=back spacing=.2
region Copper xlo=t1 xhi=t4 ylo=l1 yhi=l2 zlo=front zhi=back
init !DelayFullD

# Specify current analysis
mode current

# Specify the contact/node names where voltage will be stored and plotted
option voltage = { GoldContact MetalContact 1 }

# Define the netlist of the circuit and device
System {
  Vsource_pset v (GoldContact 0) {sine = [0.2 0.4 0.25 0.5 0.01]}
  Resistor_pset r1 (MetalContact 1) {resistance = 1000}
  Resistor_pset r2 (1 0) {resistance = 1000}
  Resistor_pset r3 (1 0) {resistance = 1000}
}

# Run the simulation
solve time=5<s> maxstep=0.05<s> init=1e-2<s>

# Print the voltage values @ circuit nodes
circuit node Potential print

# Write "Out.plx" file to plot the voltages at circuit
# nodes GoldContact, MetalContact and 1
WritePlx Out voltage

# Save results
struct tdr=resistor
```

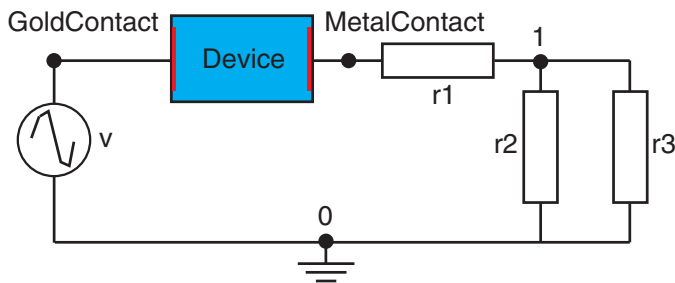


Figure 28 Mixed-mode electrical current analysis circuit

Figure 29 shows the time-versus-voltage plots for GoldContact, MetalContact, and node 1.

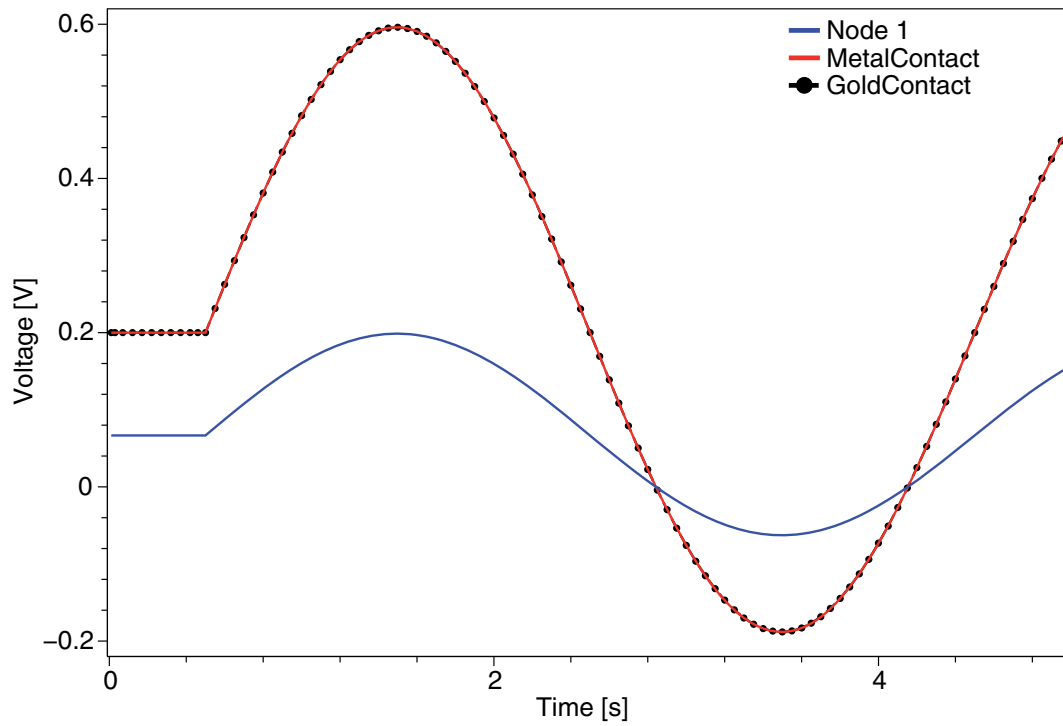


Figure 29 Time-versus-voltage plots for GoldContact, MetalContact, and node 1

8: Mixed-Mode Analysis

Example: Mixed-Mode Electrical Current Analysis

This chapter describes the complementary models that can be activated in Sentaurus Interconnect.

Grain growth in polycrystalline metals means elimination of the grain boundary, which can lead to the evolution of stress and a change in the electrical conductivity. Stress also can affect the grain growth. The grain growth model discussed here covers stress-dependent grain growth.

Grain Growth Model

Grain growth during thermal processing can be simulated by setting:

```
pdbSet <material> Grain.Growth.Model Simple ;# default= None
```

The grain growth model is given by [1]:

$$\frac{\partial L_g}{\partial t} = \text{GrainGrowthFactor GISFactor} \cdot \frac{D_g}{L_g} \cdot \frac{E_{gb}}{kT} \quad (97)$$

where:

- L_g is the grain size (GSize).
- D_g is the effective vacancy diffusivity given by:

```
pdbSet <material> Vacancy D0 <n> ;# cm^2sec^-1
```

- GrainGrowthFactor is a multiplier given by:

```
pdbSet <material> GrainGrowthFactor <string expression> ;# unitless
```

- GISFactor is the grain size dependency on stress:

$$\text{GISFactor} = \left(1 - \frac{2(\text{GB.Width})^2}{3 E_{gb}} \left(\frac{E}{1-\nu} \right) \left(\frac{1}{L_{g0}} - \frac{1}{L_g} \right) \right) \quad (98)$$

where E is the Young's modulus, ν is the Poisson ratio, and L_{g0} is the initial grain size.

GB.Width is defined per material and can be set by:

```
pdbSet <material> GB.Width <n>
```

9: Complementary Models

Grain Growth Model

GISFactor is invoked by setting:

```
pdbSet Mechanics GIS 1
```

If it is not set, GISFactor becomes 1.

NOTE The value of L_{g0} is important to see the stress effects. The smaller the value of L_{g0} , the bigger the stress suppression on growth. GrainSize and GrainSizeFactor are used to set L_{g0} .

- E_{gb} is the surface energy per atom associated with the grain boundary and is given by [1][2]:

$$\begin{aligned} E_{gb} &= \frac{E_{gb0}}{1 + \text{GBSurface.Energy.Density.h} \cdot f_n} + E_{gb1} \cdot \frac{L_g}{t_m} \\ E_{gb0} &= \frac{\text{GBSurface.Energy.Density}}{\text{GBMaxDensity}} \\ E_{gb1} &= \text{GBSurface.Energy.Density.1} \cdot E_{gb0} \end{aligned} \tag{99}$$
$$f_n = \begin{cases} \frac{L_g}{2 \cdot t_m - L_g} & \text{if } L_g < t_m \\ \frac{L_g}{t_m} & \text{if } L_g \geq t_m \end{cases}$$

where:

- t_m is the thickness of the deposited layer.
- GBMaxDensity, GBSurface.Energy.Density, GBSurface.Energy.Density.h, and GBSurface.Energy.Density.1 are material properties:

```
pdbSet <material> Vacancy GBMaxDensity <n>           ;# cm^-2
pdbSet <material> GBSurface.Energy.Density <n>       ;# Jcm^-2
pdbSet <material> GBSurface.Energy.Density.h <n>     ;# unitless
pdbSet <material> GBSurface.Energy.Density.1 <n>     ;# unitless
```

Grain growth induces new strains in affected regions. See [Grain Growth on page 84](#) for details about the grain growth–induced stress model.

Initializing the Grain Size

The grain size is initialized when a polycrystalline material is first deposited. The initial grain size is set as:

$$L_g = \begin{cases} \max(\text{Frac.TA} \cdot t_m, \text{GrainSize}) & \text{if } T \leq \text{GrainSizeTempC} \\ \text{GrainSize} + 2 \cdot \text{GrainSizeFactor} \cdot z & \text{if } T > \text{GrainSizeTempC} \end{cases} \quad (100)$$

where:

- t_m is the thickness of the deposited layer.
- z is the distance from the depositing surface.
- `Frac.TA`, `GrainSize`, `GrainSizeFactor`, and `GrainSizeTempC` are material parameters:

```

pdbSet <material> Frac.TA <n>           ;# unitless
pdbSet <material> GrainSize <n>         ;# cm
pdbSet <material> GrainSizeFactor <n>   ;# unitless
pdbSet <material> GrainSizeTempC <n>    ;# degree Celsius
  
```

Depending on the parameter settings, the initial grain-size field can be either uniform or nonuniform. When it is nonuniform, its value increases linearly from the depositing surface. The farther from the depositing surface, the larger the grain size.

Output Data Fields

Table 14 lists the data fields pertinent to complementary models, and whether the field applies to elements or nodes.

Table 14 Data fields pertinent to complementary models

Field name	Element/Node	Description	Unit
GSize	Node	Grain size	cm
GSizeAtto	Node	Solution name of grain size	am

References

- [1] L. Mei and R. W. Dutton, "A Process Simulation Model for Multilayer Structures Involving Polycrystalline Silicon," *IEEE Transactions on Electron Devices*, vol. ED-29, no. 11, pp. 1726–1734, 1982.
- [2] L. Mei *et al.*, "Grain-Growth Mechanisms in Polysilicon," *Journal of the Electrochemical Society*, vol. 129, no. 8, pp. 1791–1795, 1982.

CHAPTER 10 Advanced Calibration

This chapter describes the use of Advanced Calibration in Sentaurus Interconnect.

Overview

Synopsys' Consulting and Engineering is working continually on improving the simulation models and optimizing the model parameters for the latest technology. This effort is based on long-standing experience of model calibration for customers and a comprehensive, growing database of state-of-the-art process flows.

The Advanced Calibration set of models and parameters is located in a single file. For the current version of Sentaurus Interconnect, it has the file name `AdvCal_2017.09.sis` and is located in the directory `$(STROOT)/tcad/$(STRELEASE)/lib/sinterconnect/TclLib/AdvCal`.

Using Advanced Calibration

To use Advanced Calibration in Sentaurus Interconnect, at the beginning of the command file, insert the line:

```
AdvancedCalibration
```

or:

```
AdvancedCalibration 2017.09
```

This command sources the file `$(STROOT)/tcad/$(STRELEASE)/lib/sinterconnect/TclLib/AdvCal/AdvCal_2017.09.sis`.

You can use Advanced Calibration parameters and models from previous releases, for example:

```
AdvancedCalibration 2016.12
```

This command sources the file `$(STROOT)/tcad/$(STRELEASE)/lib/sinterconnect/TclLib/AdvCal/AdvCal_2016.12.sis`.

Additional Calibration by Users

Advanced Calibration is based on the assumption that all parameters that are not changed in the parameter files are the Sentaurus Interconnect default parameters. To use the Advanced Calibration file `AdvCal_2017.09.sis`, it must be sourced before the real process description.

The best way to perform this is to put all additional calibration in a user calibration file, for example, `user_calibration.sis`. This file includes all project-specific changes of physical parameters or callback procedures with respect to Advanced Calibration.

In the process simulation file, at the beginning of the process simulation, insert the lines:

```
AdvancedCalibration 2017.09
source ./user_calibration.sis
```

This method has distinct advantages:

- There is a clear separation between the process flow, which is contained in the Sentaurus Interconnect command file, and the selection of physical models and parameters. During calibration of Sentaurus Interconnect for a specific technology, you can first set up the process flow in the command file of Sentaurus Interconnect and then improve the accuracy of the simulation by making changes only in its parameter file. Conversely, if you want to apply the same models and parameters to a different process, it is only necessary to change the file containing the process flow.
- The Advanced Calibration file is used as a starting point. The user calibration file is usually short and clear. You can see all parameter changes with respect to the original Advanced Calibration at a glance.

NOTE For detailed documentation of the contents and physical models included in Advanced Calibration as well as a discussion of its accuracy and limitations, refer to the relevant chapters in the *Advanced Calibration for Process Simulation User Guide*.

This chapter describes the mesh algorithms and meshing parameters available in Sentaurus Interconnect.

Overview

Sentaurus Interconnect automatically generates meshes as they are needed. The behavior of the automatic-meshing scheme differs depending on the dimension of the simulation because of the time required to generate meshes. In one dimension and two dimensions, meshes are generated after every geometry operation such as etching and deposition. In three dimensions, meshes are only generated immediately before steps that require a bulk mesh, such as a `solve` command and structure saving. This scheme can reduce the time spent when there are multiple geometry-changing steps without a `solve` command (or any other step requiring a mesh) in between.

Sentaurus Interconnect uses Sentaurus Mesh as its meshing engine. Since Sentaurus Mesh is a suite of tools, when discussing its different meshing algorithms, for simplification, *Sentaurus Mesh* will be used in this chapter.

The mesh generation process starts with a bisection algorithm, which places mesh points as instructed by the user. Afterwards, the mesh elements are created using a modified Delaunay-meshing algorithm. Refer to the [Sentaurus™ Mesh User Guide](#) for details.

The meshes generated within Sentaurus Interconnect can be refined adaptively, statically, or as a combination of adaptive and static refinements. The refinement can be specified using one of the major types of refinement box:

- Field based (adaptive meshing)
- Mask based
- Uniform (standard)
- Interface axis-aligned
- Interface offsetting (offset normal to the interface)

All these refinement types are user controllable. In addition, Sentaurus Mesh enforces mesh smoothing to limit the changes in element size from one element to the next. This smoothing is important for mechanics accuracy and convergence behavior (see [Mesh Refinement on page 170](#)).

11: Mesh Generation

Overview

One important algorithm affecting refinement behavior is the `UseLines` algorithm. This algorithm inserts lines created using the `line` command into the internal bisection algorithm before any other lines are introduced. Further mesh refinement proceeds by bisecting the boxes created by the `UseLines` lines. This has the effect of isolating static regions of a structure from regions where the boundaries are moving due to geometric operations. Geometry movement naturally causes perturbations to the mesh lines. The `UseLines` lines compartmentalize this mesh movement to minimize solution degradation from interpolation. See [UseLines: Keeping User-Defined Mesh Lines on page 201](#).

NOTE Because the bisection algorithm in Sentaurus Interconnect differs from the one used to create mesh refinement in the standalone Sentaurus Mesh tool, it is not possible to create meshes identical to those created with Sentaurus Mesh. However, element quality, stability, and the Delaunay properties should be qualitatively the same.

Mesh Refinement

By default, no mesh refinement is applied. You must add refinement to achieve accurate results.

Mesh refinement is a two-step process:

- First, you define the refinement box.
- Second, the mesh is refined when the next remesh occurs either with an explicit `grid remesh` call or during standard geometry modifications such as `etch`, `deposit`, `clip`, or native layer formation.

The refinement boxes remain valid unless the list of refinement boxes is cleared with the `refinebox clear` command.

All refinement boxes have refinement criteria that add mesh and constraints that can be used to spatially limit where the mesh refinement occurs. One type of mesh refinement criteria is available for each type of refinement box, and the criteria essentially define the box type. The mesh refinement criteria and, therefore, the refinement box type can be either static or adaptive. All types of refinement box can be mixed in a command file as required.

NOTE Only one type of mesh refinement must be defined in one `refinebox` command.

The refinement box spatial constraints are specified along with the mesh refinement criteria in the `refinebox` command and can be combined in one command.

The spatial constraints available include:

- A material constraint using the `materials` argument that takes a list of materials.
- Region constraints using the `regions` argument that takes a list of region names.
- The `min` and `max` arguments that limit the size of the refinement box (which by default applies to all of the space).
- A mask constraint using the `mask`, `extrusion.min`, and `extrusion.max` arguments that limit the size of the refinement box.

Refinement information also can be extracted and written to a file readable by Sentaurus Mesh using the `mshcmd` flag in conjunction with the `tdr` or `tdr.bnd` argument of the `struct` command (see [struct on page 643](#)).

Viewing Mesh Refinement

To aid in setting mesh refinement, you can store the current minimum edge length in each direction as a field using the command:

```
pdbSet Grid Set.Min.Edge 1
```

When specified, Sentaurus Interconnect computes the smallest edge length in each direction and saves it in three fields:

- `MinXEdgeLength`
- `MinYEdgeLength` (for 2D or 3D structures)
- `MinZEdgeLength` (for 3D structures)

In addition, it prints the average edge length to the screen.

Static Mesh Refinement

This section discusses static (not adaptive) mesh refinement.

Standard Refinement Boxes

The standard refinement box allows you to specify a smoothly varying mesh density inside the refinement box at three locations in the *x*-, *y*- and *z*-directions using the `xrefine`, `yrefine`, and `zrefine` parameter lists, respectively.

11: Mesh Generation

Static Mesh Refinement

Each of the `xrefine`, `yrefine`, and `zrefine` parameters lists can contain from one to three values. If all three `xrefine`, `yrefine`, and `zrefine` values are specified, the mesh density varies quadratically in that direction. If two are specified, the variation is linear from top to bottom. If only one value is specified, a constant mesh density is assumed.

This example specifies two refinement boxes and performs a remesh:

```
refinebox min= {-0.25 0.4 0.0} max= {0.4 0.6 1.0} xrefine= {0.1 0.06 0.1} \  
  yrefine= {0.1 0.01 0.1} zrefine = {0.01} oxide  
refinebox min= {0.6 0.6} max= {0.8 0.8} xrefine= 0.1 silicon  
grid remesh
```

The first refinement box only applies to oxides within the cube delimited by $-0.25 \leq x \leq 0.4$, $0.4 \leq y \leq 0.6$, $0.0 \leq z \leq 1.0$. It specifies a quadratically varying mesh density for x and y , and a constant mesh density for z .

NOTE Calculating the linear or quadratic variation of the mesh density when two or three x -, y -, or z -direction values are given requires the specification of `min` and `max`. If `min` and `max` are not specified and at least one region is specified, the minimum and maximum values of the bounding box for that region serve as `min` and `max` for the calculation. If more than one region is specified, only the bounding box of the first region is used for the calculation, although all regions are used as constraints to the refinement.

Interface Axis-Aligned Refinement Boxes

Refinement near interfaces can be specified globally or constrained spatially inside a refinement box using the `refinebox` command. So it is possible to have a large global default minimum interface mesh-spacing, for example, and a smaller localized value inside a box. The parameters affecting interface refinement are demonstrated in the following examples:

- Set the global mesh criteria near interfaces. This is the maximum size the first normal edge can be, and it is possible for the edge to be `0.5 min.normal.size`:

```
pdbSet Grid SnMesh min.normal.size <n>
```

- Set the global growth rate of the edge size away from interfaces:

```
pdbSet Grid SnMesh normal.growth.ratio.2d <n>  
pdbSet Grid SnMesh normal.growth.ratio.3d <n>
```

- To constrain the interface mesh specification to a particular set of materials or material interfaces, set `min.normal.size`, or `normal.growth.ratio`, or both locally within a refinement box:

```
refinebox min.normal.size=<n> normal.growth.ratio=<n>  
  [interface.materials= <list> | interface.mat.pairs= <list>]
```

Interface Offsetting Refinement Boxes

In addition, the Sentaurus Mesh offsetting algorithm can be used to create offsetting layers that are conformal to the interface rather than aligned to the coordinate axes by specifying the `offsetting` keyword, which also permits regionwise interface specification in addition to the materialwise possibility:

```
refinebox offsetting min.normal.size=<n> normal.growth.ratio=<n>
  [interface.materials= <list> | interface.mat.pairs= <list>]
  [interface.regions= <list> | interface.region.pairs= <list>]
```

For Sentaurus Mesh offsetting, an additional argument `offsetting.maxlevel=<i>` defines the number of layers to be generated at the interface. It can be defined globally through the PDB parameter `Grid SnMesh offsetting.maxlevel`, or on a materialwise or regionwise basis using the `refinebox` command as shown in the following possibilities:

```
pdbSet Grid SnMesh offsetting.maxlevel <i>
refinebox offsetting.maxlevel=<i> interface.materials= <list>
refinebox offsetting.maxlevel=<i> interface.regions= <list>
```

Offset-meshing parameters defined at interfaces using `interface.mat.pairs` or `interface.region.pairs` are interpreted in a symmetric way by default. This means that, given the specification of a material or region pair x_1/x_2 , the parameters are defined for both x_1 at the x_2 interface and x_2 at the x_1 interface. If `!double.side` is given, Sentaurus Mesh interprets x_1/x_2 in a nonsymmetric way, that is, only for x_1 at the x_2 interface.

Refinement Inside a Mask

Mask-based refinements are similar to standard refinements (see [Standard Refinement Boxes on page 171](#)), except that they have an additional constraint that is defined by a volume specified by a previously existing mask. This constraint is applied in addition to the normal box constraint defined by the `min` and `max` parameters. Mask-based refinements are a way to have layout-driven refinements.

For example, if you specify `min` and `max`, the refinement area will be the intersection of the specified rectangle and the mask. If you specify a material name, the final refinement will be the intersection of the regions with such a material and the mask.

These constraints are specified using the `refinebox` command with the following options:

- A mask name (`mask`).
- Minimum and maximum coordinates in the x-direction where the refinement will be applied (`extrusion.min` and `extrusion.max`).

11: Mesh Generation

Static Mesh Refinement

- An optional parameter to see whether the refinement should extend some distance away from the mask (`extend`).

Negative masks are also allowed. Mask boundaries are never interpreted as being infinite in any direction, even if they extend far from the simulation boundary. Consequently, *shrinking* a refinement by specifying a negative extension parameter might leave a region uncovered, even if the mask originally extended past the boundary. For example, if a mask from (-0.010 to 1) covers a domain from (0 to 2), specifying `extend=-0.02` will produce a refinement extending from (0.010 to 0.98), thereby leaving the region from 0 to 0.010 unrefined.

Example

First, create a mask, and then a refinement box can be issued:

```
polygon name= pol segments= { -0.5 -0.5 -.25 -.5 -.25 -.05 .25 -.05 .25 -.5 \
    .5 -.5 .5 0 -.5 0 }
mask name= "Mask" polygons= {pol}

# Now that there is a mask, it can be used to produce a refinement.
refinebox name= "refi_mask" mask= "Mask" xrefine= { .075 .075 .075 } \
    yrefine= { .075 .075 .075 } extrusion.min= 0 extrusion.max= 0.05 \
    extend= -0.1
```

Refinement Near Mask Edges or Mark Corners

Refinement also can be constrained to be near mask edges or mask corners. The following arguments are available in the `refinebox` command:

- For mask edge-based refinement:
 - `mask.edge.refine.extent`
 - `mask.edge.mns`
 - `mask.edge.ngr`
- For mask corner-based refinement:
 - `mask.corner.refine.extent`
 - `mask.corner.mns`
 - `mask.corner.ngr`

The `mask.edge.refine.extent` argument must be specified to switch on mask edge-based refinement and to set the lateral extent of the refinement from the mask edge. Vertically, the mask edge-based refinement can be controlled with the x-coordinate of the `min` and `max` arguments. The minimum mesh spacing near the mask edge is set with `mask.edge.mns` (the default is taken from the PDB parameter `Grid SnMesh min.normal.size`), and the growth

of the edge length away from the mask edge is specified with `mask.edge.ngr` (default is 1.0, meaning the constant edges of lengths `mask.edge.mns` in the normal direction).

Equivalently, the `mask.corner.refine.extent` argument must be specified to switch on mask corner-based refinement and to set the lateral extent of the refinement from the mask corner. Vertically, the mask corner-based refinement can be controlled with the x-coordinate of the `min` and `max` arguments. The minimum mesh spacing near the mask corner is set with `mask.corner.mns` (the default is taken from the PDB parameter `Grid SnMesh min.normal.size`), and the growth of the edge length away from the mask corner is specified with `mask.corner.ngr` (default is 1.0, meaning the constant edges of lengths `mask.corner.mns` in the normal direction).

NOTE Similar to the PDB parameter `Grid SnMesh min.normal.size`, actual edge lengths may be up to two times smaller than `mask.edge.mns` or `mask.corner.mns` at the mask edge or mask corner, respectively, because of the binary-tree refinement algorithm.

NOTE The `refinebox` command can specify either mask edge-based refinement, or mask corner-based refinement, or both at the same time.

An example of using mask edge-based refinement is:

```
polygon name= p1 segments= {1.0 1.0 1.0 5.0 3.0 5.0 3.0 2.5 2.0 2.5 2.0 1.0}
mask name= m1 polygons= p1

refinebox clear

# Prevent mesh propagation by defining regular coarse mesh
refinebox yrefine= 0.5 zrefine= 0.5

# Add edge-based refinement
refinebox mask= m1 mask.edge.mns= 0.08 mask.edge.refine.extent= 0.25

grid remesh
```

The resist layer was created later using the command:

```
photo mask= m1 thickness= 0.05
```

Figure 30 shows the result.

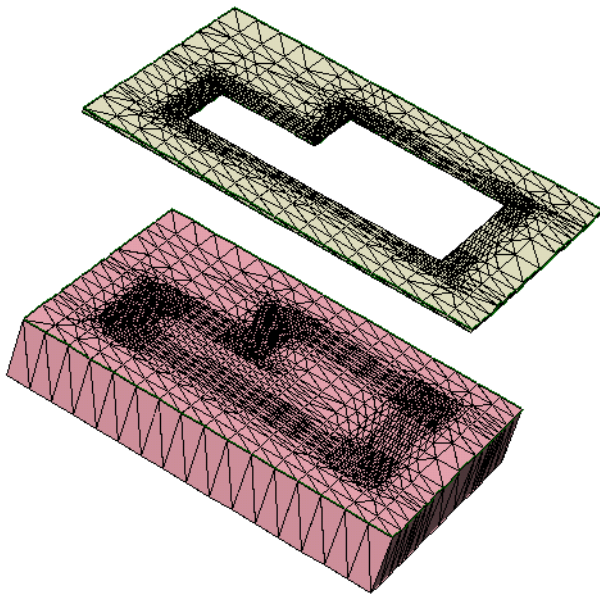


Figure 30 Mask edge-based refinement shown on the mask and in silicon

Uniform Mesh Scaling

Often in simulations, uniform mesh refinement is required to study mesh effects or convergence. After defining all of the other mesh refinement criteria, the mesh can be further split uniformly by a specified factor in each direction. This is achieved by a final split of the binary tree used in refinement.

To scale a mesh by a specified factor, you can use the `binarytree.split.factor.x`, `binarytree.split.factor.y`, and `binarytree.split.factor.z` parameters, for example:

```
pdbSet Grid SnMesh binarytree.split.factor.x 4
pdbSet Grid SnMesh binarytree.split.factor.y 2
pdbSet Grid SnMesh binarytree.split.factor.z 1

grid remesh
```

In this example, the final mesh is refined by a factor of 4 in the x-direction and a factor of 2 in the y-direction, resulting in approximately 8 times more elements.

NOTE The split factors must be specified in powers of 2.

In addition, you can apply the split factors over a specified box, which is defined in two dimensions by:

```
pdbSet Grid SnMesh binarytree.split.box {xmin 0.0 ymin 0.0 xmax 0.0 ymax 0.0}
```

and, in three dimensions by:

```
pdbSet Grid SnMesh binarytree.split.box {xmin 0.0 ymin 0.0 zmin 0.0 xmax 0.0 \  
ymax 0.0 zmax 0.0}
```

Adaptive Mesh Refinement

Tailoring a mesh to a specific problem with static refinement boxes can be tedious and time-consuming. In addition, for some applications, solution profiles evolve so much during the process that the areas where a finer mesh was needed at the beginning are very different from the areas where a finer mesh is needed at the end.

To accurately capture the entire evolution with a static mesh, it is necessary to put a fine mesh over large areas of the structure leading to long simulation times and large memory use. Adaptive meshing in Sentaurus Interconnect addresses these issues.

For details, see [Tips for Adaptive Meshing on page 186](#).

When adaptive meshing is switched on, field-based refinement is performed during every remesh step and for any dimension. This happens for all etching, deposition, native layer, regrid, and transform operations. In addition, during time-stepping at a specified step interval, a check of the current mesh is made to determine whether a remesh is required; then the remesh is performed if necessary. For details, see [Adaptive Meshing During Analysis on page 184](#).

Adaptive Refinement Criteria

Numerous refinement functions are available to deal with differing fields and situations. All functions involve some comparison between values on neighboring nodes and possible values between neighboring nodes. In some cases, the same refinement function is available in Sentaurus Mesh, and similar results to Sentaurus Mesh refinement will be obtained. The following refinement criteria are available:

- Relative difference (default)
- Absolute difference
- Logarithmic difference
- Inverse hyperbolic sine (asinh) difference
- Gradient

11: Mesh Generation

Adaptive Mesh Refinement

- Local dose error
- Interval refinement

These refinements can be applied globally (default) or they can be limited as follows:

- Boxwise
- Materialwise
- Regionwise

Detailed descriptions of the refinement types, their respective control parameters, and instructions for applying refinement constraints are given in subsequent sections. The default adaptive meshing parameters have been set to apply only relative difference criteria to the entire structure, and they typically produce a fairly coarse mesh. It is necessary to set one criterion or more to produce a mesh sufficiently fine to reach a required accuracy.

Adaptive meshing is switched off by default. To switch on adaptive meshing, use:

```
pdbSet Grid Adaptive 1
pdbSet Grid SnMesh UseLines 1 ;# Recommended with adaptive meshing
```

Relative Difference Criteria

The relative difference between two neighboring nodes is computed as follows:

$$2 \frac{|C_1 - C_2|}{(C_1 + C_2 + \alpha)} \quad (101)$$

where C_i is the field value on node i , and α is the field-specific refinement parameter set with:

```
pdbSet Grid <Field> Refine.Abs.Error <n>
```

or `def.abs.error` and `abs.error`, which are parameters of the `refinebox` command.

If the value of the expression in [Eq. 101](#) is greater than the maximum relative difference, the edge between node 1 and node 2 is split. To set the maximum relative difference, use:

```
pdbSet Grid <Field> Refine.Rel.Error <n>
```

or `def.rel.error` and `rel.error`, which are parameters of the `refinebox` command.

The quantity `<Field>` is the name of the field, and `<n>` is a unitless number for `Refine.Rel.Error` and `Refine.Abs.Error`; the units are the same as those of the field. The default values for `Refine.Abs.Error` and `Refine.Rel.Error` are set from `<Field> = AdaptiveField`.

The density of the mesh is sensitive to `Refine.Rel.Error` because it represents the target relative change of the field across an edge. For many standard situations, a number of the order of 1.25 gives a coarse mesh, and a number of approximately 0.5 often gives a fine mesh. The parameter α sets a smooth cutoff such that values of the field below α result in no refinement.

NOTE The relative difference criteria must only be used with fields that are always positive.

Absolute Difference Criteria

The absolute difference between two neighboring nodes is computed simply:

$$|C_1 - C_2| \quad (102)$$

where C_i is the field value on node i . If the value of the expression in [Eq. 102](#) is greater than the maximum absolute difference, the edge between nodes 1 and 2 is split. The maximum allowable absolute difference can be set with:

```
pdbSet Grid <Field> Refine.Max.Difference <n>
```

Logarithmic Difference Criteria

The logarithmic (base 10) difference between two neighboring nodes is computed as follows:

$$|\log(C_1 + \alpha) - \log(C_2 + \alpha)| \quad (103)$$

where C_i is the field value on node i , and α is the low value cutoff that can be set with:

```
pdbSet Grid <Field> Refine.Abs.Error <n>
```

or `def.abs.error` and `abs.error`, which are parameters of the `refinebox` command.

If the value of the expression in [Eq. 103](#) is greater than the maximum logarithmic difference, the edge between nodes 1 and 2 is split. To set the maximum logarithmic difference, use:

```
pdbSet Grid <Field> Refine.Max.LogDiff <n>
```

or `def.max.logdiff` and `max.logdiff`, which are parameters of the `refinebox` command.

NOTE The logarithmic difference criteria must only be used with fields that are always positive. Use the inverse hyperbolic sine (`asinh`) difference criteria for fields that can have negative values such as stresses.

Inverse Hyperbolic Sine (asinh) Difference Criteria

The asinh difference between two neighboring nodes is computed as follows:

$$|\operatorname{asinh}(C_1) - \operatorname{asinh}(C_2)| \quad (104)$$

where C_i is the field value on node i . If the value of the expression in [Eq. 104](#) is greater than the maximum asinh difference, the edge between nodes 1 and 2 is split. To set the maximum asinh difference, use:

```
pdbSet Grid <Field> Refine.Max.AsinhDiff <n>
```

or `def.max.asinhdiff` and `max.asinhdiff`, which are parameters of the `refinebox` command.

Gradient Criteria

The gradient between two neighboring nodes is computed as follows:

$$\frac{|C_1 - C_2|}{l_{12}} \quad (105)$$

where C_i is the field value on node i , and l_{ij} is the length of the edge between nodes i and j . If the value of the expression in [Eq. 105](#) is greater than the maximum gradient, the edge between the two nodes is split. To set the maximum gradient, use:

```
pdbSet Grid <Field> Refine.Max.Gradient <n>
```

or `def.max.gradient` and `max.gradient`, which are parameters of the `refinebox` command.

Local Dose Error Criteria

If an edge between two neighboring nodes is not split, the local dose error is computed as follows:

$$|0.5C_{12} - 0.25C_1 - 0.25C_2|l_{12}^s \quad (106)$$

where:

- C_i is the field value on node i .
- C_{ij} is the concentration at the midpoint between nodes i and j .

- l_{ij} is the length of the edge between nodes i and j .
- s_{ij} is the box size perpendicular to the edge between nodes i and j (see Figure 31).

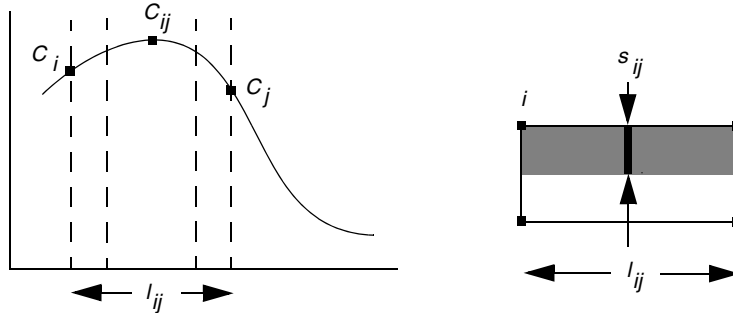


Figure 31 (Left) One-dimensional and (right) 2D representation of dose loss criteria

The function in Figure 31 (left) is taken from the previous mesh. The box with four points in Figure 31 (right) represents one cell of the mesh refinement tree. The shaded area is the part of the 2D field under consideration. The dose in the shaded area is computed in two ways:

- As is
- If the edge between i and j is split

If the difference between these two ways is greater than `max.dose.error`, the edge is split.

The box size is 1.0 (unitless) in 1D; it is the box width (in cm) in 2D; and it is the box area (cm^{-2}) perpendicular to the edge $i-j$ in 3D. If the value of the expression in Eq. 106 is greater than the normalized maximum local dose error, the edge between the two nodes is split. The local dose error can be set with:

```
pdbSet Grid <Field> Refine.Max.DoseError <n>
```

where `<n>` has units of cm^{-2} , or `def.max.dose.error` and `max.dose.error`, which are parameters of the `refinebox` command.

The local dose error is first multiplied by the simulation size before comparing it to the expression in Eq. 106. The simulation size is 1.0 (unitless) in 1D, the simulation width (in cm) in 2D, and the simulation lateral area in 3D (in cm^{-2}).

To estimate the total dose loss, you must estimate how many nodes carry a significant concentration of the field in question and then multiply that number by the local dose error to obtain approximately the maximum total dose error expected. (In practice, the dose error is often considerably less than this.) This quantity is relatively easy to understand and is less sensitive than some other parameters to process conditions.

Interval Refinement Criteria

Interval refinement provides a way to refine the mesh such that field values within a certain interval are well resolved. Interval refinement produces mesh edges of a specified length wherever the field values are within a specified interval. Four parameters are required to define an interval refinement:

- A minimum and maximum value
- C_{\min} and C_{\max}
- A target length, lt
- A target length scaling, s

To preserve the anisotropy of the mesh, interval refinement examines each edge of a refinement cell and calculates an effective edge length l_{eff} defined by:

$$l_{\text{eff}} = \text{abs}((r_1 - r_2) \cdot \nabla C) \quad (107)$$

where r_1 and r_2 are the endpoints of the edge, and ∇C is the average gradient of the field in the refinement cell. Edges that are nearly parallel to the contours of the field have effective edge lengths near zero. Edges that are nearly perpendicular to the contours have effective edge lengths near their actual edge length. Since edges are split only when they are longer than a given target length, edges that are parallel to the field contours are allowed to be longer than those that are perpendicular.

Interval refinement will split any edge whose effective edge length exceeds the effective target length. The effective target length is calculated differently depending on whether the field values on the edge overlap the interval specified by C_{\min} (`refinebox min.value`) and C_{\max} (`refinebox max.value`).

Let C_1 and C_2 be the values of the field on the endpoints of the edge. If the relation $C_{\max} > C > C_{\min}$ is satisfied for any value of C between C_1 and C_2 , the edge overlaps the interval.

For edges that overlap the interval, the effective target length is exactly the target length that you specify (`refinebox target.length`), that is:

$$l_{\text{eff}} = lt \quad (108)$$

For edges that do not overlap:

$$l_{\text{eff}} = lt(1 + \log C_a - \log C_b)^2 s \quad (109)$$

where C_a is either C_{\min} or C_{\max} , C_b is either C_1 or C_2 , and the values of C_a and C_b are chosen to minimize the difference.

The formula for l_{eff} outside the interval produces a graded mesh with an edge length that falls off parabolically with distance from the interval. Default values for the parameters of the interval refinements are defined in the Parameter Database (PDB).

Summary of Refinement Parameters

Table 15 lists `refinebox` parameters in the left column that can be used to specify boxwise refinement. The right column lists the corresponding PDB parameters that can be used to specify refinement criteria globally.

Table 15 Summary of refinement parameters

refinebox parameter	Corresponding entry in parameter database
<code>def.rel.error, rel.error</code>	Grid <Field> Refine.Rel.Error
<code>def.abs.error, abs.error</code>	Grid <Field> Refine.Abs.Error
<code>def.max.difference, max.difference</code>	Grid <Field> Refine.Max.Difference
<code>def.max.logdiff, max.logdiff</code>	Grid <Field> Refine.Max.LogDiff
<code>def.max.asinhdiff, max.asinhdiff</code>	Grid <Field> Refine.Max.AsinhDiff
<code>def.max.gradient, max.gradient</code>	Grid <Field> Refine.Max.Gradient
<code>def.max.dose.error, max.dose.error</code>	Grid <Field> Refine.Max.DoseError
<code>min.value</code>	Grid <Field> Refine.Min.Value
<code>max.value</code>	Grid <Field> Refine.Max.Value
<code>target.length</code>	Grid <Field> Target.Length
<code>target.length.scaling</code>	Grid <Field> Target.Length.Scaling

Localizing Adaptive Meshing Using `refinebox` Command

Adaptive meshing has been implemented through generalized refinement boxes. As such, adaptive refinement and the refinement parameters themselves can be set in a boxwise manner. The default adaptive refinement box covers the entire structure and relies on global parameters and field-based parameters for its default values. If you specify an adaptive refinement box, the default box is not created.

NOTE For field-based refinement, any adaptive `refinebox` that is manually created overrides the default adaptive `refinebox`. The default adaptive `refinebox` (that covers the entire structure) can be created explicitly with the `refinebox adaptive` command.

11: Mesh Generation

Adaptive Mesh Refinement

You can create one or more adaptive refinement boxes with different parameters. The most commonly used parameters control the size of the box (`min` and `max`), and the minimum and maximum edge lengths (`refine.min.edge` and `refine.max.edge`).

The default list of fields upon which to refine includes all solution fields. This list can be modified in several ways. For example, the following command overrides the default list:

```
pdbSet Grid <field> DoNotAdapt 1
```

The next example adds `Field1` and `Field2` to the default list for this particular box:

```
refinebox refine.add.fields= { Field1 Field2 ... }
```

The following command redefines the list of fields to be used as the basis for refinement; if set, this command overrides any add or subtract settings for this particular box:

```
refinebox refine.fields= { Field1 Field2 ... }
```

Examples

To switch on adaptive meshing, use:

```
pdbSet Grid Adaptive 1
```

To apply adaptive meshing only inside a box and to set the anisotropic edge minimum in the same box, use:

```
refinebox min= {0.0 0.0} max= {0.01 0.5} refine.min.edge= {0.001 0.25} adaptive
```

To refine only considering arsenic and boron, use:

```
refinebox refine.fields= {Arsenic Boron} adaptive
```

To create a default box and, in addition, to create a refinement box where r_F is modified locally for all species and α_F is modified for only boron, use:

```
refinebox adaptive
refinebox min= {0.0 0.0} max= {0.01 0.5} def.rel.error= 0.9 \
abs.error= {Boron= 1.0e14} adaptive
```

Adaptive Meshing During Analysis

Adaptive meshing during thermal and electrical analysis is switched on by default when adaptive meshing is switched on (in other words, `pdbGet Grid Adaptive` returns 1). An additional control that prevents adaptive meshing at low temperatures is specified as:

```
pdbSet Grid Min.Adaptive.Temp <Temp C>
```

The command:

```
pdbSet Compute Compute.Regrid.Steps 10      ;# during inert annealings
```

sets the fixed interval of time steps between meshing. The default value for the parameter `Compute.Regrid.Steps` is 4. After the specified number of steps, the mesh is checked to decide whether the refinement criteria are satisfied (within some tolerance); a remesh is performed if necessary.

The refinement criteria check is performed as follows: Axis-aligned edges are checked to see whether they satisfy:

$$\text{actual} < \text{Refine.Factor} * \text{error} / \text{maxerror} \quad (110)$$

where:

- `Refine.Factor` is a direction-dependent parameter of the PDB under `Grid`.
- `error` is the error functions given in [Eq. 101–Eq. 106](#).
- `maxerror` is the maximum error parameter associated with each refinement type.
- `actual` is the ‘actual’ edge length.

The parameter `Grid Refine.Percent` limits the percentage of edges that fail ([Eq. 110](#)) before a remesh is called. This check procedure is performed for every `Compute Compute.Regrid.Steps` whether a remesh is called or not. You can omit the refinement criteria check (which can be time-consuming for large meshes) and force a remesh by setting:

```
pdbSet Grid Refine.Check 0
```

[Table 16](#) summarizes the parameters available for adaptive meshing.

Table 16 Adaptive meshing parameters

Parameter	Comment
<code>Compute Compute.Regrid.Steps</code>	Number of solution steps before refinement criteria are checked to determine whether remeshing is required.
<code>Grid Refine.Check</code>	If this parameter is set to 1 (true), refinement criteria are checked and remeshing occurs if necessary. If it is set to 0 (false), remeshing occurs without a check.
<code>Grid Refine.Factor</code>	Tolerance factor for marking an edge as too long.
<code>Grid Refine.Percent</code>	Allowed percentage of edges that can be too long.

Tips for Adaptive Meshing

Some useful suggestions when using adaptive meshing are:

- When setting boxwise meshing criteria, remember that any global criteria you have specified still apply inside the box. This means you cannot use boxwise meshing criteria to establish less stringent meshing criteria (such as a larger relative error) inside a box because the more stringent global criteria still apply. If you want to use different criteria for different parts of the structure, set the global criteria to the least stringent criteria and use boxes for more stringent criteria.
- To switch on adaptive meshing and use all the defaults, all that is needed is `pdbSet Grid Adaptive 1`. The main parameter for adjusting the amount of refinement is `pdbSet Grid AdaptiveField Refine.Rel.Error`, which defaults to 1.5. In many cases, this does not refine sufficiently. Decreasing the value causes more refinement. The number of mesh points is sensitive to this value, and it is not generally recommended to use a value less than 0.25. This parameter generally meshes doping gradients well, but may leave the peaks too coarse. To refine the peaks, the best criterion to use is maximum dose error (`Grid AdaptiveField Max.Dose.Error`).
- To override the default refinement box used for field-based refinement (which covers the whole structure and applies to all solution variables), you need only to create an adaptive refinement box. To add criteria in addition to the default criteria, for example, to add finer criteria under the gate while preserving standard parameters elsewhere, you can create your own default refinement box.

For example:

```
refinebox adaptive  
refinebox min= {-0.01 -0.01} max= {0.15 0.05} adaptive def.rel.error= 0.75
```

- The default refinement setting for a solution field can give too fine a mesh. Increase `Grid AdaptiveField Refine.Target.Length` from the default value of 0.002 to reduce refinement.

This command specifies refinement at all interfaces to both `<material1>` and `<material2>`.

```
refinebox interface.mat.pairs= {<material1> <material2>}
```

This command specifies interface refinement at all interfaces where one side of the interface is `<material1>` and the other side is `<material2>`.

The interfaces that are refined are the union of `interface.materials` (all interfaces touching materials in the list) and `interface.mat.pairs` (only refined on material pairs found in the list first and second, third and fourth, and so on).

The default `min.normal.size` for all interface refinement boxes including the default ones is taken from the `pdb` parameter `Grid SnMesh min.normal.size`. Similarly, the default

value of `normal.growth.ratio` for all interface refinement boxes is taken from the `pdb` parameter `Grid SnMesh normal.growth.ratio.2d` in two dimensions and from `Grid SnMesh normal.growth.ratio.3d` in three dimensions.

To add an interface refinement, use the `refinebox` command.

To remove an existing interface refinement, first execute `refinebox clear`, and then start again.

Examples

The interfaces to be refined are defined as follows:

```
# Change the default min.normal.size (in micrometers)
pdbSet Grid SnMesh min.normal.size 2.0e-3

# Now modify which materials to apply interface refinement
# refine at all interfaces to silicon and poly (use the global min.normal.size
# and normal.growth.ratio)
refinebox clear
refinebox interface.materials= {silicon poly}
```

The next example shows refinement only at the silicon–oxide and polysilicon–oxide interfaces, and specifies a local value for interface refinement parameters:

```
refinebox clear
refinebox min.normal.size= 0.005 normal.growth.ratio= 3 \
  interface.mat.pairs= {Silicon Oxide PolySilicon Oxide}
```

Manipulating Refinement Boxes: transform.refinement Command

Transformations can be performed on refinement boxes using the `transform.refinement` command (see [transform.refinement on page 671](#)), which works like the `transform` command, except for refinement boxes.

The `transform.refinement` command accepts the following:

- Transformation can be either cut, flip, reflect, rotate, stretch, or translate.
- Arguments depend on the type of transformation.
- `name` – applies the transformation to a particular refinement box if specified or to all refinement boxes otherwise.
- `name.new` – specifies the name of the transformed refinement box.

11: Mesh Generation

Settings for Sentaurus Mesh

- `keep.original` – specifies whether to retain the original refinement box. Specifying `keep.original` preserves the original refinement box and creates a new transformed one. Specifying `!keep.original` transforms the specified refinement box, This option is useful when you want to ‘copy and paste’ refinements by, for example, translating them to a different position while keeping the original in place.

For example, the following command creates a new refinement box called `newRefBox`, which is identical to `refbox` but is displaced 0.1 μm in the x-direction:

```
transform.refinement name= "refbox" name.new= "newRefBox" \  
  translate= {0.1 0 0} keep.original
```

Settings for Sentaurus Mesh

Table 17 lists the parameters available for Sentaurus Mesh. To set the parameters, use:

```
pdbSet Grid SnMesh <Parameter> <value>
```

Table 17 Parameters available for Sentaurus Mesh

Parameter	Default	Description
<code>AllowRegionMismatch</code>	false	<p>If this parameter is set to true, when Sentaurus Mesh checks whether the number of regions in the input boundary and the number of regions at the end of the meshing process are the same, if there is a difference between the numbers of regions, Sentaurus Mesh will ignore the discrepancy, and the meshing process will continue. If this parameter is set to true and <code>MinimumRegionMismatchVolume</code> has also been specified, Sentaurus Mesh checks the volumes of all deleted regions:</p> <ul style="list-style-type: none">• If the volume of a deleted region is <i>less than the value</i> specified by <code>MinimumRegionMismatchVolume</code>, the meshing process will continue and the number of deleted regions is reported.• If the volume of a deleted region is <i>greater than the value</i> specified by <code>MinimumRegionMismatchVolume</code>, the meshing process will stop. <p>By default (when this parameter is set to false), when Sentaurus Mesh checks whether the number of regions in the input boundary and the number of regions at the end of the meshing process are the same, if there is a difference between the numbers of regions, the meshing process will stop.</p>
<code>Apply.Brep.DelPSC</code>	false	<p>Indicates whether the Delaunay refinement for piecewise smooth complex (DelPSC) algorithm is applied to the boundary at the beginning of a mesh generation step. DelPSC multithreading is controlled by:</p> <pre>math numThreadsSnMesh</pre>

Table 17 Parameters available for Sentaurus Mesh

Parameter	Default	Description
Apply.Brep.DelPSC.Accuracy	1e-4 μm	Specifies the accuracy used by the DelPSC algorithm when approximating high-curvature areas. In general, setting this parameter to 2% of the radius of curvature is appropriate. Larger values allow the DelPSC algorithm to run faster but generate coarser discretization on curved surfaces. Smaller values make the DelPSC algorithm run slower and generate finer discretization on curved surfaces.
Apply.Brep.DelPSC.Resolution	1e-2 μm	Controls the size of small triangles in the DelPSC algorithm. In general, setting this parameter to 10% of the radius of curvature is appropriate. Larger values allow the DelPSC algorithm to run faster but generate bigger triangles next to geometric features and triple lines on curved surfaces. Smaller values make the DelPSC algorithm run slower and generate smaller triangles next to geometric features and triple lines on curved surfaces.
Apply.Brep.DelPSC.Ridge.Angle	150 degrees	Angle threshold to detect geometric features.
binarytree.split.box		Specifies the region where the binary-tree split factors are applied. In two dimensions: <code>pdbSet Grid SnMesh binarytree.split.box \</code> <code>{xmin 0.0 ymin 0.0 xmax 1.0 ymax 1.0}</code> In three dimensions: <code>pdbSet Grid SnMesh binarytree.split.box \</code> <code>{xmin 0.0 ymin 0.0 zmin 0.0 xmax 1.0 ymax 1.0 \</code> <code>zmax 1.0}</code>
binarytree.split.factor.x	1	Splits the final binary tree used for refinement by the specified factor in the x-direction. Powers of 2 must be used.
binarytree.split.factor.y	1	Same as <code>binarytree.split.factor.x</code> but for the y-direction.
binarytree.split.factor.z	1	Same as <code>binarytree.split.factor.x</code> but for the z-direction.
CoplanarityAngle	175 degrees	Any pair of faces with an angle of <code>CoplanarityAngle</code> or more will be considered coplanar.
CoplanarityDistance	1.0e-6 μm	Maximum deformation caused to the boundary when swapping the edge shared by a pair of adjacent faces.
DecimateBeforeImprint	true	Decimates the boundary before imprinting it with the axis-aligned mesh.
DelaunayTolerance	1.0e-4	Specifies how close the ridges and boundary faces conform to the Delaunay criterion.

11: Mesh Generation

Settings for Sentaurus Mesh

Table 17 Parameters available for Sentaurus Mesh

Parameter	Default	Description
DelaunayToleranceMat		Specifies an array pair of materials and tolerances to be used in those materials. For example: <pre>pdbSet Grid SnMesh DelaunayToleranceMat {Silicon 0.01 Oxide 1.0}</pre>
DelaunayToleranceReg		Specifies an array pair of regions and tolerances to be used in those regions.
DelaunayType	conformal	Types of mesh generated by Sentaurus Mesh. Available types are box method, conformal, or constrained.
DelPsc	false	Indicates whether the Delaunay refinement for piecewise smooth complex (DelPSC) algorithm is applied to the boundary at the beginning of a mesh generation step.
DelPscAccuracy	1e-4 μm	Specifies the accuracy used by the DelPSC algorithm when approximating high-curvature areas. This parameter is used during standard mesh generation (as opposed to using DelPSC during moving boundary).
EdgeProximity	0.05	Specifies the minimum ratio of the edges generated when an edge is split.
FaceProximity	0.05	Specifies the minimum ratio of the faces generated when a face is split.
ImprintAccuracy	1e-5 μm	Distance used to determine when two points are too close during imprinting.
ImprintCoplanarFacesOnly	true	Imprints the binary tree on the coplanar sets of faces. This is useful to avoid over-refinement in curved areas.
ImprintCoplanarityAngle	179 degrees	Angle used to decide when two faces are coplanar. If two adjacent faces have an angle greater than this value, they will be added to the set of faces to be imprinted with the binary refinement tree cells.
ImprintCoplanarityDistance	1e-4 μm	Distance used to determine when two faces are coplanar. If the distance from one face to the plane of the other face is less than this value, they will be added to the set of faces to be imprinted with the binary refinement tree cells.
max.box.angle.2d	120 degrees	Maximum angle in binary tree (2D only).
max.box.angle.3d	150 degrees	Maximum angle in binary tree (3D only).
max.lateral.size	10 μm	Specifies the maximum lateral spacing between elements in the direction parallel to the interface.
MaxAspectRatio	1e6	Specifies the maximum-allowed aspect ratio of an element in the binary tree.

Table 17 Parameters available for Sentaurus Mesh

Parameter	Default	Description
MaxBoundaryCutRatio.2d	0.01	Specifies the maximum-allowed ratio between the lengths of adjacent axis-aligned edges cutting material boundaries (2D only).
MaxBoundaryCutRatio.3d	0.01	Specifies the maximum-allowed ratio between the lengths of adjacent axis-aligned edges cutting material boundaries (3D only).
MaxConnectivity	1e37	Specifies the maximum number of elements connected to a point in the final mesh.
MaxNeighborRatio	3.0	Binary-tree smoothing is performed after refinements have been added to the binary tree. This prevents sudden changes in the element size that can be especially detrimental to mechanics results. The ratio of neighboring collinear edges in the binary tree can be adjusted with this parameter.
MaxPoints	500000	Maximum number of points allowed by the Sentaurus Mesh delaunization module.
MaxSolidAngle	360 degrees	Specifies the maximum solid angle allowed in the elements of the mesh (3D only).
MaxTetQuality	1e37	Specifies the maximum circumscribed sphere radius-to-shortest edge ratio allowed in the mesh (3D only).
min.normal.size	8e-4 μm	Specifies the smallest normal (to the interface) mesh element size on either side of an interface.
MinAngle	0	Specifies the minimum angle allowed in the elements of the mesh (2D only).
minedge	2.0e-6 μm	Minimum edge length request.
MinimumRegionMismatchVolume	0	Specifies a region volume that Sentaurus Mesh uses when checking deleted regions. It is used in conjunction with <code>AllowRegionMismatch</code> set to true.
normal.growth.ratio.2d	2.0	Factor used to increase the size of the elements in the direction normal from the interface. The size of each layer equals the size of the previous layer multiplied by this factor (2D only).
normal.growth.ratio.3d	3.0	Factor used to increase the size of the elements in the direction normal from the interface. The size of each layer equals the size of the previous layer multiplied by this factor (3D only).
offsetting.maxlevel	3	Specifies the number of offsetting layers at an interface when Sentaurus Mesh offsetting is used at an interface.
SliverAngle	175 degrees	Limits the maximum dihedral angle on one element when the delaunizer performs the sliver removal step.

11: Mesh Generation

Displaying Refinement Boxes

Table 17 Parameters available for Sentaurus Mesh

Parameter	Default	Description
SliverDistance	1e-2 μm	Limits the amount of “damage” done to the standard Voronoi diagram by the sliver removal algorithm. Note that the grid produced by the sliver removal algorithm is weighted Delaunay, so the standard Voronoi diagram is “damaged” unless the Voronoi weights are stored (see the StoreDelaunayWeight parameter). When the box method library reads those weights, it calculates the correct Voronoi diagram and coefficients to solve the PDEs.
SliverRemovalAlgorithm	1	Algorithm for sliver removal, either 1 (original) or 2 (reduced non-Delaunay elements).
StoreDelaunayWeight	0	When set to 1, stores the Delaunay–Voronoi weight (DelVorWeight) for the box method library.
UseLines	true	Specify 1 or 0. UseLines is specified in the line command in the mesh generated by Sentaurus Mesh.

Displaying Refinement Boxes

Refinement boxes are saved to TDR files for displaying in Sentaurus Visual. When loading a TDR file in Sentaurus Visual, if refinement boxes are present, RefinementBox will be listed on the **Materials** tab (see [Figure 32 \(left\)](#)). The list of individual refinement boxes is found on the **Lines/Particles** tab (see [Figure 32 \(right\)](#)).



Figure 32 (Left) Materials tab showing RefinementBox and (right) Lines/Particles tab showing refinement boxes

By default, refinement boxes are not displayed. They may be displayed in bulk or with borders. [Figure 33 on page 193](#) shows a 2D structure with two refinement boxes and its corresponding mesh. [Figure 34 on page 193](#) shows a 3D structure with two refinement boxes and its corresponding mesh.

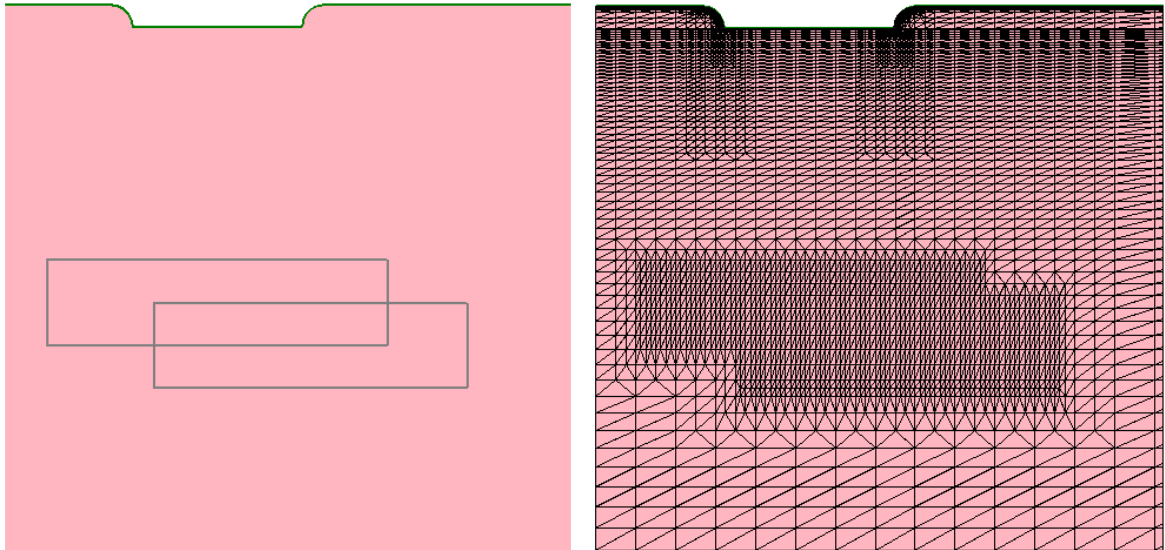


Figure 33 (Left) Two-dimensional structure with two refinement boxes and (right) corresponding mesh

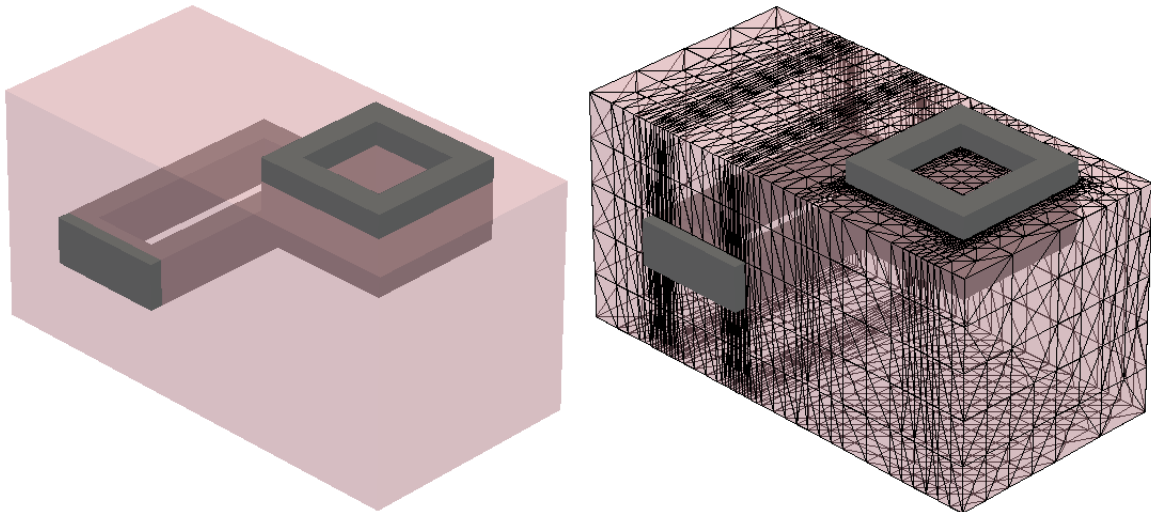


Figure 34 (Left) Three-dimensional structure with two refinement boxes and (right) corresponding mesh

Controlling the Mesh During Moving-Boundary Problems

Moving-boundary problems create new regions and dramatically alter the shape of existing ones. Controlling the mesh is important. This section covers some mesh control methods.

NOTE The moving boundary in Sentaurus Interconnect is switched off by default. Use `pdbSet Grid Modify.Mesh 1` to switch it on.

TSUPREM-4 Moving-Boundary Meshing Library

The TSUPREM-4 moving-boundary meshing library is available from within Sentaurus Interconnect (hereafter, referred to as the *TS4 mesh library*). By default, the TS4 mesh library performs all mesh updates in 2D moving-boundary oxidation simulations.

Control Parameters in TS4 Mesh Library

The control parameters of the TS4 mesh library are specified with:

```
pdbSet Grid TS4Mesh <control parameter> <value>
```

The available control parameters are:

`MergeSubAndAdd <0|1>` (default: 1)

It optimizes the speed performance by merging the grid subtraction and addition procedures.

`DoSubAfterStep <0|1>` (default: 0)

Grid points that are too close to the moving boundary are removed after each solve step, while grid addition is performed after each mechanics step followed by the solve step. Switching on this flag forces only one solve step per each mechanics step. When this flag is switched on, `MergeSubAndAdd` is ignored.

`SubTimeFactor <double>` (default: 1.5)

The time step given by mechanics for grid removal is scaled by `SubTimeFactor`.

NOTE Do not change the default.

`MinSpaceOnInterface <double>` (default: 2e-6 [μm])

The nodes on an interface mesh must be rebuilt after meshing on the moving boundary since the bulk meshes along the interface can be added or removed. Instead of destroying

and rebuilding the interface mesh, the TS4 mesh library tries to reuse the original node data on the interface mesh to minimize the interpolation error. The original nodes are detected when the location difference is less than `MinSpaceOnInterface`.

NOTE Do not change the default.

`ExactGridSpace <0|1> (default: 1)`

On the growing material side of the interface, the triangular mesh elements expand. To maintain solution accuracy in the material, you must add nodes to the growing material. The addition of nodes to the growing material is controlled by `perp.add.dist`, `ExactGridSpace`, and `LocalGridSpace`.

NOTE `perp.add.dist` is the grid control parameter of each material, for example: `pdbSet Oxide Grid perp.add.dist 0.01e-4`

Precise grid spacing is obtained by adding new nodes in a growing layer at the distance specified by `perp.add.dist` from the existing node in the layer. Because only one node can be added at each point on an interface during a simulation time step, the size of the time step may need to be reduced to achieve the required spacing. This reduction in the time step size can be disabled by specifying:

```
pdbSet Grid TS4Mesh ExactGridSpace 0
```

By default, `ExactGridSpace` is set to 1 to allow reducing the size of the time steps to control the grid spacing. The algorithm does not allow grid points to be added at spacings less than 1 Å, and control of the spacing may not be precise for spacings less than 2 Å.

`OrderFlatTri <0|1> (default: 1)`

When the area of a shrinking triangle becomes less than $1e-15$ (cm²) after a time step, the triangle is removed. When the shrinking triangle to be removed is located at a material interface and the removal of the triangle will result in a bad mesh, the material type of the shrinking triangle is replaced with the type of the growing neighbor material, instead of removing it. When those triangles are adjacent to each other, the reordering algorithm for the replacements smooths the interface shape after conversion.

`MinAreaRemovalRatio <double> (default: 10.0)`

When a region has only one triangle surrounded by neighbors of different materials and its area is less than `MinAreaRemovalRatio` multiplied by $1e-15$ (cm²), the material type of the triangle is replaced with the neighbor material that shares the longest edge with the triangle.

`Min.Split.Distance <double> (default: 1e-8 [cm])`

When multiple regions with the same material meet at one point, the point is split by inserting new elements. The parameter determines the minimum split distance.

Moving Mesh and Mechanics Displacements

The displacements computed by the mechanics solution are applied to the nodes after checking against the `MinimumVelocity` criterion defined for each region. Velocity is the computed solution variable and is multiplied by the time step to compute displacements. The nodes are moved by this amount.

The computed velocities are compared against `MinimumVelocity` and, if the computed velocity is greater than `MinimumVelocity`, the displacements are computed and applied. The `MinimumVelocity` is set with the command:

```
pdbSetDouble Silicon Grid MinimumVelocity <n>
```

The moving boundary is switched off by default. It can be switched on using the command:

```
pdbSet Grid Modify.Mesh 1
```

If the moving boundary is required, the mechanics step is performed before the solve step, since the mesh is moved during the solve step. In addition, if the heat equation is solved, a heat step is included before the mechanics step.

NOTE Moving mesh is not supported in high-order elements or mixed meshes.

Controlling the Grid Spacing

Grid spacing in the growing region is controlled by `perp.add.dist`. Its unit is centimeter, and the edges in growing regions are checked to see whether they are nearly perpendicular to the interface.

If they are perpendicular, they are split if their length exceeds the value of `perp.add.dist`. This value is set with the command:

```
pdbSet Oxide Grid perp.add.dist 2e-7
```

Cleaning Up the Grid

During boundary movement, a region can increase at the expense of a shrinking region. The shrinking regions then have a problem of small edges. Below a certain value, these edges must be removed entirely, and the mesh around them must be adjusted.

The short edge criterion is specified by the `Remove.Dist` parameter, which is specified in centimeters and is set as follows:

```
pdbSet Silicon Grid Remove.Dist 3e-8
```

NOTE Due to mesh quality constraints, this value must be more than 2×10^{-8} cm.

Miscellaneous Tricks

Since Sentaurus Interconnect moving boundary does not allow the interface to traverse more than one element thickness at a time, speed can be achieved by having elements with longer edge lengths near the interface.

This can be controlled by refinement boxes or the `pdb` parameter `Grid SnMesh min.normal.size`. Large structures, like those used in power devices, may need `min.normal.size` of 0.01 μm , while submicron CMOS devices need 8 \AA .

The parameter `Grid SnMesh normal.growth.ratio.3d` controls the mesh away from the interface. If the mesh spacing does not increase fast enough, this number can be increased.

In large structures, the interface fidelity may not need to be as tight as that of 45-nm or 32-nm gate transistors. The `accuracy` argument of the `mgoals` command can be increased to 1 \AA (`mgoals accuracy= 1e-4`), which will cause MGOALS to clean up interfaces of small (sub-1 \AA) features.

These are options available to the simulation engineer; however, care must be exercised in varying these parameters since they may affect the final structure significantly.

Meshing for 3D Moving-Boundary Problems

Maintaining a conformal high-quality mesh during the simulation of a 3D moving mesh is very difficult because of the following requirements: moving boundaries, accurate profiles, dose conservation, minimization of the number of mesh points, and maintaining high-quality mesh elements. In particular, handling the frequent collision of the moving front with points inside the other regions can cause intractable problems for the local mesh operations needed for maintaining dose conservation.

Before each time-step, the mesh is checked for the maximum-possible time step until the first tetrahedral element collapses (becomes flat) in the material being consumed (that is, silicon). If necessary, the time step is reduced. During simulation, all mesh points are moved using the velocity and the time step. The mesh topology is not changed during the solve time step. At the

11: Mesh Generation

Controlling the Mesh During Moving-Boundary Problems

end of the solve time step, the mesh quality is improved and flat elements are removed, allowing the next time step to be sufficiently large.

MovingMesh Algorithm

NOTE The MovingMesh algorithm is experimental and requires careful parameter settings. Contact TCAD Support for technical assistance.

The MovingMesh algorithm is used for 3D moving-boundary problems such as mechanical bending. It can be switched on and off with the following command before a `solve` command:

```
pdbSet Grid Use.MovingMesh 1      ;# switched on by default
```

Three important parameters control MovingMesh:

```
pdbSet Oxide Grid perp.add.dist 0.005e-4      ;# centimeter
pdbSet Grid Remove.Dist 0.001e-4              ;# centimeter
pdbSet Grid MovingMesh Remove.Dist.On.Interface 0.0001e-4 ;# centimeter
```

The `perp.add.dist` parameter specifies the distance (in cm) that the oxide interface can move before new mesh points are inserted in the oxide.

The `Remove.Dist` parameter specifies the shortest distance (in cm) the mesh vertices are allowed from the interface into the bulk. For lengths shorter than this distance, the vertices will be removed. Do not specify a distance larger than the minimum material thickness, which is typically the thickness of the native layer. For a typical example with 1.5 nm native layer, `Remove.Dist` of 1.0 nm or less is appropriate.

The `Remove.Dist.On.Interface` parameter controls small triangles on material interfaces. Triangles with an edge shorter than this distance will be removed. A smaller number makes the interfaces smoother, but results in a larger number of triangles.

The moving interfaces can develop problematic geometric features such as knife edges, noisy surfaces, or extremely thin gaps. You can enable geometry repair and surface remeshing by using:

```
pdbSet Grid MovingMesh Repair.Geometry 1      ;# switched on by default
```

The criteria to trigger geometry repair are based on the minimum dihedral angle and the maximum face angle:

```
pdbSet Grid MovingMesh Repair.Geometry.Min.Dihedral.Angle 5 ;# degree
pdbSet Grid MovingMesh Repair.Geometry.Max.Face.Angle 175 ;# degree
```

If the minimum dihedral angle between two triangles is below the threshold or the maximum face angle of a triangle is above the threshold, the geometry repair procedure starts.

The geometry repair procedure involves a multimaterial level-set (MLS) formulation. The resolution of the level-set cell size is controlled by:

```
pdbSet Grid MovingMesh Repair.Geometry.Resolution 0.001 ;# micrometer
```

NOTE The parameter `Repair.Geometry.Resolution` must be, at most, one-third the thickness of the thinnest region. Otherwise, the thin region may be considered noise, and it disappears.

NOTE The amount of geometry smoothing performed by the MLS algorithm depends on both the curvatures in the input and the level-set cell size. A noisy surface has a high curvature, so it will be smoothed to a large extent to remove noise. On the other hand, a planar surface has zero curvature and is well preserved. Unfortunately, a sharp corner has a theoretically infinite curvature, so it will become a rounded corner. When the corner is rounded, the next iteration of MLS smoothing will have less effect. The specified level-set cell size is the threshold to distinguish between the noise to be removed and the features to be preserved.

The boundary representation (brep) of the new geometry must go through a meshing algorithm for curved surfaces called the Delaunay refinement for piecewise smooth complex (DelPSC) that improves the quality of triangles on brep surfaces. This algorithm is enabled by:

```
pdbSet Grid MovingMesh Apply.Brep.DelPSC 1 ;# switched on by default
```

The DelPSC algorithm performs adaptive sampling on ridges (1D geometric feature) according to the refinement fields, the curvatures of the ridges, and the proximity among the ridges. On each surface patch (2D geometric feature), the DelPSC algorithm performs adaptive sampling according to the refinement fields and the curvatures of the surface. Multithreading in the DelPSC algorithm is controlled by `math numThreadsSnMesh`.

Ridge sampling also is controlled by:

```
pdbSet Grid MovingMesh Apply.Brep.DelPSC.Resolution 0.005 ;# micrometer
```

The above parameter ensures no ridge edge will be longer than the specification. It is useful, for example, when you have a straight line (no curvature) next to curved surfaces. You want the sampling points on the straight line to be fine enough to support the adjacent curved surfaces.

Note that you no longer require `Apply.Brep.DelPSC.Resolution` to be as small as the thin native layer thickness, because of the adaptive sampling based on proximity between nearby ridges.

11: Mesh Generation

Controlling the Mesh During Moving-Boundary Problems

To control accuracy in high curvature areas, you can specify the acceptable distance between the old and new curved surfaces using:

```
pdbSet Grid MovingMesh Apply.Brep.DelPSC.Accuracy 0.0001 ;# micrometer
```

NOTE Every time the DelPSC algorithm is applied, the new curved surface can deviate from the old curved surface by, at most, the value of `Apply.Brep.DelPSC.Accuracy`. New vertices lie exactly on the old surface, but new triangles cannot lie exactly on the old surface *unless* the old surface is flat. In general, the smaller the value of `Apply.Brep.DelPSC.Accuracy` is, the smoother the new surface becomes, and the more accurate the new surface represents the old surface.

These are typical settings for a small transistor structure:

```
pdbSet Oxide Grid perp.add.dist 2e-7 ;# cm 2nm
pdbSet Silicon Grid perp.add.dist 1e-6 ;# cm 10nm
pdbSet Grid Remove.Dist 9e-8 ;# cm 9A
pdbSet Grid MovingMesh Remove.Dist.On.Interface 3e-8 ;# cm 3A
pdbSet Grid MovingMesh Repair.Geometry.Resolution 3e-4 ;# um 3A
pdbSet Grid MovingMesh Apply.Brep.DelPSC.Resolution 3e-3 ;# um 3nm
pdbSet Grid MovingMesh Apply.Brep.DelPSC.Accuracy 1e-4 ;# um 1A
```

These are typical settings for a large power structure.

```
pdbSet Oxide Grid perp.add.dist 2e-6 ;# cm 20nm
pdbSet Silicon Grid perp.add.dist 1e-5 ;# cm 100nm
pdbSet Grid Remove.Dist 9e-7 ;# cm 9nm
pdbSet Grid MovingMesh Remove.Dist.On.Interface 3e-7 ;# cm 3nm
pdbSet Grid MovingMesh Repair.Geometry.Resolution 3e-3 ;# um 3nm
pdbSet Grid MovingMesh Apply.Brep.DelPSC.Resolution 3e-2 ;# um 30nm
pdbSet Grid MovingMesh Apply.Brep.DelPSC.Accuracy 1e-3 ;# um 1nm
```

MovingMesh has facilities for troubleshooting runtime failures. A typical setting would be:

```
# Switch on level-1 diagnostics
pdbSetDouble debugLevel MovingMesh 1 ;# switched off by default

# Save intermediate result every 100 time steps
pdbSetDouble Grid MovingMesh Save.Interval 100 ;# switched off by default

# Save diagnostic files for Repair.Geometry and DelPSC
pdbSetBoolean Grid MovingMesh Repair.Geometry.Monitor 1
;# switched off by default
```

In level-1 diagnostics, the intermediate result will be saved after a certain number of time steps in the files:

```
<NodeName>_MovingMeshGridTimeStep<xxxx>.tdr
```

The frequency of saving is specified by the `Save.Interval` parameter. The files are written after mechanics and before diffusion to analyze the grid-limited time step.

In the event of failure in `Repair.Geometry`, the level-1 diagnostics will save files with names such as:

```
<NodeName>_remeshBrep{In,MLS,PSC,Out}.tdr
<NodeName>_applyBrepDelPSC{In,Out}.tdr
```

They are useful for checking whether the resolution parameters are adequate. The most likely cause of failure is a too coarse resolution to capture thin oxide layers and other small geometric features.

If the parameter `Repair.Geometry.Monitor` is set to 1, the intermediate files with names such as:

```
<NodeName>_remeshBrep_<xxxx>_{In,MLS,PSC,Out}.tdr
<NodeName>_applyBrepDelPSC_<xxxx>_{In,Out}.tdr
```

will be saved every time the repair geometry operation is triggered. These files are useful to monitor how the MLS and DelPSC algorithms perform at various points in a simulation.

UseLines: Keeping User-Defined Mesh Lines

During the `init` command, the line location and spacing specifications given by `line` commands are expanded into ticks and stored in the PDB and in TDR files. This is performed by default.

By carefully placing lines, you can isolate areas of the structure that changed (because of etching, deposition, and so on) from those that do not (such as bulk silicon). In this way, the mesh in areas that do not change will have the least amount of change, the least interpolation, and the most accurate results. Even the mesh in regions that do change will have a similar starting point and should also have minimal mesh-point movement coming from remeshing.

In the simplest case, all the `line` commands are specified before the `init` command, and they are saved and reused every time a remesh is performed. However, there are other cases described in the following sections that allow this feature to be more powerful.

Using line Commands After init Command

The expansion of lines from `line` commands into ticks (in other words, all starting mesh line locations) is performed only at the point that the lines in that direction are needed. For example,

11: Mesh Generation

UseLines: Keeping User-Defined Mesh Lines

x-lines are always expanded in the `init` command, but y-lines are only expanded when the first etch with a mask is given. Therefore, it is possible to load a 1D structure, give y-lines, and then expand to 2D, or give both y- and z-lines and expand to 3D.

After a particular direction or dimension is expanded, it is only possible to insert one tick at a time in that direction using the `line` command (in other words, the `spacing` parameter is thereafter ignored).

This is handled by specifying one of the following:

- The `line` command in a dimension greater than the current dimension. For example, a y-line specification when the simulation is one dimensional.
- The `line` command in a dimension at or less than the current dimension. For example, either an x-line or a y-line specified when the simulation is two dimensional.

For more information, see [Automatic Dimension Control on page 36](#).

Dimension Within Current Spatial Dimension

This is encountered if the user-specified x-lines and the current spatial dimension of analyses is 1D. Or, it could happen if you specify x- or y-lines in 2D, or x-, y-, or z- lines in 3D.

In this case, the `line` command ignores the `spacing` parameter and tries to insert only one tick as long as that tick (line) is not too close to an existing tick.

Inside the `init` command, the `line` commands are expanded into ticks using the `spacing` specifications for the dimension as they are needed. When additional `line` commands are given for dimensions where the ticks have already been expanded, the `spacing` parameter is ignored and one additional tick is added as long as it is not too close to an existing tick.

Dimension Greater Than Current Spatial Dimension

This is encountered if you specified y- or z-lines, and the current spatial dimension of analyses is 1D. Or, it could happen if you specify z-lines and the current spatial dimension of analyses is 2D.

In this case, the `line` command is considered in its entirety, and the `spacing` parameter is used. All the intermediate lines are included in the list of ticks kept.

Using line Commands to Create Virtual Spacing

By default, the ticks created by the `line` command with the `spacing` argument are *hard* lines that Sentaurus Mesh enforces whenever possible. Since such lines traverse the entire structure, they can result in unnecessary refinement, especially if the `spacing` argument is used with small values.

Using the `virtual.spacing` option, the ticks created between `line` commands become *soft* lines. This means that the refinement lines created by Sentaurus Mesh, during the binary tree construction, will snap to these ticks, thereby building a more predictable pattern instead of the pattern generated by the default bisection algorithm.

For example, the following commands create a virtual spacing of 1 nm on the device between the (0, 0) and (1, 1) coordinates. All user-defined refinement will snap to the coordinates defined in this virtual grid:

```
line virtual.spacing
line x loc=0 spacing=0.001
line x loc=1 spacing=0.001
line y loc=0 spacing=0.001
line y loc=1 spacing=0.001
```

NOTE The `virtual.spacing` option is applied globally and affects all lines at the same time.

UseLines and the transform Command

The ticks must be handled in a special manner with the `transform` command (see [transform on page 661](#)).

Reflection

Using `transform reflect`, in the reflected region, the ticks are created after applying lateral inversion along the appropriate plane.

Stretch

On applying `transform stretch` at a given coordinate in a given direction, the existing ticks in the stretched area are translated by the amount of the stretch. You must insert lines in the stretched area appropriately.

11: Mesh Generation

UseLines: Keeping User-Defined Mesh Lines

Rotation

When applying `transform rotate`, the ticks also are rotated and properly transferred between x-ticks, y-ticks, and z-ticks.

Translation

The `transform translate` command shifts the ticks by the specified amount.

Cut

Using `transform cut`, the lines in the part of the structure that is cut are deleted.

Example: Testing line Commands

Use the following example to test line commands:

```
line x loc=0 tag=a spacing=0.05
line x loc=0.1 spacing=0.05
line x loc=1 tag=b spacing=0.05
line y loc=0 tag=c spacing=0.01
line y loc=0.6 tag=d spacing=0.1

region silicon xlo=a xhi=b ylo=c yhi=d
init !DelayFullD

deposit oxide thickness=0.002 iso
grid remesh info=2

line y loc=0.026
line y loc=0.027
line y loc=0.028
line y loc=0.029
line y loc=0.025
line y loc=0.024
line y loc=0.023

grid remesh

deposit poly thickness=0.18 iso
mask name=m1 left=-0.1 right=0.025
etch aniso thickness=0.2 poly mask=m1

struct tdr=linetest
```

Example: Showing Clearing Lines for a New Structure

Use the following example to show clearing lines and to prepare for another structure definition within the same command file:

```
line x loc=0 tag=a spa=0.125
line x loc=1 tag=b spa=0.125
line y loc=0 tag=c spa=0.125
line y loc=1 tag=d spa=0.125
region silicon xlo=a xhi=b ylo=c yhi=d
init

grid FullD

line clear

line x loc=0 tag=a spa=0.125
line x loc=1 tag=b spa=0.125
line y loc=0 tag=c spa=0.125
line y loc=1 tag=d spa=0.125
region silicon xlo=a xhi=b ylo=c yhi=d
init

line y loc=0.3 spa=0.01

grid FullD
```

Data Interpolation

Sentaurus Interconnect stores a copy of the mesh with all its data before performing any geometry-changing operation. This is the reference mesh used to interpolate data onto the new mesh. In three dimensions, a mesh is generated only when it is necessary, so you can have multiple `etch`, `deposit`, `photo`, and `strip` commands without the need to remesh in between. When a new mesh is required, data is interpolated from the stored mesh and data.

Data interpolation is performed materialwise. This is important because some nodal data can be discontinuous at material interfaces; for example, segregation causes a jump in concentration at the silicon–oxide interface. In addition, the precise location of an interface can change slightly due to numeric noise in geometry-moving algorithms. Therefore, it is necessary to allow the data to be interpolated from points in the old mesh nearby, but only from the same material.

Data also can be interpolated from materials that are *Like* materials (that is, the material in the old mesh is *Like* the material in the new mesh, or the material in the new mesh is *Like* the material in the old mesh). When interpolating data at an interface, the preference is to use data

11: Mesh Generation

Data Interpolation

from the same region, then data from the same material, and finally data from *Like* materials. If no match is found, then 0 is set for all data at that point.

For data defined on elements, the overlap of elements from the old mesh to the new mesh is used for weighting. Similar to nodal data, interpolation of elements near interfaces uses the region, material, and *Like* material preference order.

Multithreaded interpolation can be used to speed up interpolation in large 3D structures. Because of the memory-intensive nature of interpolation, the performance benefit of multithreading may saturate, but it depends strongly on the example. If the performance degrades when using a large number of threads, the following command can be used to reduce the number of threads for interpolation:

```
math numThreadsInterp=<i>
```

Data Interpolation Near Boundaries

Region boundaries in a mesh can move for various reasons, including:

1. Geometric changes such as etching or deposition.
2. Boundary cleanup operations designed to reduce the number of boundary points and, therefore, the overall number of mesh points.
3. Moving-boundary simulations such as oxidation, silicidation, and model 0 type epitaxy.

The interpolation module has no information about all the operations that may have occurred since the last bulk mesh was created. Therefore, a tolerance is used to distinguish between boundary points displaced by case (1) where the boundary is expected to move more than a small amount, so interpolated values should be taken from exact locations in the new mesh, and cases (2) and (3) where the boundary should move only a small amount, so the values at the boundary should be transferred by interpolating from the nearest location on the old boundary.

The tolerance is set with the following command, where the distance is measured in micrometers:

```
pdbSet Grid Interpolation.Search.Distance <n>
```

Figure 35 illustrates the meaning of this parameter.

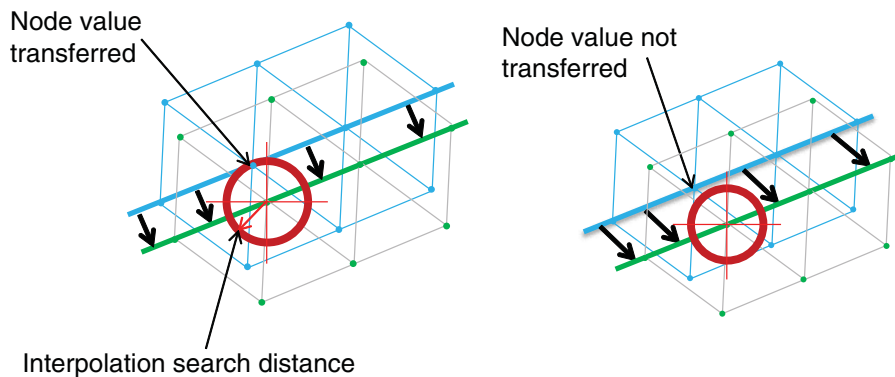


Figure 35 (Left) The new mesh (blue) has moved from its previous position, but the distance is smaller than Interpolation.Search.Distance, so values are interpolated from the green mesh to the blue mesh at the boundary. (Right) The distance is too large, so only those points in the blue mesh that are the same material as the green mesh are interpolated.

Troubleshooting

Sometimes, the mesh generation step fails and it is unclear what the problem could be. The following are recommendations of where to look when problems arise during meshing:

- Set InfoDefault to 2 or higher, for example:

```
pdbSet InfoDefault 2
```

- When Sentaurus Mesh prints the message:

```
"Short edge 1e-8 around points (x1, y1, z1) (x2, y2, z2)"
```

look at the input structure around the coordinates (x1, y1, z1) or (x2, y2, z2), and check whether there is a singularity in that area (such as a crack, fold, or surface overlap).

Sometimes, these singularities are the product of an etching or a deposition step, and action can be taken to improve the quality of the structure.

- Check the quality of the boundary printed for the steps preceding the mesh generation process. In particular, the following line provides an indication of quality (this is output if InfoDefault is 2 or higher):

```
minDihedralAngle: <angle> [near (x1, y1, z1), (x2, y2, z2)] at  

region=Nitride_1.
```

If you see an angle less than 3° in the geometry, this might indicate a problem in the structure at the given coordinates. Look at the preceding process steps in Sentaurus Visual to see whether they can be modified to avoid creating the problem.

11: Mesh Generation

Troubleshooting

To visualize the problem area in Sentaurus Visual, choose **View > Camera Configuration** to open the Camera Properties panel. On the panel, click the **Rotation Point** tab, enter one of the coordinates where the small dihedral angle is reported, and select the **Show** option under Point/Line. This will display a red 3D crosshair marker indicating the location of the center of rotation. You can hide all regions, except the one where the minimum dihedral angle is reported and zoom in around the red marker to locate the problem area. Sometimes you need to rotate the structure around the red marker to see what is happening to the geometry.

- You might need to add `!repair` to the `etch` or `deposit` command to ensure the structure is not repaired. This makes it easier to find the problem in Sentaurus Visual.
- Frequently save snapshots of the boundary file of the structure, especially before all mesh generation operations. This will help you to investigate possible problems in the input to the mesh generator. To accomplish this, use the command:

```
struct tdr.bnd=<c>
```

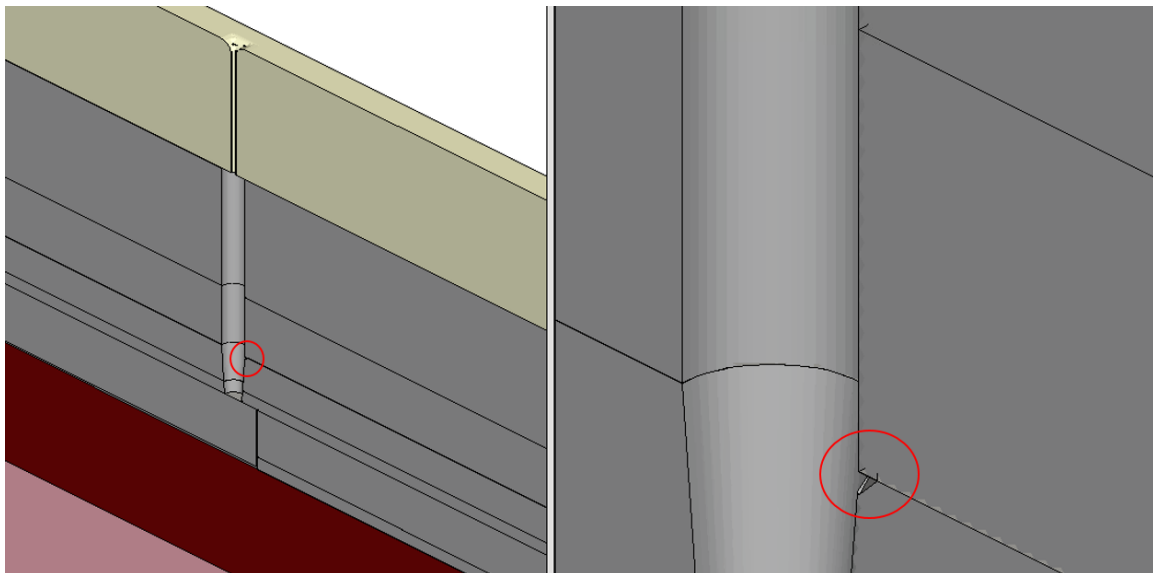


Figure 36 (Left) Example of artifact in geometry and (right) magnification of artifact (circled region)

This chapter describes etching, deposition, insertion, and other geometric transformations available in Sentaurus Interconnect.

Overview

During device fabrication, several etching and deposition steps are necessary. Such steps can be modeled in Sentaurus Interconnect by simple geometric operations or simple mathematical formulations in which no physical processing is simulated.

The creation of complex 3D shapes can be assisted by an interface to Sentaurus Structure Editor (see [Sentaurus Structure Editor Interface on page 277](#)) or the insertion of predefined pieces of a structure (see [Inserting Polygons on page 274](#) and [Inserting Polyhedra on page 274](#)). A shape library is available for commonly used structures (such as shallow trench isolation). These shapes are already parameterized (see [Shape Library on page 238](#)).

For physical etching and deposition, Sentaurus Interconnect provides interfaces to Sentaurus Topography and Sentaurus Topography 3D (see [Sentaurus Topography Interface on page 280](#) and [Sentaurus Topography 3D Interface on page 282](#)).

Sentaurus Interconnect provides several etching and deposition operations, in addition to purely geometric operations to help shape the geometry of devices:

- Etching is described in [Etching on page 210](#).
- Deposition is described in [Deposition on page 229](#).
- Masks offer an effect (similar to a masking layer) to limit the etching or deposition process to a certain window or to provide a convenient way to mimic lithographic patterning (see [Masks and Lithographic Patterning on page 251](#)).
- Several geometric transformations are available (see [Geometric Transformations on page 259](#)).
- The internal module, MGOALS, which is used to perform geometric etching and deposition, is described in [MGOALS on page 264](#).

Etching

Three main specifications are required for all etching steps:

- Etching type
- Material or materials to be etched
- Amount of material to be removed

The supported etching types are specified using the `type` argument of the `etch` command.

Table 18 Supported etching types

Etching type	Description
<code>type=angles.rates</code>	Etches according to a definition of a piecewise linear etching rate. See Piecewise Linear Etching on page 227 .
<code>type=anisotropic</code>	Etches in the vertical direction only. See Anisotropic and Directional Etching on page 214
<code>type=cmp</code>	Performs chemical-mechanical polishing (CMP). The coordinate of the new surface must be specified as <code>coord</code> . See Polygonal Etching and CMP on page 218
<code>type=cmp.flat</code>	Similar to CMP, except the new surface specified by <code>coord</code> is flattened following stress relaxation and has zero displacement.
<code>type=crystal</code>	Performs angle-dependent etching where the etching rate depends on the crystallographic direction. See Crystallographic Etching on page 222
<code>type=directional</code>	Etches in one specific direction only. See Anisotropic and Directional Etching on page 214 .
<code>type=fourier</code>	Performs angle-dependent etching where the etching rate is a cosine expansion of the etching angle. See Fourier Etching on page 219 .
<code>type=isotropic</code>	Performs etching where the etching rate is uniform in all directions. See Isotropic Etching on page 213 .
<code>type=polygon</code>	Etches according to a user-supplied polygon (2D operations only). See Polygonal Etching and CMP on page 218 .
<code>type=trapezoidal</code>	Allows etching with undercut and taper angle specifications (2D operations only), or taper and bottom angle specifications (3D operations only). See Trapezoidal Etching on page 222

Some etching types have an option that can be used instead of specifying `type`:

- `anisotropic` (instead of `type=anisotropic`)
- `cmp` (instead of `type=cmp`)
- `isotropic` (instead of `type=isotropic`)
- `trapezoidal` (instead of `type=trapezoidal`)

Each etching type requires the setting of parameters particular to that type. Many options are available and some options are available only with certain etching types. Table 19 summarizes the syntax options for each etching type.

Table 19 Options for etch command syntax

Area	Parameter name	angles.rates	anisotropic	cmp, cmp.flat	crystal	directional	fourier	isotropic	polygon	trapezoidal
Rate	rate		*			*		*		*
	angles.rates	*								
	coeffs						*			
	mat.coeffs						*			
	crystal.rate				*					
Stop criteria	time	*	*		*	*	*	*		*
	thickness		*			*		*		*
	etchstop	*	*	*	*		*	*		
	coord			*						
	etchstop.overetch	*	*	*	*		*	*		
	isotropic.overetch		*			*				
Shape	polygon								*	
	angle									*
	undercut									*a
	bottom.angle									*b
	bottom.thickness									*b
	direction					*				
	ambient.rate									*c
Beam	sources	*					*			
	shadowing	*	*c			*	*			
	shadowing.nonisotropic						*			
Mesh	remesh	*	*	*	*	*	*	*	*	
	Adaptive	*	*	*	*	*	*	*	*	
Mode	1D								*	
	force.full.levelset		*		*d	*	*d	*		*

- a. In 2D when not using `force.full.levelset`.
- b. In 3D when not using `force.full.levelset`.
- c. When used in conjunction with `force.full.levelset`.
- d. Full level set is the default scheme for crystallographic and Fourier etching.

12: Structure Generation

Etching

NOTE To remove materials exposed to the top gas, use the `strip` command not the `etch` command. The `strip` command is used specifically for this purpose. It is more straightforward, less prone to user error, and more robust in delivering the expected results. For example:

```
strip Photoresist
```

Etching types have been implemented in Sentaurus Interconnect using three different methods (analytic, fast level-set, and general time-stepping level-set), which are described in [MGOALS on page 264](#).

These methods may require different inputs to perform the steps and may take different effects into account. The method is selected depending on the specified parameters and the structure to be etched.

NOTE The simplest and fastest algorithm possible is chosen by default.

If simple isotropic, anisotropic, directional, or CMP etching of a single material is requested and for polygonal etching, an analytic method is used. The analytic method is the fastest and most accurate. However, in some cases, the resulting etching front might intersect itself. Sentaurus Interconnect detects such situations and switches to the fast level-set method.

NOTE Although analytic methods are fast and can handle most simple etching tasks, they do not consider shadowing or visibility effects, and they cannot etch more than one material at a time.

The general time-stepping level-set method is chosen if you specify any rate versus angle-type etching (Fourier or crystallographic), or if you choose to etch different materials at different rates, or if the option `force.full.levelset` is specified. In addition, the general time-stepping level-set method can handle multiple etching beams and, optionally, shadowing.

The general time-stepping level-set method used in Sentaurus Interconnect has the same limitations as all level-set methods:

- Sharp corners in the evolving front are rounded.
- Small front movement requires a fine level-set mesh, resulting in large memory use and long simulation times.
- The accuracy is limited by the size of the level-set mesh.

Besides the etching type, the materials to be etched and the amount of material to be removed must be specified. The amount of material to be etched can be specified in three ways:

- Thickness
- Rate and time
- Using an etch stop

The etching rate may be specified using etching beams that are created with the `beam` command. Beams can be used only with Fourier etching. If an etch stop is specified, the etching stops as soon as the specified material is exposed to gas.

In addition, a mask specification can be given for all etching types, except CMP and polygonal to limit the areas where material is removed.

NOTE Trapezoidal etching supports mask specification in 3D only.

Isotropic Etching

Isotropic etching removes material at the same rate in all directions.

When isotropic etching uses either the fast level-set method or the general time-stepping level-set method, the final surface is obtained by solving a differential equation on a discrete mesh.

NOTE To control errors in the fast level-set method, use `resolution` in the `mgoals` command. For the general etch method, use either `resolution` or the `dx` and `dy` arguments of the `mgoals` command.

An example of a single-material isotropic etch is (see [Figure 37](#)):

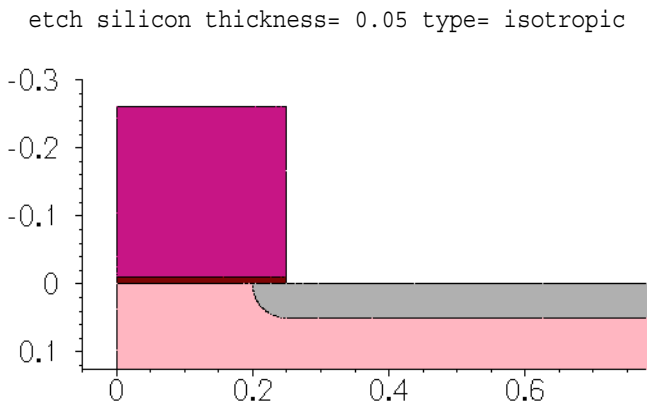


Figure 37 Single-material isotropic etching

12: Structure Generation

Etching

An example of a multimaterial isotropic etch is (see [Figure 38](#)):

```
etch material= {Silicon Oxide Poly} rate= {1.0 1.5 1.0} time= 0.05 \  
type= isotropic
```

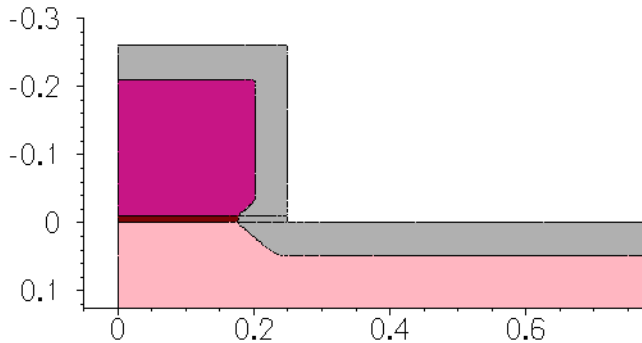


Figure 38 Multimaterial isotropic etching

Anisotropic and Directional Etching

Anisotropic etching is designed to work primarily with masks or masking layers. Anisotropic etching removes material in a direction that is purely vertically downwards (see [Figure 39](#)).

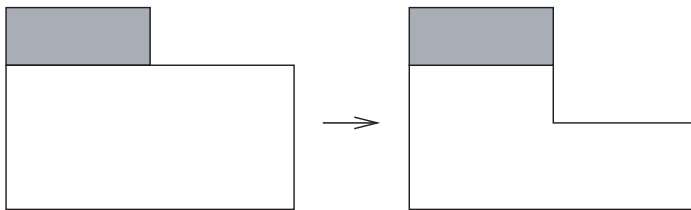


Figure 39 Anisotropic etching

Anisotropic etching can take more than one material if the same rate is specified for all etched materials. This can be useful to create multiple spacers since it does not produce small gaps, which are difficult to avoid when etching one spacer at a time. If you want to use different rates for each material, a better alternative is Fourier etching.

If anisotropic etching is performed to etch the shaded region in the structure shown in [Figure 40](#), instabilities can arise. The resulting structure can be very different depending on the numeric round-off errors.

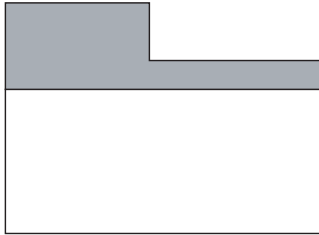


Figure 40 For this structure, anisotropic etching would not be stable

NOTE With regard to anisotropic etching:

- Anisotropic operations are sensitive to numeric noise at vertical or nearly vertical walls.
- If the `etch` command is supposed to remove the entire layer, care must be taken to overetch by a small amount to prevent thin regions remaining due to numeric round-off errors.
- It is more robust and better to use the `strip` command to remove all exposed layers of a certain material.

In two dimensions, when anisotropic etching is performed, portions of the etched material lying underneath other material are protected by those materials and, therefore, no etching occurs in those areas. For example, in [Figure 41 on page 216](#), when silicon is etched using anisotropic etching, the portion lying underneath oxide is not etched.

On the other hand, depending on the etching rate, in three dimensions, the silicon area underneath oxide can be etched away, as shown in [Figure 42 on page 216](#). To make the behavior consistent with 2D anisotropic etching, you can specify the `aniso.etching.protect.materials` option of the `mgoals` command:

```
mgoals aniso.etching.protect.materials
```

12: Structure Generation
Etching

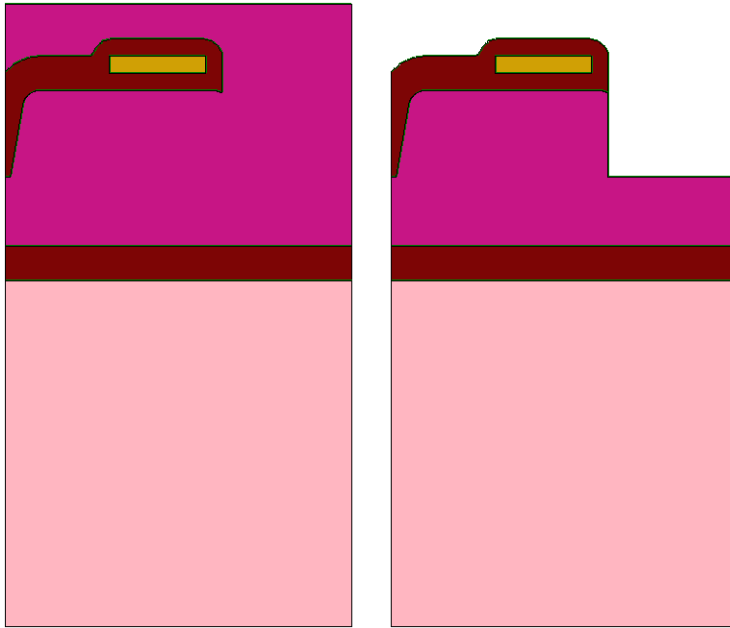


Figure 41 Two-dimensional anisotropic etching with material providing protection:
(left) original structure and (right) result after etching

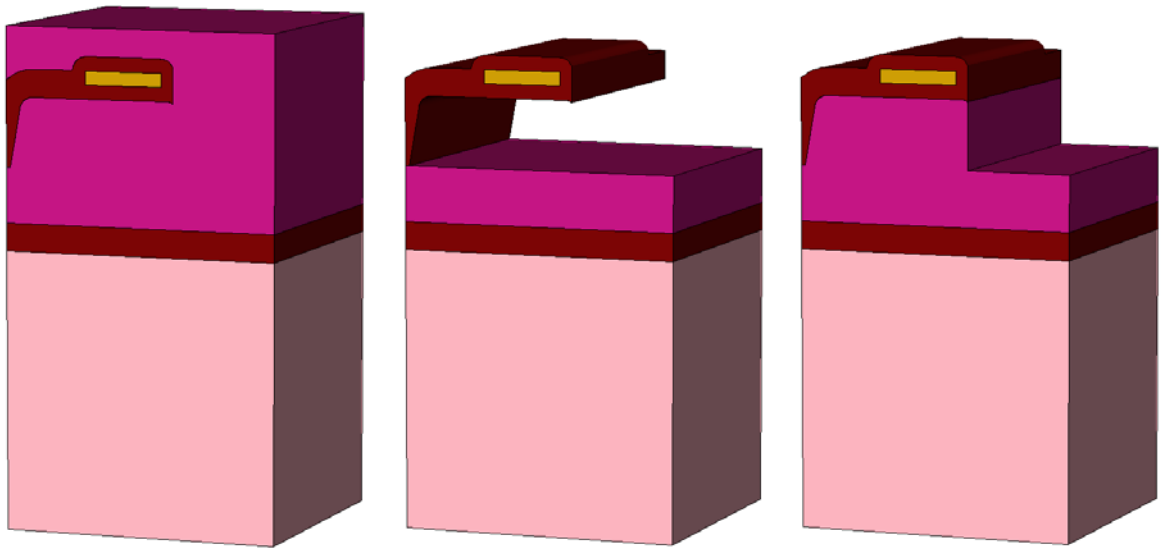


Figure 42 Three-dimensional anisotropic etching with material providing protection:
(left) original structure, (middle) default behavior leaving a floating region, and
(right) behavior with aniso.etching.protect.materials option

Directional etching is similar to anisotropic etching. In this case, the specified etching rate is applied in the direction of the etching beam. Visibility effects are not considered. The etching window is determined from user-defined masks and from the exposed areas of the etched material (see [Figure 43](#)). For example:

```
etch silicon thickness= 0.05 type= anisotropic
```

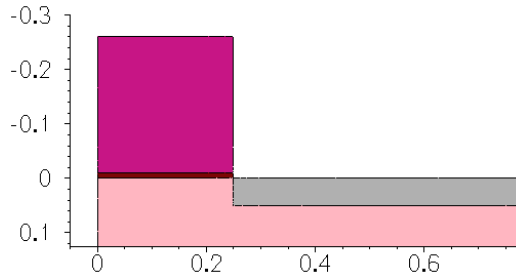


Figure 43 Anisotropic etching

In the next example of directional etching, the argument `direction` specifies the direction of the etching beam by setting values for $\{x\ y\ z\}$. This direction vector is normalized to 1.0 before the `etch` command is executed.

```
etch material= silicon rate= 0.05 time= 1.0 type= directional direction= {1 1}
```

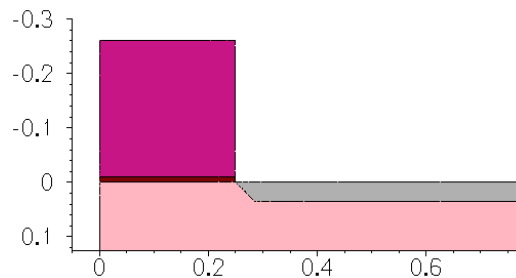


Figure 44 Directional etching (angled wall)

```
etch material= silicon rate= 0.05 time= 1.0 type= directional direction= {1 -1}
```

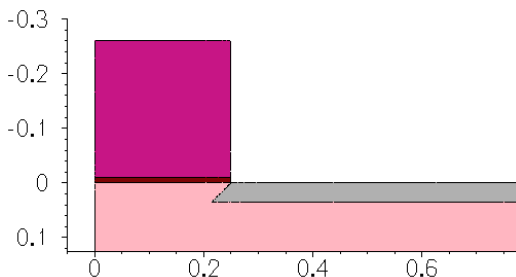


Figure 45 Directional etching with undercut

Polygonal Etching and CMP

Polygonal etching provides a way of modifying a region without having to define etching rates or the direction of the etch. The parts of the region inside the polygon that is etched are replaced by gas.

CMP is handled as a special case of polygonal etching. Mesh elements are intersected at the specified coordinate. All elements of the specified material above the coordinate are reassigned to gas.

This example etches a polygon given as $x_1 y_1 x_2 y_2 \dots x_n y_n$:

```
etch type= polygon material= silicon \  
  polygon= {-0.1 0.1 0.05 0.1 0.05 0.6 -0.1 0.6}
```

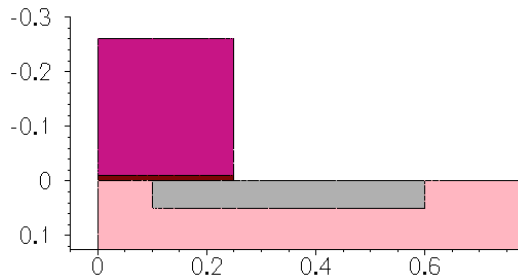


Figure 46 Polygonal etching

For example:

```
etch type= cmp coord= 0.05 material= all
```

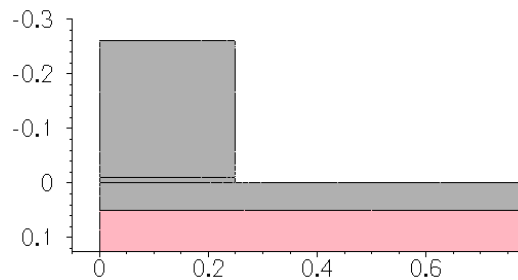


Figure 47 CMP

Fourier Etching

In Fourier etching, the etching rate is a function of the angle between the incident etching beam and the normal vector of the surface being etched. This allows for reasonably directional etching with control of the slope of sidewalls. The coefficients A_n are defined using `coeffs` (for a single-material etch) or `mat.coeffs` (for a multimaterial etch), and the etching rate is computed according to:

$$\text{etch rate} = \sum_{i=0}^m \text{factor}_i \sum_{j=0}^n A_j \cos^j \theta_i \quad (111)$$

where:

- θ_i is the angle between the incident beam i and the normal to the surface being etched.
- factor_i is the `factor` given in the beam command for beam i .

Any number of coefficients A_j can be given for each material. If the parameters A_j are chosen such that negative etching rates would result in some slope angles, no etching will occur on the parts of the surface that have that slope.

It is common to set the parameters A_j such that the etching rate for angles less than a certain angle are positive and drop below zero (resulting in no etching) above that angle. This produces a trench with a rounded bottom and a sidewall given by the angle where the etching rate drops to zero.

Fourier etching uses the full level-set method formulated after Lax–Friedrichs. This formulation shows good stability, leading to good accuracy of etch wall-angle control. The Lax–Friedrichs formulation results in slightly less corner sharpness.

The `beam` command defines the direction and the relative strength of etching beams to be used with Fourier etching. The syntax is:

```
beam name=<c> (direction= {<x> <y> <z>} | incidence=<n>) factor=<n>
```

You specify the angle of incidence of each beam using either `direction` (the specified direction vector is normalized automatically to unit length) or `incidence` (`incidence=0` defines a vertical beam).

The angle θ_i in [Eq. 111](#) is measured from the surface perpendicular to the angle of incidence for beam i . The relative strength `factor` is the weight (strength) of each beam. Each etching beam must be given a unique name.

Etching beams are assumed to be collimated, that is, a slight angular spread of beam direction is not considered.

12: Structure Generation

Etching

The `sources` argument of the `etch` command lists the names of etching beams to be used in an etching operation.

For example (see [Figure 48](#)):

```
beam name= src1 direction= {1 0 0} factor= 1
etch material= silicon type= fourier sources= src1 coeffs= {0 0 1.0} time= 0.05
```

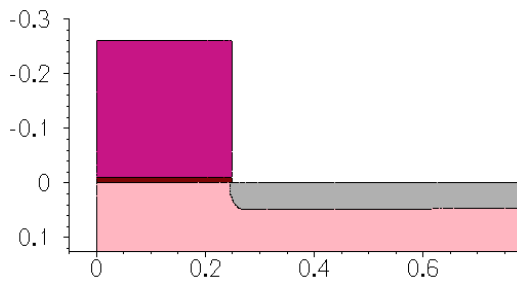


Figure 48 Fourier etching

Through the selection of Fourier coefficients, the angle of the etching wall can be controlled to a large degree. In particular, the first coefficient in the list, A_0 , corresponds to the equivalent of the rate of isotropic etching. The second coefficient in the list, A_1 , corresponds approximately to the equivalent of the rate of anisotropic or directional etching. The approximate formula for determining the etch wall angle is given as:

$$\varphi \approx \cos^{-1} \frac{-A_0}{A_1} \quad (112)$$

where φ is the angle of the etch wall measured from the horizontal plane. For example, $A_0 = -0.5$ and $A_1 = 0.7071$ result in an etch wall at an angle of approximately 45° from the horizontal as shown in [Figure 49 on page 220](#):

```
beam name= src1 direction= {0.1 0 0} factor= 1
etch material= {silicon} type= fourier sources= {src1} coeffs= {-0.5 0.7071} \
time= 1.0
```

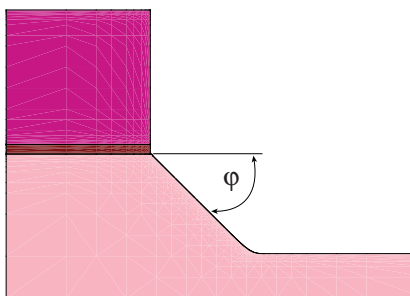


Figure 49 Fourier etching with etch wall angle of 45°

Another Fourier etching example shows the functionality in three dimensions and how multiple rates for multiple materials are specified using `mat.coeffs`. For 3D Fourier etching, also use the command `sde off`.

Etching coefficients chosen for this example are illustrative and might not be physically meaningful:

```
beam name= src1 direction= {1 0 0} factor= 1
etch sources= {src1} type= fourier remesh= false time= 1 \
  mat.coeffs= { Silicon= {-1 2} Nitride= {-0.7 1.2} Oxide= {0.01} \
  PolySilicon= {-0.05 0.2} }
```

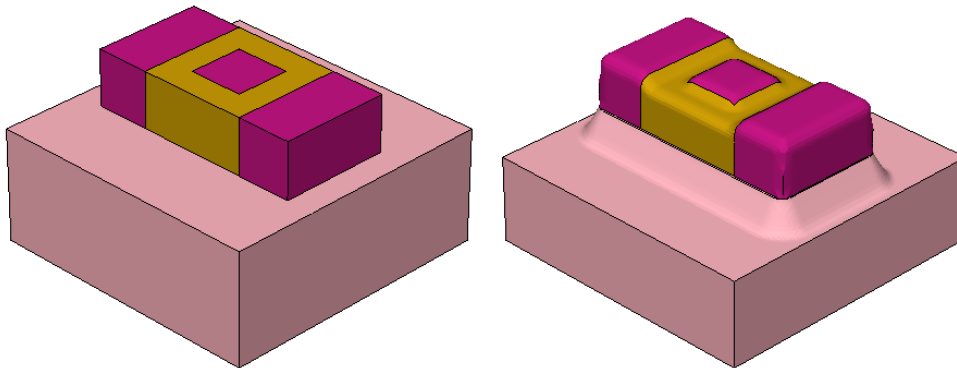


Figure 50 Three-dimensional multimaterial Fourier etching: (*left*) before etching and (*right*) after etching

The shadowing and shadowing.nonisotropic Options

NOTE Shadowing is not implemented in 3D Fourier etching.

The `shadowing.nonisotropic` option is used instead of `shadowing` when you want to allow only the 0th-order Fourier coefficient to etch areas where the beam is shadowed.

The `shadowing` option prevents all Fourier etching in areas shadowed from the beam; while the `shadowing.nonisotropic` option prevents only the Fourier coefficients of order one and higher from etching in areas where the beam is shadowed.

Be aware that even when `shadowing` is specified, the 0th-order Fourier coefficient A_0 should continue to etch areas where the beam is shadowed. This permits pseudo-isotropic etching that is independent of shadowing, while at the same time, full Fourier etching occurs only in areas where the beam is not shadowed.

Crystallographic Etching

The argument `crystal.rate` defines etching rates for different crystallographic orientations to be used in crystallographic etching (see [Defining the Crystal Orientation on page 35](#)).

Crystallographic etching rates are specified in `crystal.rate` as a list of one or more Miller indices i , j , and k with corresponding etching rates. The currently supported indices $\langle ijk \rangle$ are $\langle 100 \rangle$, $\langle 110 \rangle$, $\langle 111 \rangle$, $\langle 311 \rangle$, $\langle 511 \rangle$, $\langle 911 \rangle$, and $\langle 221 \rangle$:

```
crystal.rate= {"<100>" =<etch_rate> "<110>" =<etch_rate> "<111>" =<etch_rate>}
```

NOTE You must add a space between the double quotation mark (") after the orientation and the equal sign.

Interpolation of the rate at a given point along the etch front is calculated as a linear combination of the $\langle ijk \rangle$ rates weighted by the component of the etch front normal vector along the corresponding crystallographic direction.

A crystallographic etching example is (see [Figure 51](#)):

```
etch material= silicon type= crystal \  
  crystal.rate= {"<100>" =1.0 "<110>" =0.5 "<111>" =0.001} time= 0.25
```

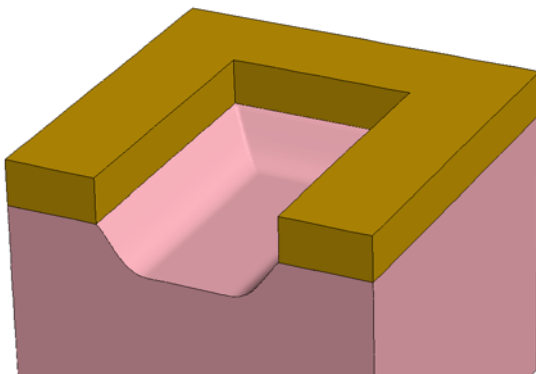


Figure 51 Crystallographic etching

Trapezoidal Etching

Trapezoidal etching provides a simple but flexible approximation to a number of real etching processes. The location of the etch is determined by masking layers (that is, layers of nonetchable material that, if nonexistent, can be easily created with the `photo` command). In three dimensions, the etch location can be specified through the definition of a mask.

Trapezoidal etching uses the following arguments to specify the shape of the region to be removed:

- `thickness` specifies the vertical depth (or a combination of `rate` and `time`).
- `angle` specifies the angle (in degrees) of the resulting sidewalls.
- `undercut` specifies the horizontal penetration of the etch under the edges of the masking layer. It only works in two dimensions.
- `bottom`, `angle` and `bottom.thickness` specify the angle and thickness of the sidewalls for a second etching after `thickness` and `angle` are already etched (3D operations only).

These arguments can be used to approximate a number of real etching processes including:

- Combinations of vertical and isotropic etches.
- V-groove etches.
- Etches that produce retrograde sidewall profiles.

Two-Dimensional Trapezoidal Etching

Trapezoidal etching is performed as follows:

1. Perform vertical etching to depth `thickness`. This etching does not apply to portions of the surface that are masked by nonetchable materials or shadowed by etchable or nonetchable materials. In addition, it is not used on segments of the surface that form an angle greater than `angle` to the horizontal.
2. Perform horizontal etching. Surfaces that were exposed at the start of Step 1 are etched horizontally by the distance `undercut`. Surfaces that were exposed during Step 1 are etched by a distance proportional to the length of time between when they first became exposed and the end of Step 1. Therefore, a sidewall exposed three-fourths of the way into Step 1 is etched horizontally by one-fourth of `undercut`. (An exception is made when an `angle` greater than 90° is specified.)
3. Perform a vertical upwards etch (that is, in the direction) where overhangs of etchable material are present at the end of Step 2. On surfaces that were exposed at the start of Step 2, this etching is to a distance `undercut`. On surfaces that were first exposed during Step 2, the distance of this etching is reduced in proportion to the time from the start of Step 2. This step approximates the undercutting of the mask due to the isotropic component of the etching.

NOTE Trapezoidal etching using a mask definition is not supported in 2D operations.

When the `thickness`, `angle`, and `undercut` arguments satisfy the following relationship, the etching approximates a vertical etching with an isotropic component:

$$\text{thickness} = \text{undercut} \cdot \tan(\text{angle}) \quad (113)$$

12: Structure Generation

Etching

This is the case whenever two or fewer of these arguments are specified with the `Trapezoidal.Etch.Undercut` parameter switched on (default is 0):

```
pdbSet Grid Trapezoidal.Etch.Undercut 1
```

For example:

```
etch material= silicon type= trapezoidal thickness= 0.25 undercut= 0.1
```

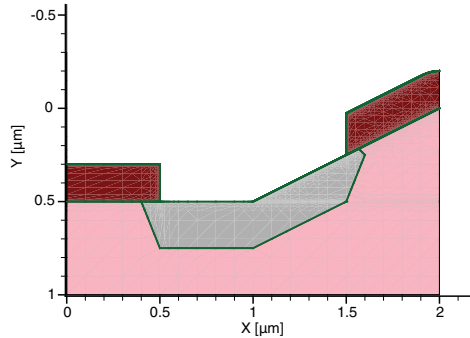


Figure 52 Basic trapezoidal etching example where the surface is nonplanar

The left half of [Figure 52](#) shows the result when etching a planar substrate. The etch region is a trapezoid of depth `thickness`, extending a distance `undercut` beneath the mask edge. The right half of [Figure 52](#) shows the result when etching a nonplanar surface.

Step 1 of the sequence etches the exposed surface vertically to a depth of `thickness` micrometers. Step 2 etches the resulting sidewall in the horizontal direction, producing an undercutting of the mask and the sloped sidewall. In this case, Step 3 also has an effect, etching upwards from the undercut region. Therefore, the hook in the final silicon profile is the result of approximating the isotropic component of the etch. In each case, the intersection between the bottom of the etched region and the sidewall occurs directly under the edge of the mask.

[Figure 53](#) and [Figure 54](#) on page 225 show what happens when [Eq. 113](#) is not satisfied.

```
etch material= silicon type= trapezoidal thickness= 0.3 undercut= 0.1 angle= 45
```

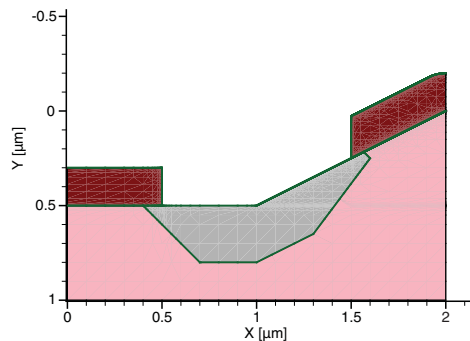


Figure 53 Trapezoidal etching example generating a narrow trench bottom

```
etch material= silicon type= trapezoidal thickness= 0.3 undercut= 0.1 \
angle= 135
```

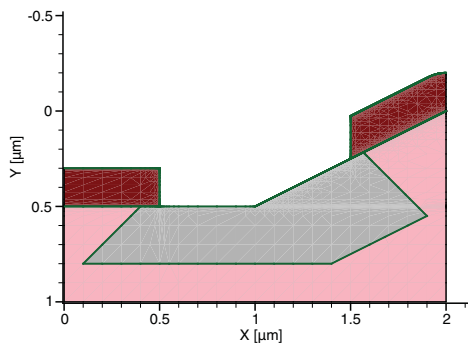


Figure 54 Trapezoidal etching example generating a wide trench bottom

In [Figure 53 on page 224](#), $\text{thickness}/\text{undercut}$ are less than $\tan(\text{angle})$. In this case, the sloped sidewall of the etch extends out under the opening in the mask. The intersection between the bottom of the etch region and the sidewall is no longer directly beneath the edge of the mask. If the mask opening is narrow enough, the bottom of the etch region disappears entirely, resulting in a V-groove etch. To produce this etch shape, Step 1 of the etch process is modified to reduce the depth of the vertical etch near the edges of the mask opening. Note that, in this situation, even the smallest amount of nonetchable material can produce a triangular mound of unetched material in the final structure.

[Figure 54](#) shows the case with an angle greater than 90° . In this case, the bottom of the etched region is wider than the opening in the masking layer, producing overhanging sidewalls. This etch is accomplished by modifying Step 2 of the procedure to etch further horizontally at the bottom of the sidewalls formed by Step 1 than at the top. The apparent etch depth of $0.5 \mu\text{m}$ at the right side of the mask opening is the result of a $0.3 \mu\text{m}$ vertical etch of the original sloped surface (Step 1) followed by a $0.4 \mu\text{m}$ horizontal etch of the sloped ‘bottom wall’ that results from Step 1.

Three-Dimensional Trapezoidal Etching

There are two possible cases for 3D trapezoidal etching:

- Case 1: A thickness (or rate and time) and an angle (optional) are specified. If angle is not specified, it defaults to 90° (vertical). In contrast with case 2, angle allows any value between 0° and 180° . [Figure 55 on page 226](#) shows examples of both cases.
- Case 2: A thickness (or rate and time), an angle (optional), and bottom.thickness and bottom.angle (optional) are specified. If angle is not specified, it defaults to 90° (vertical). Here, a special condition applies: angle must be greater than or equal to 90° , while bottom.angle must be smaller than or equal to 90° . That is, the *first etching* penetrates behind the mask, while the second one does the opposite. [Figure 56 on page 226](#) shows an example of this type.

12: Structure Generation

Etching

NOTE For 3D trapezoidal etching to succeed, the initial etching surface must be flat.

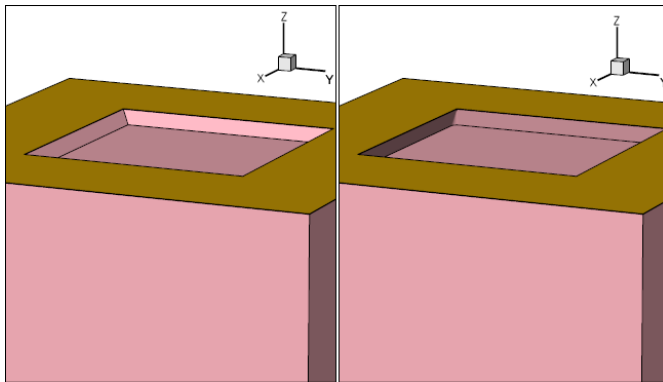


Figure 55 Three-dimensional trapezoidal etching: (*left*) angle=45 thickness=0.3 and (*right*) angle=110 thickness=0.44

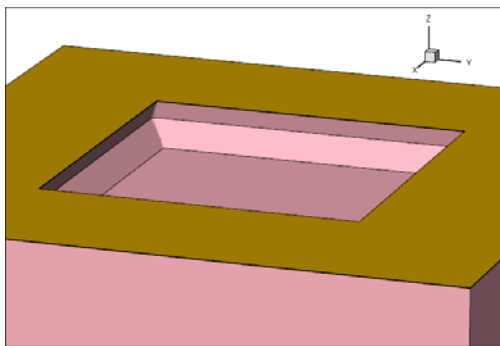


Figure 56 Three-dimensional trapezoidal etching produced with angle=110, thickness=0.3, bottom.angle=45, and bottom.thickness=0.400

Trapezoidal Etching Using force.full.levelset Option

When the `force.full.levelset` option is specified for trapezoidal etching:

- The etching depth is controlled by `rate` and `time`, not by `thickness`.
- The argument `ambient.rate` is used to approximate the underetch effect otherwise available in the non-`force.full.levelset` case controlled in two dimensions by `undercut`. If `undercut` is specified, `ambient.rate` will be approximated by `undercut/time`, and a warning message will be issued. The `ambient.rate` argument also approximates underetching, controlled in three dimensions in the non-`force.full.levelset` case by the combination of `angle`, `bottom.angle`, and `bottom.thickness`.

- The options supported by the level-set solver such as `shadowing` become available.
- An additional argument `roundness` (default 1.0) can be used to increase the curvature of etching sidewalls.

Piecewise Linear Etching

In piecewise linear etching, the etching rate is a user-defined piecewise linear function of the angle between the incident etching beam and the normal vector of the surface being etched. You define the points of angle versus rate on a material-by-material basis, as per the following syntax:

```
angles.rates= {
  matA = {angleA0 rateA0 angleA1 rateA1 ... angleAn rateAn}
  matB = {angleB0 rateB0 angleB1 rateB1 ... angleBn rateBn}
  ...
}
```

where $\text{angle}_{n-1} \leq \text{angle}_n$.

The rates and angles are interpreted as follows:

- Angles are given in degrees in the range $[0^\circ, 180^\circ]$.
- The rate for $\text{angle} < \text{angle}_0$ is rate_0 .
- The rate for $\text{angle} > \text{angle}_n$ is 0.
- The rate is calculated as the linear interpolation of the nearest two angle–rate pairs within which the angle lies.

While Fourier etching and trapezoidal etching also define the etching rate according to the angle between the beam direction and the surface normal, a piecewise linear function is a more general parameterization of etching rate versus angle that users control directly.

For example (see [Figure 57 on page 228](#)):

```
beam name= src1 direction= {1 0 0} factor= 1
etch type= angles.rates sources= src1 time= 0.2 \
  angles.rates= { Silicon= { 25 1.0 45 0.3 } }
```

12: Structure Generation

Etching

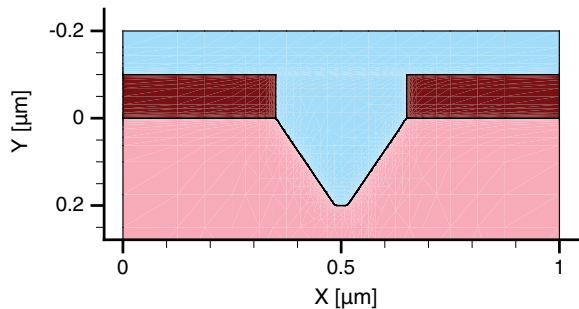


Figure 57 Piecewise linear etching

You must define the piecewise linear function as smoothly as possible, avoiding discontinuous changes, to ensure well-defined level-set results.

Etching Beams

The `beam` command defines the direction and the relative strength of etching beams to be used with piecewise linear etching (see [Fourier Etching on page 219](#)).

The argument `sources` of the `etch` command lists the names of etching beams to be used in an etching operation.

Etching Tips

Some tips for etching are:

- If the total etch thickness is exactly the same thickness as the layer to be etched, numeric round-off errors can cause thin pieces of the material to be retained. You should etch a little more (for example, 0.1%) than the thickness of the layer.
- Etching small thicknesses using a small `isotropic.overetch` or etching large structures can cause the simulation to allocate a large amount of memory and to increase the simulation time to solve the level-set equation. The argument `resolution` of the `mgoals` command can be increased for the simplified boundary movement mode, and the arguments `dx` and `dy` can be increased in the general boundary movement mode to reduce memory consumption. However, this may affect accuracy.

Deposition

Three main specifications are required for all deposition steps:

- Deposition type
- Material to be deposited
- Amount of material to be deposited

The supported deposition types are specified using the `type` argument of the `deposit` command.

Table 20 Supported deposition types

Deposition type	Description
<code>type=anisotropic</code>	Performs anisotropic deposition.
<code>type=crystal</code>	Performs crystallographic deposition. The <code>crystal.rate</code> argument must be specified. See Crystallographic Deposition on page 233 .
<code>type=directional</code>	Performs anisotropic deposition using a specified direction.
<code>type=fill</code>	Performs a fill of the structure with the specified <code>material</code> up to the coordinate specified with the argument <code>coord</code> . See Polygonal Deposition and Fill on page 232 .
<code>type=fourier</code>	Performs Fourier deposition. See Fourier Deposition on page 233 .
<code>type=isotropic</code>	Performs isotropic deposition. See Isotropic Deposition on page 232 .
<code>type=polygon</code>	Performs a polygonal deposition that requires the <code>polygon</code> argument. The specified polygon is used to intersect all mesh elements of material gas. Then, elements inside the polygon are assigned to the specified material. (Not available in 3D operations.) See Polygonal Deposition and Fill on page 232 .
<code>type=trapezoidal</code>	Performs a trapezoidal deposition that requires the <code>angle</code> and <code>thickness</code> arguments (3D operations only). See Trapezoidal Deposition on page 234 .

Some deposition types have an option that can be used instead of specifying `type`:

- `anisotropic` (instead of `type=anisotropic`)
- `crystal` (instead of `type=crystal`)
- `fill` (instead of `type=fill`)
- `fourier` (instead of `type=fourier`)
- `isotropic` (instead of `type=isotropic`)

12: Structure Generation

Deposition

Each deposition type requires the setting of parameters particular to that type. Many options are available and some options are available only with certain deposition types. [Table 21](#) summarizes the syntax options for each deposition type.

Table 21 Options for deposit command syntax

Area	Parameter name	anisotropic	crystal	directional	fill	fourier	isotropic	polygon	trapezoidal
Rate	rate	*		*			*		
	coeffs					*			
	mat.coeffs					*			
	crystal.rate		*						
Stop criteria	time	*	*	*		*	*		
	thickness	*		*			*		*
	coord				*				
Shape	polygon							*	
	angle								*
	direction			*					
Beam	sources					*			
	shadowing	*		*		*			
	shadowing.nonisotropic					*			
Mesh	remesh	*	*	*		*	*	*	
	Adaptive	*	*	*		*	*	*	
Mode	1D							*	
	force.full.levelset		*a	*		*a	*		

a. Full level set is the default scheme for crystallographic and Fourier deposition.

To specify the thickness of the deposited layer for anisotropic and isotropic deposition, use either `thickness` or `rate` and `time`. Besides the deposition type and thickness, you must specify the material to be deposited (only one material is allowed per `deposit` command). To do this, either specify the material name (`<material>`) or `material=<c>` in the command.

The number of steps for a deposition is specified using `steps=<n>`. The specified `time` or `thickness` is subdivided accordingly. Subdividing a deposition into several steps might be useful if stresses are initialized in the deposited layer. A stressed film of a given thickness can be deposited at the same time or in several steps. Sentaurus Interconnect simulates stress rebalancing after each deposition step.

Multistep deposition is known to generate more realistic stress profiles compared to depositing the entire layer and then performing one stress rebalancing calculation.

By default, the material is deposited on the surface exposed to the upper gas region. If a structure has buried gas bubbles, they remain untouched. To deposit inside these gas bubbles:

- Specify the `fill.buried` option in the `deposit` command to fill all gas bubbles with the deposited material.
- Specify the `fill.buried.match.surrounding` option in the `deposit` command to fill all gas bubbles with the material surrounding them, while leaving bubbles at a material interface filled with gas. This option will override the behavior of `fill.buried` if it also is specified.
- Specify `fill.buried.interface=<material>` in the `deposit` command to fill all gas bubbles at a material interface with the specified material, *but only* if the material is already present in the simulation (for example, silicon).

The name of a mask also can be specified in the `deposit` command. In this case, the material is deposited outside the specified mask. Deposition inside a mask requires the mask to be inverted by specifying `negative` in the `mask` command defining the mask (see [Photoresist Masks on page 254](#)).

For deposition, the analytic method, the fast level-set method, and the full level-set method are available.

In two dimensions, the analytic method is preferred for performing deposition, and the level-set method is used when the analytic method is not possible because a front collision is detected.

In three dimensions, the analytic method is used for anisotropic deposition, the fast level-set method is used for isotropic deposition, and the full level-set method is used for Fourier deposition.

In the newly deposited region, constant field values can be initialized. For isotropic deposition, you can define piecewise linear solution fields as a function of the distance from the original surface.

Isotropic Deposition

For simple, conforming, isotropic deposition, the boundary is offset an equal distance in all directions. For example (see [Figure 58](#)):

```
deposit nitride thickness= 0.05 type= isotropic
```

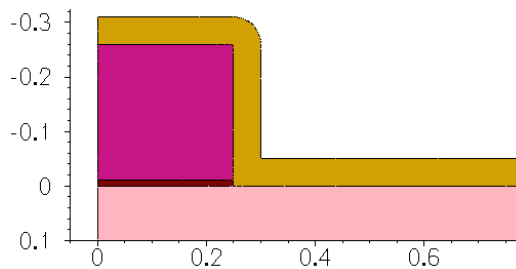


Figure 58 Isotropic deposition

Polygonal Deposition and Fill

Polygonal deposition allows you to deposit polygonal shapes. The corners must be specified as pairs of xy coordinates. Existing non-gas regions will not be overwritten.

NOTE Polygonal deposition is not available in 3D simulations.

For example (see [Figure 59](#)):

```
deposit type= polygon material= nitride polygon= {0.1 0.1 0.1 0.6 -0.31 \  
0.6 -0.31 0.1}
```

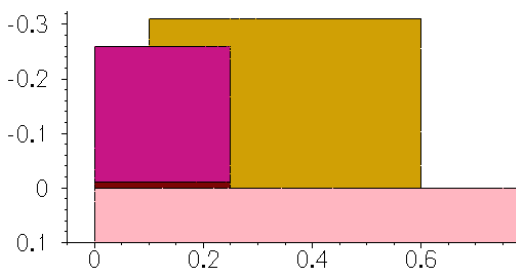


Figure 59 Polygonal deposition

Fill is a special case of polygonal deposition. Only the topmost coordinate must be specified. The simulation domain will be filled up to that coordinate with the specified material.

For example (see [Figure 60](#)):

```
deposit material= nitride type= fill coord= -0.31
```

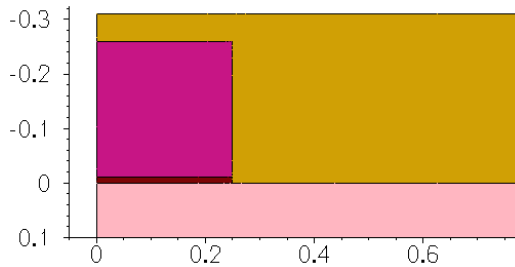


Figure 60 Fill deposition

Crystallographic Deposition

Crystallographic deposition uses the full level-set method to grow single materials whose rate of growth is determined by the crystallographic directions. The crystal is assumed to be cubic regardless of the material being deposited. Deposition rates can be set for one or more of the directions $\langle 100 \rangle$, $\langle 110 \rangle$, $\langle 111 \rangle$, $\langle 311 \rangle$, $\langle 511 \rangle$, $\langle 911 \rangle$, and $\langle 221 \rangle$. These rates will be applied to their respective equivalent directions based on cubic symmetry, for example, the $\langle 100 \rangle$ rate will apply to the $\langle 010 \rangle$, $\langle 001 \rangle$, $\langle \bar{1}00 \rangle$, $\langle 0\bar{1}0 \rangle$, and $\langle 00\bar{1} \rangle$ directions.

For example (see [Figure 61](#)):

```
deposit type= crystal material= nitride time= 0.05 \  
  crystal.rate = {"<100>" =1.0 "<110>" =0.1 "<111>" =0.05}
```

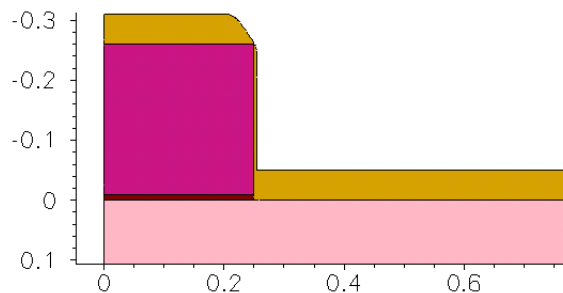


Figure 61 Crystallographic deposition: vertical direction is $\langle 100 \rangle$, lateral direction is $\langle 011 \rangle$

Fourier Deposition

Fourier deposition uses the full level-set method to grow single materials whose rate of growth is defined by a function of the angle between the surface normal of the material boundary and

12: Structure Generation

Deposition

the deposition beams. The definition and calculation of the deposit shape are exactly analogous to Fourier etching (see [Fourier Etching on page 219](#)).

For Fourier deposition, first, define the deposition beam in the same way as in Fourier etching:

```
beam name=<c> (direction= {<x> <y> <z>} | incidence=<n>) factor=<n>
```

Use the `sources` and `coeffs` arguments in the same way as in Fourier etching, for example:

```
beam name= src1 direction= {1 0 0} factor= 1  
deposit nitride time= 0.2 fourier sources= {src1} coeffs= { -0.3 0.7 }
```

The `coeffs` argument is given in units of $\mu\text{m}/\text{minute}$. By default, `time` is given in minutes.

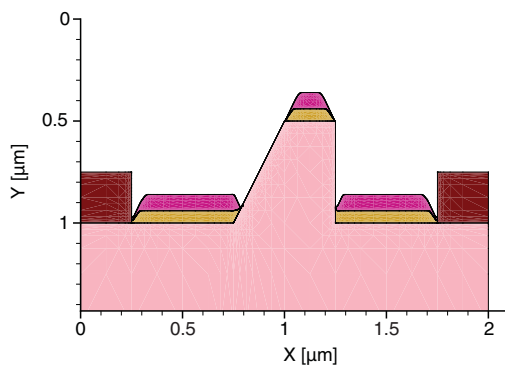


Figure 62 Two-dimensional Fourier deposition for demonstration purposes, nitride is deposited selectively on silicon using Fourier deposition and, then, an additional Fourier deposition step adding polysilicon selectively on nitride is performed

Trapezoidal Deposition

Trapezoidal deposition creates a shape with sidewalls of a defined height and angle. Height is defined by `thickness`. Angle is defined by `angle`, measured in degrees from the horizontal, where:

- `angle = 90°` is vertical.
- `angle > 90°` spreads outward as height increases.
- `angle < 90°` closes inward with increasing height.

NOTE Trapezoidal deposition is available only in three dimensions.

Separate outward and inward deposition steps can be used to create diamond-like shapes. Trapezoidal deposition requires a flat starting surface, for which `type=fill` can be used.

The following example demonstrates the syntax, whose results are shown in [Figure 63](#):

```
# First step deposition
deposit material= PolySilicon type= trapezoidal \
  selective.materials= Silicon thickness= 0.25 angle= 120

# Provide flat surface for second step
deposit type= fill coord= -0.25 Resist

# Second step deposition
deposit material= PolySilicon type= trapezoidal \
  selective.materials= PolySilicon thickness= 0.25 angle= 60
```

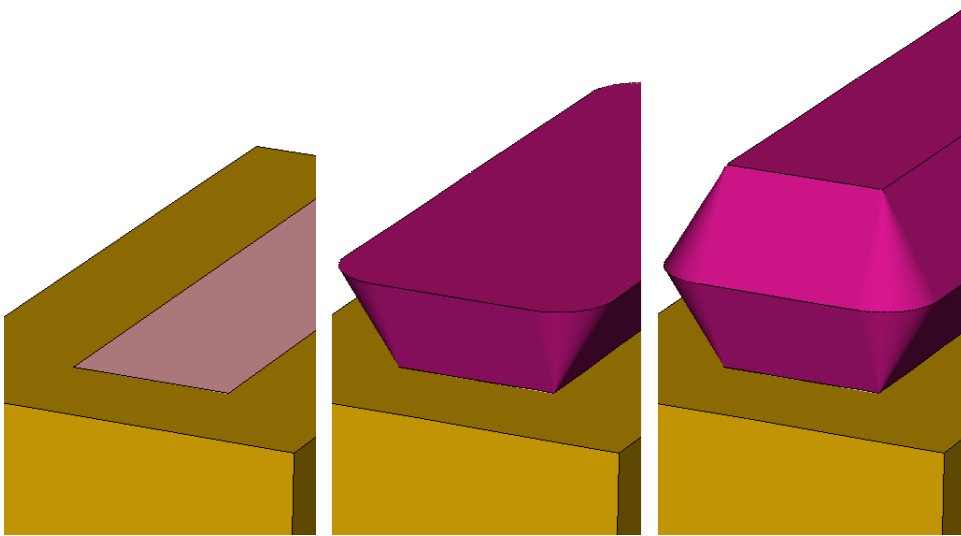


Figure 63 Trapezoidal deposition: (*left*) before deposition, (*middle*) first deposition step with angle=120, and (*right*) second deposition step with angle=60

Selective Deposition

Selective deposition is optionally available and can be used with isotropic, anisotropic, or Fourier deposition.

Using the `selective.materials` argument, you can select one or more materials to seed growth of the overlayer. In 3D SDE mode, only one material can be specified using `selective.materials`.

Fields in Deposited Layers

For isotropic deposition, piecewise linear fields can be specified in the deposited layer. For crystallographic deposition, piecewise linear fields with time also can be specified. A `doping` command must be used for each field; each `doping` command must assign a unique name.

For example, the following commands create a linear Ge field in the newly deposited oxide layer. Depth 0 corresponds to the initial surface (the bottom of the new layer):

```
doping name= strainGe field= Germanium depths= {0 0.1} values= {1e22 1e22}
deposit material= silicon doping= {strainGe} type= isotropic thickness= 0.1
```

The following commands create a linear boron field in the newly deposited silicon layer:

```
doping name=gradB field=Boron times= {0.0 0.015} values= {1e20 3e20}
deposit type=crystal doping= {gradB} material=Silicon time= 0.015 \
  crystal.rate= {<100>=1.0 <110>=0.5 <111>=0.9} selective.materials=Silicon
```

Constant field values can be defined for all deposition types as follows:

```
deposit material= silicon type= isotropic thickness= 0.1 \
  fields.values= {Vacancy=1e10 Germanium=2e22}
```

NOTE To create layers with intrinsic stress, use the field names `StressELXX`, `StressELXY`, `StressELYY`, `StressELZZ`, `StressELYZ`, and `StressELXZ`. It is not necessary to specify all components of stress. Those that are not specified are assumed to be initially at zero. If stresses are added in this way, they will be rebalanced after the deposition is completed. The actual value of the stress may differ from the value that was deposited.

Constant concentrations can be defined for known solution fields (known solution fields must have been defined before the `deposit` command, either in the `SINTERCONNECT.models` file or the command file of the user) as:

```
deposit material= {silicon} type= isotropic thickness= 0.1 Vacancy \
  concentration= 1e10
```

Handling Stress in Etching and Deposition

In addition to optionally including an automatic ramp-up or ramp-down before etching and deposition, by default, Sentaurus Interconnect automatically rebalances the stresses after etching and deposition. This updates the stress fields at the temperature of the etching or deposition step based on the new geometry.

For the best stress results, you must control the temperature history. This includes thermal ramp-up to process temperature, back to room temperature, and similarly ramp-up and ramp-down for both etching and deposition. However, as a minimum, the elastic stress rebalancing can be handled automatically by switching on stress history (see [Automated Tracing of Stress History on page 91](#)).

It is sometimes useful to switch off this stress rebalancing step in 3D simulations because the rebalance triggers a new mesh to be created. Therefore, if you are more concerned about simulation time than stress accuracy, you should specify the following to allow multiple etching and deposition steps to be performed without a mesh being generated in between:

```
pdbSet Mechanics EtchDepoRelax 0
```

Suppressing Mesh Generation in 3D Simulations

During typical operations in three dimensions, a new mesh is generated only when necessary to perform a task that requires a mesh. Operations such as etching and deposition are performed on the boundary representation and do not necessarily require a new mesh to be generated. However, mechanics steps such as stress rebalance and initializing fields in newly deposited layers do require a new mesh be generated.

By default, a mechanics step is performed after etching and deposition steps to balance the stresses after the structure has changed. The parameter `Mechanics EtchDepoRelax` controls this behavior. In addition, the parameter `Mechanics StressHistory` controls whether a stress rebalance step is added before `etch`, `deposit`, and `solve` commands. Either of these rebalance steps can trigger the generation of a new mesh.

If there are multiple etching and deposition steps in sequence, depending on the mechanics material models being used and the thermal history, it may be a good approximation to delay stress rebalancing until after the last step. This can be accomplished with the `suppress.remesh` option of the `etch` and `deposit` commands. Except for the case of deposition with field initialization, which requires a new mesh, this option will suppress mesh generation before and after etching and deposition steps.

NOTE When using `suppress.remesh`, be aware that stresses will not be rebalanced if the mesh is out-of-date because of a boundary change.

Shape Library

The shape library provides commands for generating special-shaped polyhedra in Sentaurus Interconnect. Sentaurus Structure Editor creates these shapes. The shape library is an interface to use those shapes in Sentaurus Interconnect.

There are two ways to use shapes from the shape library:

- In MGOALS mode, polyhedra are created using Sentaurus Structure Editor. The generated polyhedra then can be inserted into a Sentaurus Interconnect structure using the `insert` command.
- In SDE mode, polyhedra are not created directly. Instead, the Sentaurus Structure Editor structure itself is modified by inserting the shapes (replacing other materials). To activate the SDE mode, use the command `sde on`.

The commands available in the shape library are:

- `PolyHedronSTI` creates a shallow trench isolation (STI)-shaped polyhedron.
- `PolyHedronSTIacc` creates an STI concave active corner-shaped polyhedron.
- `PolyHedronSTIaccv` creates an STI convex active corner-shaped polyhedron.
- `PolyHedronCylinder` creates a cylinder-shaped polyhedron.
- `PolyHedronEllipticCylinder` creates an elliptic cylinder-shaped polyhedron.
- `PolygonSTI` creates a 2D STI-shaped polygon.
- `PolygonWaferMask` creates a wafer mask polygon.
- `PolyHedronEpiDiamond` creates an epitaxial diamond-shaped polyhedron.

You can define additional commands that create parameterized custom shapes using the scripting capabilities of Sentaurus Interconnect and Sentaurus Structure Editor.

PolyHedronSTI

The `PolyHedronSTI` command creates an STI-shaped polyhedron.

The syntax is:

```
PolyHedronSTI name direction X0 Y0 Depth Zmin Zmax Tsti Asti Hsti Rd Rb Ru  
[material]
```

where:

- name is the name of the polyhedron.
- direction can be left, right, front, or back, which sets the facing direction of the STI polyhedron.
- material is optional and specifies the material of the inserted shape in SDE mode. In MGOALS mode, the material of the inserted shape can be specified in the insert command.
- [Figure 64](#) describes the other parameters.

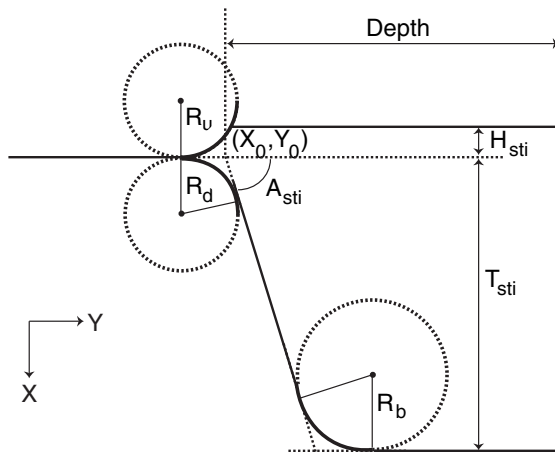


Figure 64 Parameters for generating STI-shaped polyhedron

Figure 65 shows some generated STI shapes in different directions.

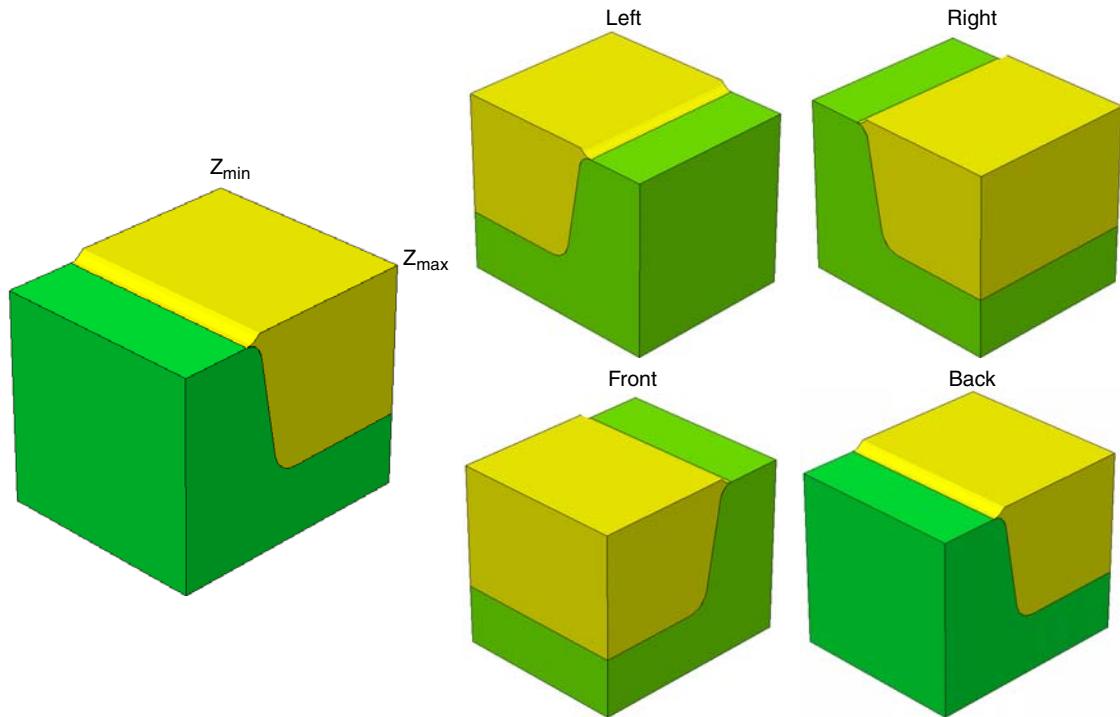


Figure 65 STI-shaped polyhedra in different directions

Figure 66 shows STI shapes with different T_{sti} and R_b values.

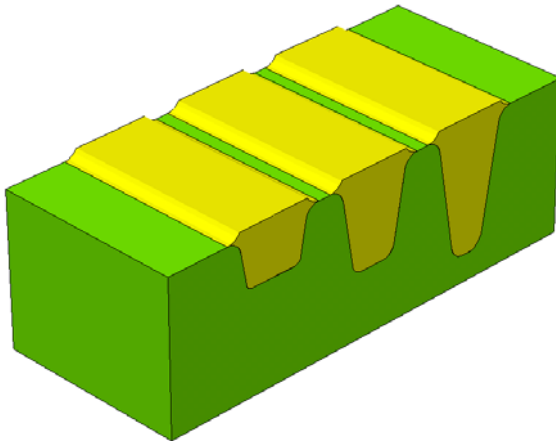


Figure 66 STI-shaped polyhedra with different T_{sti} and R_b

PolyHedronSTIaccc

The PolyHedronSTIaccc creates an STI concave active corner-shaped polyhedron.

The syntax is:

```
PolyHedronSTIaccc name direction X0 Y0 Z0 Tsti Asti Hsti Rd Rb Ru Rac  
[material]
```

where:

- name is the name of the polyhedron.
- direction can be rb (right back), lb (left back), lf (left front), or rf (right front).
- Rac is the radius of the STI concave corner.
- [Figure 64 on page 239](#) describes the other parameters.

[Figure 67](#) shows STI concave corner-shaped polyhedra in different directions.

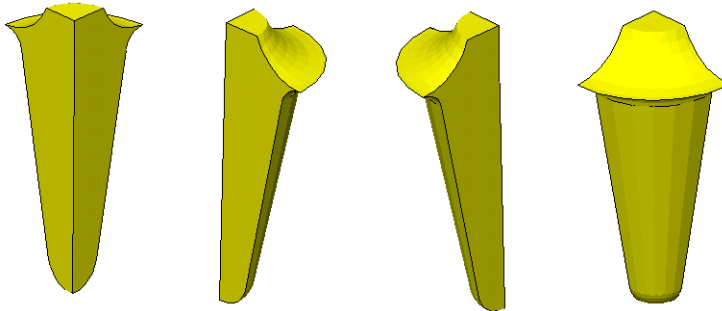


Figure 67 STI concave corner-shaped polyhedra in different directions: (*from left to right*) left back, right back, left front, and right front

PolyHedronSTIaccv

The PolyHedronSTIaccv command creates an STI convex active corner-shaped polyhedron.

The syntax is:

```
PolyHedronSTIaccv name direction X0 Y0 Z0 Depth Tsti Asti Hsti Rd Rb Ru Rac  
[material]
```

where:

- name is the name of the polyhedron.
- direction can be rb, lb, lf, or rf (as for the PolyHedronSTIaccc command).

12: Structure Generation

Shape Library

- R_{ac} is the radius of the convex corner.
- [Figure 64 on page 239](#) describes the other parameters.

[Figure 68](#) shows a generated STI convex corner-shaped polyhedron.

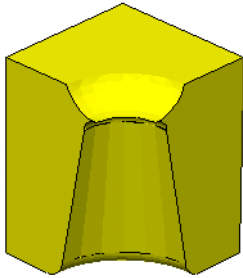


Figure 68 STI convex corner-shaped polyhedron

[Figure 69 \(left\)](#) shows a structure generated by combining the above three STI commands. [Figure 69 \(right\)](#) illustrates the directions of the STI shapes.

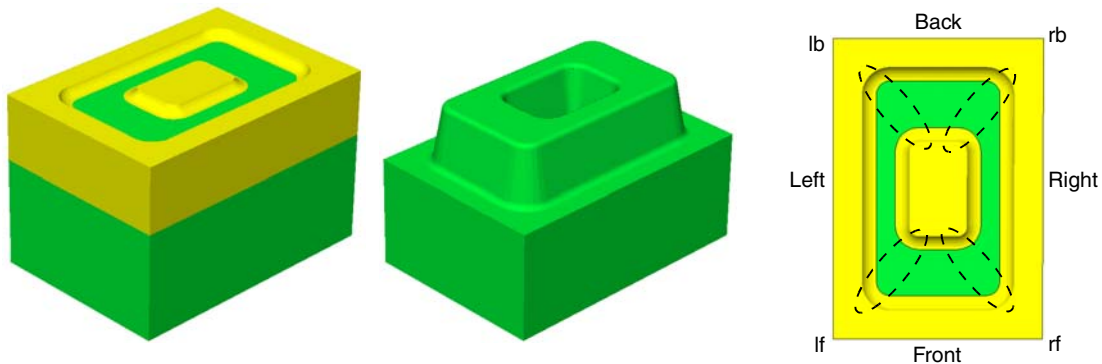


Figure 69 (Left) STI structures and (right) polyhedron directions

PolyHedronCylinder

The `PolyHedronCylinder` command creates a cylinder-shaped polyhedron.

The syntax is:

```
PolyHedronCylinder name X0 Y0 Z0 Rc Hc [material] [Rotate.Y | Rotate.Z] [angle]
```

where:

- `name` is the name of the polyhedron.

- If `Rotate.Y` is specified, the cylinder is rotated at position (X_0, Y_0, Z_0) along the y-axis for `angle` degrees. If `Rotate.Z` is specified, the cylinder is rotated along the z-axis. The right-hand rule determines the direction of the rotation.
- `angle` is the rotation angle in degree.
- Other parameters give the center coordination, the radius, and the height for the cylinder (see Figure 70).

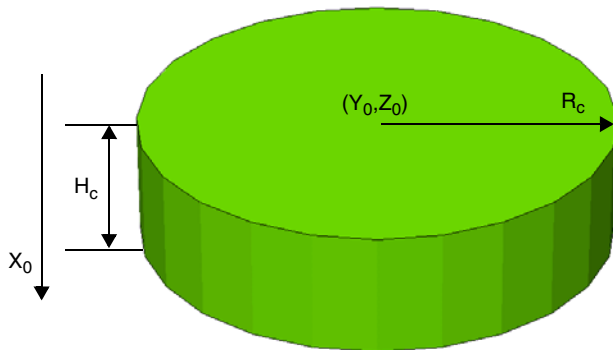


Figure 70 Cylinder-shaped polyhedron

PolyHedronEllipticCylinder

The `PolyHedronEllipticCylinder` command creates an elliptic cylinder-shaped polyhedron.

The syntax is:

```
PolyHedronEllipticCylinder name X0 Y0 Z0 Y1 Z1 Ratio Hc [material]
    [Rotate.X | Rotate.Y | Rotate.Z] [angle]
```

where:

- `name` is the name of the polyhedron.
- X_0, Y_0, Z_0 are the center coordinates of the base ellipse.
- Y_1 and Z_1 are the y- and z-coordinates of the major axis of the base ellipse. The base ellipse is on the yz plane.
- `Ratio` is the ratio of the minor axis to the major axis.
- `Hc` is the height for the cylinder. The cylinder is extruded in the $-x$ -direction.
- If `Rotate.X` is specified, the cylinder is rotated at position (X_0, Y_0, Z_0) along the x-axis for `angle` degrees.
If `Rotate.Y` is specified, the cylinder is rotated at position (X_0, Y_0, Z_0) along the y-axis for `angle` degrees.

12: Structure Generation

Shape Library

If `Rotate.Z` is specified, the cylinder is rotated along the z-axis. The right-hand rule determines the direction of the rotation.

- `angle` is the rotation angle in degrees (see [Figure 71](#)).

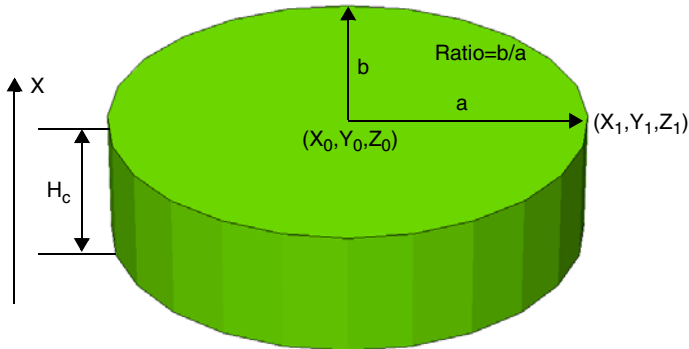


Figure 71 Elliptic cylinder-shaped polyhedron

PolygonSTI

The `PolygonSTI` command creates a 2D STI-shaped polygon. The syntax is the same as for `PolyHedronSTI`, except it does not have the `Zmin` and `Zmax` parameters (see [PolyHedronSTI on page 238](#)).

The syntax is:

```
PolygonSTI name direction X0 Y0 Depth Tsti Asti Hsti Rd Rb Ru [material]
```

where:

- `name` is the name of the polygon.
- `direction` can be `left` or `right` only. This sets the facing direction of the STI polygon.
- `material` is optional and specifies the material of the inserted shape in SDE mode. In MGOALS mode, the material of the inserted shape can be specified using the `insert` command.
- [Figure 64 on page 239](#) describes the other parameters.

PolygonWaferMask

The PolygonWaferMask command creates a wafer mask polygon.

The syntax is:

```
PolygonWaferMask name Y0 Z0 Rw Lf
```

where:

- name is the name of the polygon.
- Other parameters give the location and the size of the mask (see [Figure 72](#)).

NOTE This command works only in MGOALS mode.

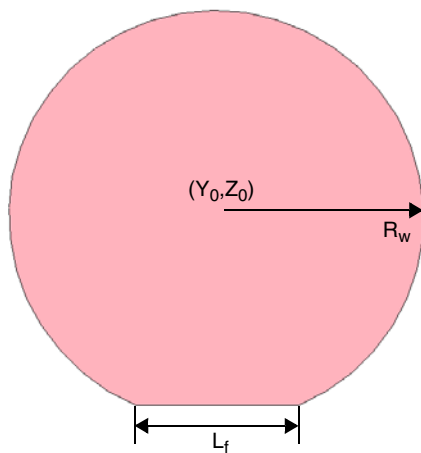


Figure 72 Wafer mask-shaped polygon

PolyHedronEpiDiamond

The PolyHedronEpiDiamond command creates an epitaxial diamond-shaped polyhedron.

The syntax is:

```
PolyHedronEpiDiamond name X0 Y0 Z0 Wepi Lepi Hup Hdown Drecess [material]
```

where:

- name is the name of the polyhedron.
- [Figure 73 on page 246](#) describes the other parameters.

12: Structure Generation
Tailored Structure Generation

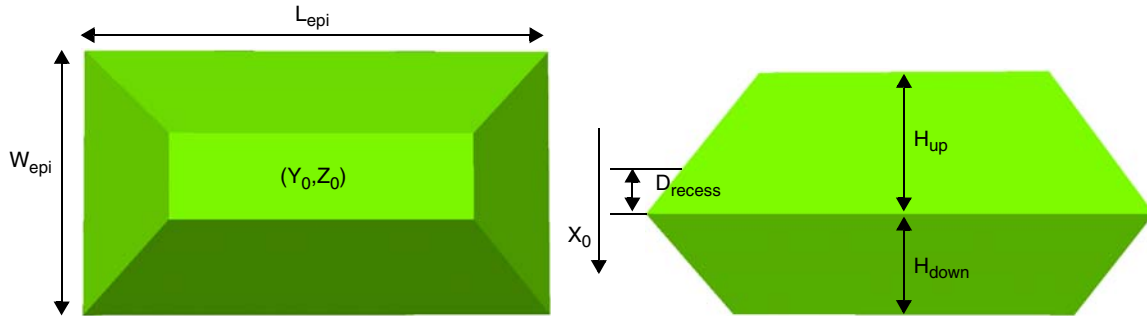


Figure 73 Parameters for generating epitaxial diamond-shaped polyhedron

Tailored Structure Generation

This section describes how to generate a pillar structure and a multiple-pillar structure with a hexahedral mesh.

Pillar Structure With Hexahedral Mesh

Various pillar structures with a hexahedral mesh can be generated quickly to be used for mechanical analysis. Figure 74 shows the geometry of a general pillar structure. One or more concentric or eccentric pillars are embedded in a surrounding material, the pillars can be straight or tapered, and the structure can have up to two pillar layers with three interfaces.

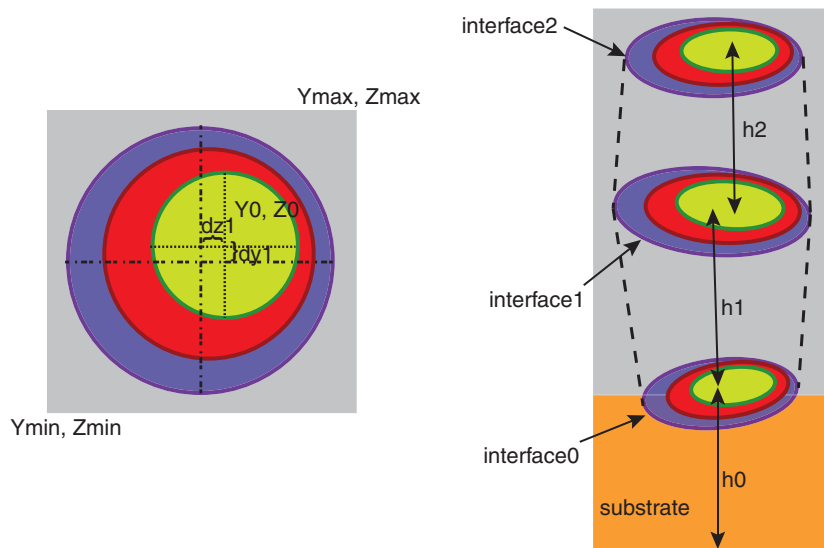


Figure 74 Geometry of a general pillar structure

To generate a pillar structure with a hexahedral mesh, you can use the `init` command with the `unit.cell` and `unit.cell.param` arguments. The syntax for these arguments is described in [init on page 458](#).

The `unit.cell` argument indicates the structure is a pillar.

The `unit.cell.param` argument consists of a list of arguments that specify the geometry of the structure as shown in [Figure 74 on page 246](#).

NOTE After a pillar structure with a hexahedral mesh is generated with the `init` command, no other process steps should be performed. Otherwise, the hexahedral mesh will be destroyed.

Examples of Generating Different Pillar Structures

The following example generates an eccentric tapered pillar structure. The two interfaces are different and form a tapered pillar structure. There are three eccentric circles on each interface, which result in three pillar regions. The materials for the pillar regions from inside to outside are `PolySilicon`, `Aluminum`, and `Copper`, respectively. The substrate material is `Silicon` and the surrounding material is `Oxide`. The grid size in both the `yz` plane and the `x`-direction is $0.02\ \mu\text{m}$ (see [Figure 75 on page 248 \(a\)](#)):

```
init unit.cell=pillar unit.cell.param= { \
  setup= { {Silicon Oxide} {0. 1.5 -0.4 0.4 -0.4 0.4} 0.02} \
  thicknesses= {0.5 1.} \
  pillar.materials= {PolySilicon Aluminum Copper} \
  interface0= { {0. 0. 0.15} {0.02 0.02 0.2} {-0.03 -0.03 0.35} } \
  interface1= { {0. 0. 0.1} {0.02 0.02 0.15} {-0.03 -0.03 0.25} } }
```

The following example generates an eccentric pillar structure with one straight part and one tapered part. It has two pillar layers with three interfaces. The `interface1` argument is the same as `interface0` to generate the straight part (see [Figure 75 \(b\)](#)):

```
init unit.cell=pillar unit.cell.param= { \
  setup= { {Silicon Oxide} {0. 1.5 -0.4 0.4 -0.4 0.4} 0.02} \
  thicknesses= {0.5 0.5 0.5} \
  pillar.materials= {PolySilicon Aluminum Copper} \
  interface0= { {0. 0. 0.15} {0.02 0.02 0.2} {-0.03 -0.03 0.35} } \
  interface1= { {0. 0. 0.15} {0.02 0.02 0.2} {-0.03 -0.03 0.35} } \
  interface2= { {0. 0. 0.1} {0.02 0.02 0.15} {-0.03 -0.03 0.25} } }
```

12: Structure Generation

Tailored Structure Generation

The following example generates an eccentric pillar structure with two tapered parts, and three interfaces are required to define such a structure (see [Figure 75 \(c\)](#)):

```
init unit.cell=pillar unit.cell.param= { \  
  setup= { {Silicon Oxide} {0 1.5 -0.4 0.4 -0.4 0.4} 0.02} \  
  thicknesses= {0.5 0.5 0.5} \  
  pillar.materials= {PolySilicon Aluminum Copper} \  
  interface0= { {0. 0. 0.1} {0.02 0.02 0.15} {-0.03 -0.03 0.25} } \  
  interface1= { {0. 0. 0.15} {0.02 0.02 0.2} {-0.03 -0.03 0.35} } \  
  interface2= { {0. 0. 0.1} {0.02 0.02 0.15} {-0.03 -0.03 0.25} } }
```

The following example generates an eccentric straight pillar structure with varying spacing along the x-direction. For the straight pillar, since `interface1` is the same as `interface0`, `interface1` is not required. To specify the spacing along the x-direction, all the interface locations must be defined, including the top of the structure at $0.0\ \mu\text{m}$, the interface between the substrate and the pillar region at $1.0\ \mu\text{m}$, and the bottom of the structure at $1.5\ \mu\text{m}$ (see [Figure 75 \(d\)](#)):

```
init unit.cell=pillar unit.cell.param= { \  
  setup= { {Silicon Oxide} {0. 1.5 -0.4 0.4 -0.4 0.4} 0.02 \  
  {0. 0.02 1.0 0.02 1.5 0.06} } \  
  thicknesses= {0.5 1.} \  
  pillar.materials= {PolySilicon Aluminum Copper} \  
  interface0= { {0. 0. 0.1} {0.02 0.02 0.15} {-0.03 -0.03 0.25} } }
```

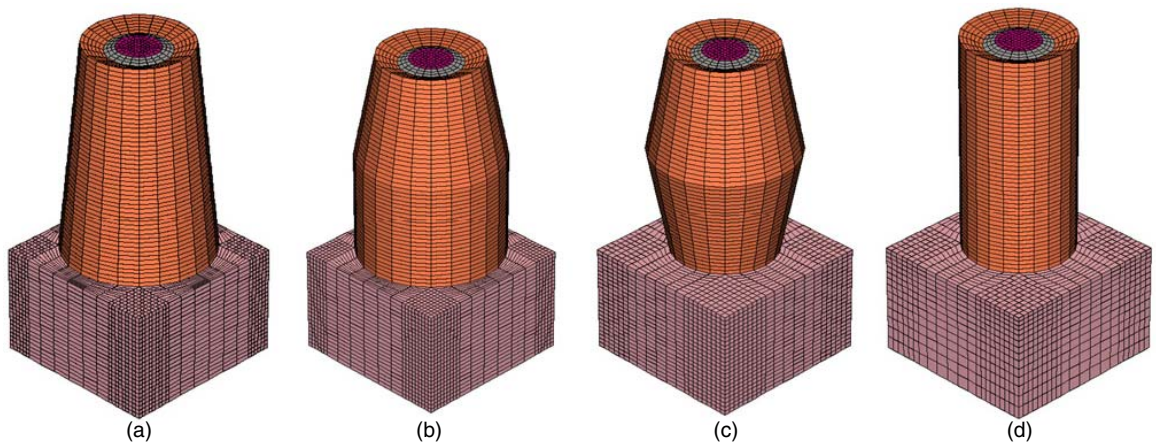


Figure 75 Examples of different pillar structures with hexahedral mesh

Multiple-Pillar Structure With Hexahedral Mesh

Structures that are composed of multiple-pillar cells can also be generated. All cells can be identical or can have process variability that allows you to vary the center locations and radii of pillars according to certain variation types.

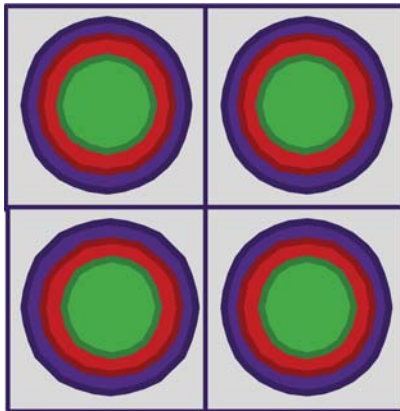


Figure 76 Top view of a multiple-pillar structure: all cells have the same size and the materials in each cell are identical

All pillar cells can be identical or can have process variations. When a parameter has variation, a range must be provided, and the parameter values for different pillar cells will randomly vary within the range around the provided parameter value. For example, if the radius of the inner circle is R_0 for the single cell, then in the multiple cell structure, the radius of each inner circle for each pillar cell will randomly vary in the range of $[R_0 - \text{range}, R_0 + \text{range}]$. The available variation types are Uniform, Linear, and Gaussian, as specified by `variation.type` of the `multiple.cell.param` argument of the `init` command. For example, Figure 77 shows different variation types of the inner circle radius r_0 around the provided radius value R_0 .

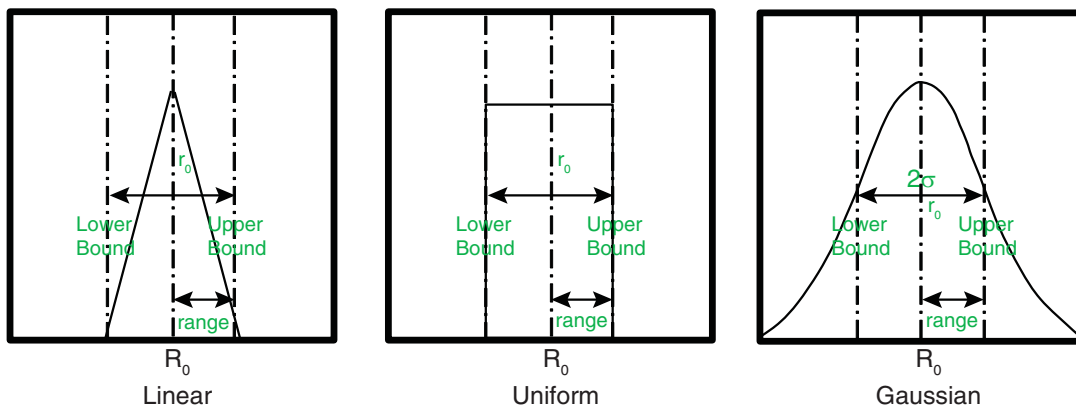


Figure 77 Different variation types

12: Structure Generation

Tailored Structure Generation

For `Uniform` variation, the values of radii in different cells are uniformly distributed between the lower bound and the upper bound.

For `Linear` variation, the values of radii in different cells vary piecewise linearly from the lower bound to the upper bound. The values of radii are calculated according to the following probability density function:

$$p(x|R_0, range) = \frac{|x - R_0|}{range^2} \quad (114)$$

For `Gaussian` variation, the values of radii are calculated according to a Gaussian distribution, which is described by the following probability density function:

$$p(x|R_0, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-R_0)^2}{2\sigma^2}} \quad (115)$$

For `Gaussian` variation, only values between the lower bound and the upper bound are selected.

To generate a multiple-pillar structure, you can use the `init` command with the `unit.cell`, `unit.cell.param`, and `multiple.cell.param` arguments. See [init on page 458](#) for the syntax of these arguments. The `unit.cell.param` argument specifies the geometry of the single pillar cell, and `multiple.cell.param` specifies the number of pillar cells in the structure and process variation parameters for all cells.

The following example generates a multiple-pillar structure composed of 3×2 eccentric pillar cells. The radii of all pillars vary uniformly in a range of $0.01 \mu\text{m}$. The interfaces `interface1` and `interface2` were not specified in the `multiple.cell.param` argument, which means they are the same as `interface0`:

```
init unit.cell=pillar \  
  multiple.cell.param= { \  
    number.pillars = { 3 2 } \  
    variation.type = Uniform \  
    interface0 = { {0.0 0.0 0.01} {0.0 0.0 0.01} {0.0 0.0 0.01} } } \  
  unit.cell.param = { \  
    setup= { {Silicon Oxide} {0. 7. -0.2 0.2 -0.2 0.2} 0.005} \  
    thicknesses= {2. 2.5 2.5} \  
    pillar.materials= {PolySilicon Oxide Nitride} \  
    interface0= { {0.02 0. 0.05} {-0.01 0. 0.08} {-0.02 0. 0.11} } \  
    interface1= { {0.04 0. 0.06} {-0.02 0. 0.12} {-0.04 0. 0.16} } \  
    interface2= { {0.02 0. 0.05} {-0.01 0. 0.08} {-0.02 0. 0.11} } }
```

Masks and Lithographic Patterning

The `photo` command can be used to mimic lithographic patterning. The `photo` command takes a mask and effectively performs a resist spin-on step followed by an exposure bake and etch. The resist layer produced has vertical walls and is a negative of the mask by default, but the positive sense can optionally be created as well.

In addition to the `photo` command, the `etch` and `deposit` commands allow you to use a mask directly. The `mask` argument in the `etch`, `deposit`, and `photo` commands specifies the name of one mask that has been previously defined using a `mask` command or by reading in masks from a layout using the IC WorkBench Edit/View Plus–TCAD Sentaurus interface (see [Chapter 13 on page 285](#)).

The mask will have an effect similar to a masking layer; it limits the etching or deposition process to a certain window. By default, etching is not performed for points inside the mask, unless the `negative` argument is used in the mask definition. Similarly, deposition of a new layer in the `deposit` command and deposition of the photoresist layer in the `photo` command are performed outside the mask unless the `negative` argument is specified in the `mask` command, in which case, deposition occurs inside the mask only.

NOTE Always specify the masks and the simulation domain such that masks do not end exactly on the boundary of the simulation domain, but end inside or extend safely beyond the boundary of the simulation domain.

The `mask` command creates a mask. You can define the geometry of the mask directly in the command file or can read masks from a layout file. Masks defined in the command file must be given a name; otherwise, the names are read from the layout file.

If `list` is specified in the `mask` command, information about the existing masks is printed. If `name` is specified as well, information about the specified mask is printed.

If `clear` is specified in the `mask` command, all previous mask definitions are removed. If `name` is specified as well, only the specified mask is removed.

A mask can be defined directly in the command file using different types of geometry object:

- Segments
- Rectangles
- Polygons

Each mask may be composed of an arbitrary number of such objects.

12: Structure Generation

Masks and Lithographic Patterning

For example, segments are defined as:

```
mask name= pmask segments= { -0.1 0.6 0.7 1.1 }
```

Pairs of subsequent values define the y-coordinates of the beginning and end of one mask segment. Therefore, an even number of coordinates must be specified in the `segments` argument. The pairs may be defined in arbitrary order, and the segments defined by pairs of coordinates may touch or overlap each other. In three dimensions, masks defined by segments are extended over the entire range of z-coordinates.

For example, a rectangular mask is defined as:

```
mask name= nimask left= 0.2 right= 1 front= 0.2 back= 1 negative
```

where:

- `left` and `right` define the minimum and maximum extensions of the mask along the y-axis, respectively.
- `front` and `back` define the minimum and maximum extensions of the mask along the z-axis, respectively.

NOTE Only one rectangle can be specified per `mask` command. The `front` and `back` arguments may be omitted; in this case, the mask is equivalent to a mask with one segment. Additional `mask` commands with the same name can be used to add rectangles. The rectangles defined for a mask may arbitrarily intersect or touch each other.

Masks also can be defined by a list of names of polygons. These named polygons must have been defined before the `mask` command using one `polygon` command for each named polygon:

```
polygon name= LShape2 segments= {0.0 -1.5 0.0 -0.5 0.5 -0.5 0.5 1.5 1.5 \  
1.5 1.5 -1.5}  
mask name= Mask2 polygons= {LShape2} negative
```

NOTE The segments in the `polygon` command are defined as a sequence of y- and z-coordinates. The polygon is closed implicitly by connecting the last point to the first. Each polygon for a mask must not touch or intersect itself. The polygon may be specified with arbitrary orientation (clockwise or counterclockwise in the yz plane).

Masks also can be defined by a combination of segments, rectangles, and polygons. Different objects may touch or overlap each other.

In one dimension, the entire simulation domain is masked if the coordinate origin is masked. Any point along the y-axis in a 2D simulation and any point of the yz rectangle of the simulation domain in a 3D simulation are inside the mask if they are contained in any one of

the geometry objects defined for the mask. Specifying `negative` inverts the mask. In other words, any point outside all the geometry objects defined for the mask is *masked*.

Masks can be empty. With an empty mask, the entire simulation domain is unmasked. Correspondingly, with a negative empty mask, the entire simulation domain will be masked.

These commands can be used to invert a mask at any time after it has been defined:

```
mask name=<c> negative
mask name=<c> !negative
```

NOTE In the `etch` command, the masked area is not etched. While in the `photo` and `deposit` commands, the photoresist or the specified material is deposited in the unmasked area.

Masks also can be combined using a set of Boolean operations, which are specified using the `bool` argument (see [Boolean Masks on page 254](#)).

Layouts that have been defined in the GDSII format can be read into Sentaurus Interconnect using the IC WorkBench Edit/View Plus–TCAD Sentaurus interface (see [Chapter 13 on page 285](#)).

When using a layout, the relation between the layout coordinate system and the Sentaurus Interconnect coordinate system may need to be defined. By default, the layout-x axis corresponds to the Sentaurus Interconnect z-axis. The layout-y axis corresponds to the Sentaurus Interconnect y-axis. This definition matches the default definition of the argument `slice.angle` of the `init` command and the coordinate x- and y-axes when displaying the Sentaurus Interconnect simulation results.

The coordinate transformation between the Sentaurus Interconnect coordinate system and the layout coordinate system can be defined in two ways:

- In the `mask` command that specifies the layout file, the name of one mask may be specified. If a mask with the specified name is contained in the layout file, it is used to position and orientate the simulation domain in the layout.
- Otherwise, a mask with the specified name must have been defined before using a `mask` command. The specified mask is defined in layout coordinates. It may be defined as a rectangle or a polygon, containing at least two points.

In the case of a rectangle defined in a `mask` command, the point with the minimum layout-x and layout-y coordinate is used as the origin of the Sentaurus Interconnect coordinate system. The direction from (`min.layout-x`, `min.layout-y`) to (`min.layout-x`, `max.layout-y`) is used as the Sentaurus Interconnect y-axis.

If a polygon of at least two points is used (for example, a mask defined as a polygon), the first point defined is used to place the origin of the Sentaurus Interconnect coordinate system. The

12: Structure Generation

Masks and Lithographic Patterning

direction from the first to the second point of the mask is used as the orientation of the Sentaurus Interconnect y-axis.

The local coordinates of the specified mask with respect to the selected Sentaurus Interconnect y-axis and origin are used as default extensions of the simulation domain. If a polygon mask with only two points is used, the default extension in the z-direction is 0. The default extension in the y-direction is defined by the distance between the two points. The default extensions in the y- and z-directions as defined by the mask are reported. If no extensions have been defined using the `line y` command, or the `line z` command, or both commands, the default extensions are defined for the simulation when the layout file is read.

If a layout file is loaded, but no mask name is specified, the `Cutline2D` command that may have been specified in the `init` command to define the parameter `slice.angle` will be used to orientate the coordinate systems. The first point specified in the `Cutline2D` command is used as the origin of the Sentaurus Interconnect coordinate system. The direction from the first to the second point is chosen as the direction of the Sentaurus Interconnect y-axis. If `Cutline2D` is used, no default extensions of the simulation domain are defined.

Photoresist Masks

To define photoresist layers, use the `photo` command and specify a mask. Sentaurus Interconnect defines photoresist layers by specifying the minimum thickness of the resist and selecting the name of a mask that has been defined by the `mask` command. By default, the photoresist will be deposited outside the specified mask and will have a flat top similar to resist spin-on. If `negative` has been specified when defining the mask, a photoresist is created inside the mask.

Boolean Masks

Two masks can be combined using the `bool` argument of the `mask` command. [Table 22 on page 255](#) presents the Boolean operations that can be specified in the `bool` argument as well as examples. The `bool` argument only accepts simple expressions.

Complex nested expressions (for example, `bool= "(M1 + M2) - bias(-50, M3 + M4)"`) are not possible. Therefore, they must be reduced to simple operations using a single operator (for example, `bool= "(M1 + M2)"`). Masks requiring a combination of three or more masks must be created using temporary masks to store the intermediate results. In addition, the `bool` argument cannot be used together with `polygon` and `negative`.

Table 22 Boolean operations

Operation	Example	Description
+	bool= "mask1+mask2"	Unites (merges) mask1 and mask2.
-	bool= "mask1-mask2"	Subtracts mask2 from mask1. Note that this is a geometric subtraction. Sentaurus Interconnect takes mask1 and removes from it the portion of mask2 that overlaps mask1. In particular, the operation mask1-mask2 is not equivalent to mask1+(-mask2).
*	bool= "mask1*mask2"	Produces the intersection of mask1 and mask2.
^	bool= "mask1^mask2"	Produces a mask that contains the nonoverlapping portions of mask1 and mask2 (XOR operation).
-	bool= "-mask"	Produces a mask that is the complement of the input mask.
array	bool= "array(nx, ny, dx, dy, mask)"	Produces an array of $n_x \times n_y$ masks separated by a distance specified with dx and dy.
bias	bool= "bias(delta,mask)"	All mask edges on the input mask are offset in the normal direction by the specified amount. A positive delta value expands the mask, while a negative delta shrinks it. Zero or negative area sections of the mask are eliminated from the output mask. Overlapping sections of the mask are merged.
mirror	bool= "mirror(axis, [coordinate,]mask)"	Mirrors a mask with respect to a local axis specified by x or y. The optional parameter coordinate defines the local intercept of the local axis-aligned reference line used for the mirror operation.
offset	bool= "offset(dy, dz, mask)"	Translates the mask by the specified amount. The dz parameter is ignored in two dimensions.
over_under	bool= "over_under(delta, mask)"	Expands and then shrinks the input mask by delta. This effectively merges areas in close proximity and is equivalent to bias(delta, bias(-delta, mask)).
rotate	bool= "rotate(direction,mask)" bool= "rotate(direction,x0, y0, mask)"	Produces a mask that is rotated with respect to the input mask. The direction parameter can be either left-90 or right-90. The optional x0 and y0 parameters specify the center of rotation. They are specified using the local mask coordinate system. If x0 and y0 are not specified, the rotation is performed with respect to the center of the bounding box of the mask.

Table 22 Boolean operations

Operation	Example	Description
scale	bool= "scale(factor,mask) " bool= "scale(factor,x0, y0, mask) "	Produces a mask that is scaled with respect to the input mask using the floating-point value of <code>factor</code> . The optional <code>x0</code> and <code>y0</code> parameters specify the center of scaling. They are specified using the local mask coordinate system. If <code>x0</code> and <code>y0</code> are not specified, the scaling is performed with respect to the center of the bounding box of the mask.
under_over	bool= "under_over(delta, mask) "	Shrinks and then expands the input mask by <code>delta</code> . This eliminates small areas and is equivalent to <code>bias(delta, bias(-delta, mask))</code> .

Line Edge Roughness Effect

Line edge roughness (LER) is the deviation of feature edges from ideal straight lines due to statistical fluctuations in photolithographic processes. Sentaurus Interconnect uses the `line_edge_roughness` command to apply randomized deviations to straight mask edges, for example:

```
line_edge_roughness normal= "Z" masks= {mask1} correlation.length= 25.00<nm> \
standard.deviation= 5.00<nm> max.segment.length= 5.00<nm>
```

The random noise function f_{random} applied to mask edges by the `line_edge_roughness` command is generated from the power spectrum of a Gaussian autocorrelation function. The Gaussian autocorrelation shape is characterized by the standard deviation distance δ specified by the `standard.deviation` argument and the correlation length λ , specified by the `correlation.length` argument:

$$\text{Autocorrelation}(f_{\text{random}}) = \delta^2 \sqrt{\pi} \lambda e^{-(x^2/\lambda^2)} \quad (116)$$

f_{random} is obtained by Fourier synthesis, applying the inverse Fourier transform to Eq. 116, after adding random phases. In this way, random deviations of the mask edges can be obtained from run to run, which correspond to LER profiles having the same standard deviation δ and correlation length λ .

These random deviations are added in discrete form to the mask edges in question. First, the mask edge is subdivided into discrete segments complying with the `max.segment.length` argument. Second, the deviation at each segment endpoint is added in the direction normal to the initial mask edge orientation.

LER is applied to all edges of the mask by default. You can limit which edges in a named mask receive LER using `normal`, which specifies either the y-axis ("Y") or the z-axis ("Z"). If

`normal` is specified, only those edges in the named masks normal to the given axis are chosen for LER to be applied. LER is applied only once per mask. Mask segments along the device bounding box do not receive LER.

The option `!random.reseed` bypasses the reseeding of the random number generator before the random phases are added. Using this option, the shape of the noise function and, therefore, the LER result, can be reproduced from one run to the next if needed for comparison.

The option `random.seed` can reproduce specific LER calculations from one run to the next by setting the same random seed in both runs. When stored in a TDR file in split simulations, `random.seed` is included when saving `line_edge_roughness` to the TDR file, even if it is not specified by users, to ensure proper reproduction of the same LER in a subsequent reload of the TDR file.

NOTE The structure is extruded automatically to three dimensions if it is less than three dimensions and the `line_edge_roughness` command is used.

The following strategy is used to address the problem of nearly collinear LER mask points that may trigger removal by decimation during meshing. Avoiding the decimation of nearly collinear LER mask points is desirable because removing such points may perturb the power spectrum of the Gaussian autocorrelation function represented by the mask shape and may also result in meshing difficulties.

If `max.tries` is set to a nonzero value, LER masks are checked for nearly collinear points, which would result in decimation by the mesher:

```
line_edge_roughness max.tries= 30
```

If any points in the LER mask are decimated by the mesher, based on the current `mgoals accuracy=<n>` setting, the LER mask is rejected and the LER generation process is restarted. After each restart, the detection and restart process is repeated until an acceptable LER mask is generated or until the number of attempts exceeds `max.tries`.

When `max.tries` is exceeded, Sentaurus Interconnect stops with an error message that suggests using a smaller value of `mgoals accuracy=<n>` or a larger value of `max.segment.length` in the `line_edge_roughness` command.

The default value of `max.tries` is 0, meaning no decimation check is performed, no retries are attempted, and the initial mask LER is accepted as it is, even with nearly collinear points.

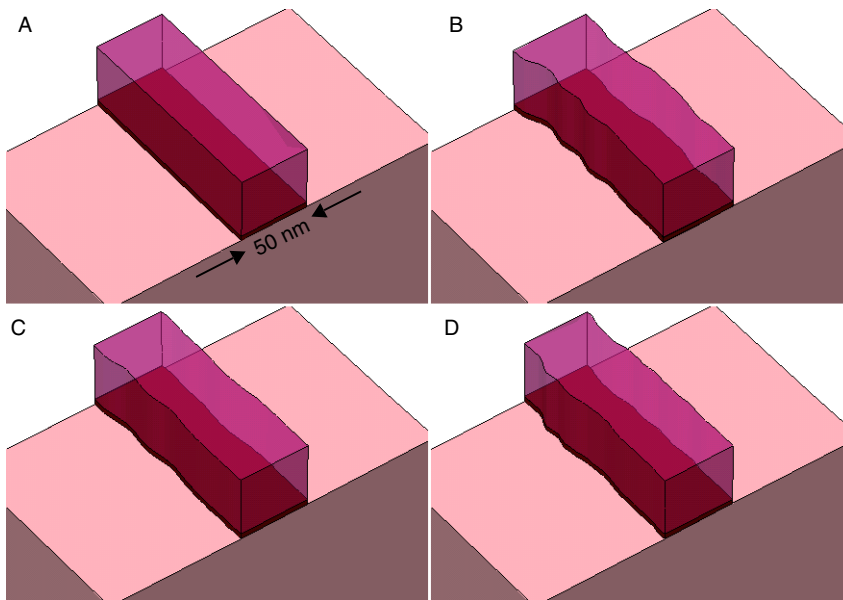


Figure 78 Example of 3D structure with LER applied using different values of standard deviation δ and correlation length λ : (A) no LER applied, (B) LER applied with $\delta = 4$ nm and $\lambda = 20$ nm, (C) LER applied with $\delta = 2$ nm and $\lambda = 20$ nm, and (D) LER applied with $\delta = 2$ nm and $\lambda = 12$ nm

Mirror Boundary Conditions

The addition of LER is inherently 3D and nonsymmetric; therefore, it is not easily compatible as a simulation problem with the assumption of symmetry reduction. Adding LER to a structure reduced by symmetry may produce rough geometric transitions at joined geometric boundaries when pieces are reunited.

To accommodate the possibility of joining symmetry-reduced structures, an additional argument `smooth.points` defines the smoothing of LER at structure boundaries. The default value of `smooth.points` is 0, implying no smoothing.

The use of nonzero `smooth.points` alters the LER function that creates mirror boundary conditions at the structure boundaries. The smoothing occurs at the intersection of masks with the structure boundary sides. The argument `smooth.points` defines the number of segments in the discrete LER function, counting from the boundary edge of the mask that should be adjusted or smoothed. The length of segments in the discrete LER function is determined using `max.segment.length`.

Smoothing is calculated by generating the LER function, then creating a virtual mirroring at the structure boundary edge where a cubic spline interpolation is applied with a mirror

boundary condition. The spline fit is made with the LER at `smooth.points` from the boundary. The smoothed LER function is then applied to the mask.

Applying nonzero values of `smooth.points` affects the randomness and the power spectrum characteristics of the LER offset function at the boundaries as a trade-off for permitting the smooth joining of symmetry-reduced structures.

Geometric Transformations

Transformations supported in Sentaurus Interconnect are reflection, stretch, cut, rotation, translation, and flip:

- The `transform reflect` command is used with `left`, `right`, `front`, or `back` to perform the reflection centered on the outer boundary of the simulation domain. At the reflection side, regions are not merged immediately to allow a clean `transform cut` afterwards if required. The grid can be merged with the `grid merge` command manually, but if not, the structure will naturally be merged for any geometry-changing operation later (except `transform reflect`). It is also possible to discard the original structure when reflecting by specifying `!keep.original`.
- The `transform stretch` command is used to extend the structure. The algorithm cuts the mesh in two pieces using a cutplane defined by `location` and the `left`, `right`, `front`, or `back` arguments. The resulting pieces of mesh are translated perpendicular to the cutplane to create a gap of size `length` in the direction given by `left`, `right`, `front`, or `back`. After this translation, a set of new elements is inserted to connect both sections of the mesh.
- The `transform cut` command crops the structure. The algorithm cuts the structure in two pieces using a cutplane defined by `location` and the `left`, `right`, `front`, or `back` arguments. If `location` is omitted, the algorithm will cut the structure in the middle. The `left`, `right`, `front`, or `back` arguments instruct the algorithm to remove the portion of the structure along the given direction. For a more general crop operation, the `cut` argument can be used with `min` and `max` that specify the cropping box.
- The `transform rotate` command rotates the structure. The `axis` and `angle` arguments must be specified. Only 90° , 180° , and 270° angles are allowed for the x-axis, and 180° for the y- and z-axis. For 2D simulations, rotations will produce an extruded 3D simulation.
- The `transform translate` command shifts the structure by the specified quantity.

Refinement Handling During Transformation

All transformations apply to the existing refinements created with either the `refinebox` command or the `line` command by default.

To disable this feature, use:

```
pdbSet Grid Transform.Updates.Refinement 0
```

Contact Handling During Transformation

The only special contact handling occurs during `transform reflect`. In this case, contacts that straddle or touch the reflecting plane are enlarged to the reflected area (only one contact remains). The remainder of the contacts are duplicated and are renamed by appending a suffix as follows:

- For right or left reflection, the contact on the left after reflection will be named `<original contact name>.1` (where `<original contact name>` was the name of the contact before the reflection operation), and the contact on the right after reflection will be named `<original contact name>.2`.
- Similarly, for front or back reflection, the front contact (which has a larger z-coordinate) will be named `<original contact name>.2`, and the back contact will be named `<original contact name>.1`.
- For up or down reflection, the upper contact will be named `<original contact name>.1`, and the lower contact will be named `<original contact name>.2`.

You can rename contacts after the `transform reflect` command, or at any time, using the command:

```
contact name=<c> new.name=<c>
```

For example, after reflection, you can use the command:

```
contact name= SourceDrainContact.1 new.name= Source
```

where the original contact name before reflection was `SourceDrainContact`.

NOTE It is recommended to specify all contacts *after* all transformation operations other than `transform reflect` to avoid problems during contact creation.

Reflection

The `transform reflect` command reflects the structure about the left, right, front, or back boundary (at minimum y, maximum y, maximum z, or minimum z). If any remeshing or other mesh modification operations are performed after a `transform reflect` command, the symmetry will be lost. Using `!keep.original` discards the original structure leaving only the reflected one.

Examples:

```
transform reflect left
transform reflect ymin
transform reflect front
transform reflect left !keep.original
```

NOTE The option `remesh` is disabled in `transform reflect` because it may disrupt the symmetry of the reflected structure. However, the command `grid merge` can be used afterwards to remove same-material interfaces at the reflecting plane.

Refinement Handling During Reflection

In the case of `!keep.original`, lines coming from both the `line` command and refinement boxes are reflected along with the structure. However, when the original structure is kept, some special handling is required.

Typically, during any geometry operation, lines created with the `line` command that have been defined outside the bounding box will be removed. Therefore, there is no danger of the reflected lines conflicting with the original lines.

For refinements created with refinement boxes, if the refinement box is constrained spatially (that is, `min` or `max` has been used in the definition), the box will be duplicated, and the name of the new box will be `reflected_<c>`, where `<c>` is the name of the original refinement box. If the original and reflected refinement boxes overlap, there is no problem since the refinement criteria are the same.

Stretch

The `transform stretch` command stretches the structure in the left, right, down, up, front, or back directions at a given coordinate location by offsetting one side of the structure by the specified length. If there is no vertical line of edges at the specified location, MGOALS creates such a line and then stretches the structure. The data at the two ends of the stretched region is exactly the same as that of the unstretched mesh.

Examples:

```
transform stretch location=0.001 length=5 right
transform stretch down loc=0.5 length=200
```

NOTE Do not use a negative stretch length.

Refinement Handling During Stretch

During the stretch operation, mesh lines that were created with the `line` command in the part of the structure being stretched are translated with the structure. No new lines are introduced into the expanded region. The refinement boxes that straddle the stretch location increase in size by the stretch distance to follow the structure.

Cut

The `transform cut` command cuts at or near the requested coordinate location. The location defines a line in two dimensions, or a plane in three dimensions, that divides the structure into the left and right, or front and back, or up and down parts. You can select the left/right, or front/back, or up/down region to be removed. If a line of element edges in two dimensions, or a plane of element faces in three dimensions, can be identified by MGOALS, the operation eliminates only the elements in the removed region. This works well if a structure had been reflected and needs to be cut back to the original (unreflected) structure. If a line of element edges in two dimensions, or a plane of element faces in three dimensions, cannot be found, a mesh-cutting operation is performed. By default, MGOALS tries to find a mesh line or plane near the specified coordinate. Then, MGOALS removes entire mesh elements rather than cutting mesh elements to avoid arbitrarily small edges and poor element quality.

To disable the search feature and to perform the operation exactly where specified, use `!mesh.align` in the `transform` command, which will invoke a remesh unless `!remesh` also is specified.

Example:

```
transform cut location=0.5 right
```

You also can use the `transform cut` command to crop the mesh by specifying a rectangle or brick defined by the upper-left-front and lower-right-back corners, specified with the `min` and `max` arguments. The cut operation retains the region enclosed by the rectangle or brick. By default, MGOALS tries to find a mesh line or plane near the specified coordinate and removes entire mesh elements instead of cutting the mesh (which could lead to arbitrarily small edges and poor element quality).

To disable searching of a nearby mesh line or plane, and to perform the operation exactly at the specified location, specify `!mesh.align`, which automatically invokes a remesh unless `!remesh` also is specified.

Example:

```
transform cut min= {-10 1.35 0.15} max= {10 1.65 0.4}
```

Refinement Handling During Cut

During the cut operation, lines created with the `line` command that are outside of the simulation domain after the cut are removed. Similarly, any refinement box that lies completely outside the simulation domain after the cut is removed as well.

Flip and Backside Processing

The `transform flip` command provides a convenient way to perform process steps on the back of a wafer. During the `transform flip` command, the structure is rotated 180° about a line by default in the center of the structure parallel to the y-axis. Therefore, the structure is upside down after the flip and in the same location.

Because most operations in Sentaurus Interconnect require a gas region on top, a gas region is added automatically. In addition, many meshing operations require a solid material at the bottom of the structure, so the `Gas` region that was previously on top of the structure is converted to an auxiliary material called `BackMat`. If the structure is flipped again, the reverse happens, namely, `BackMat` is converted to `Gas`, and `Gas` is converted to `BackMat`. Any operation is allowed on a structure that has been flipped one or more times; however, the current bottom of the structure is never an active surface for any operation such as etching and deposition.

There is great flexibility in handling the auxiliary material at the back of the structure. The material itself defaults to `BackMat` as mentioned, but you can choose another material using:

```
pdbSet Grid Back.Material <material>
```

The material `BackMat` inherits its parameters from (is Like) `Gas` so that it behaves like gas for thermal and electrical simulations. For mechanics, the only way to obtain `Gas`-like mechanics boundary conditions at interfaces to the back material is to use an actual `Gas` region. Therefore, the back material is converted automatically to `Gas` before each mechanics call and is converted back directly afterwards. Finally, when a region of material `BackMat` is saved in a structure, it is first converted to `Gas`, so that other tools reading the structure will have the proper material. However, it is also given a tag specific to Sentaurus Interconnect, so that Sentaurus Interconnect knows the region should actually be `BackMat`.

Refinement Handling During Flip

Refinements during a flip operation are handled in the same way as refinements during reflection in the case of `!keep.original`. Lines coming from both the `line` command and refinement boxes are reflected along with the structure.

Rotation

The `transform rotate` command rotates the structure at the specified angle in the specified axis using (0,0,0) as the rotation center. It accepts two arguments `angle` and `axis` to specify the rotation angle and the rotation axis, respectively. For Y and Z, 180 degrees can be specified. For X, 90, 180, and 270 degrees are allowed.

It might happen that, during the rotation, the existing initial gas must be moved to a side or the bottom of the structure instead of being at the top. In these cases, new gas will be added to the top.

The `transform rotate` command applies to 2D and 3D simulations. For 2D simulations, a rotation in the z-axis will produce another 2D simulation. However, rotations in the x-axis and y-axis will produce a structure equivalent to extruding the z-axis and then performing the rotation.

Refinement Handling During Rotation

During a rotation operation, lines coming from both the `line` command and refinement boxes are rotated along with the structure.

Translation

The `transform translate` command does not change the aspect of the structure. It only adds the coordinate specified in the `translate` argument to all nodes, that is, it displaces the structure or shifts it in space. It is equivalent to changing the origin of coordinates by a fixed quantity.

Similarly, mesh lines created with the `line` command and the bounding box of refinement boxes (which are specified with `min` and `max` of the `refinebox` command) are translated with the structure.

MGOALS

By default, etching and deposition operations are performed using MGOALS in all dimensions. MGOALS operates as follows:

- The starting structure is analyzed for the interfaces that will change during the operation.
- The geometry-changing operations are performed.

- In two dimensions, the entire structure is remeshed. During remeshing, nodes in the silicon region are retained as much as possible in their original locations. In most cases, a high percentage of nodes are retained after remeshing. This minimizes interpolation errors. In three dimensions, the structure is remeshed only if the next step requires an up-to-date mesh.

MGOALS Boundary-Moving Algorithms

MGOALS uses either the analytic method or the fast level-set method to perform boundary-modifying operations. In general, the analytic method is fast, less memory intensive, and more accurate, and uses a simplified string algorithm. However, it cannot handle deposition in concave regions or etching of convex areas when there are boundary collisions and self-intersections.

Due to speed and accuracy advantages, MGOALS always tries to perform an analytic operation. If self-intersections are detected in the new boundary, MGOALS automatically switches from the analytic method to the fast level-set method. Both the analytic method and the fast level-set method can handle simple etching and deposition processes.

Besides the analytic method and the fast level-set method, the general time-stepping level-set method is available to handle more complex etching types such as Fourier, crystallographic, and multimaterial etching, and to include shadowing effects.

Both level-set methods use an approach similar to that described in [1]:

- First, the level-set method identifies the interface or the part of an interface to be moved. This computation is based on nonetched overlayers, masks, and if necessary, visibility due to directional constraints specified by users.
- Second, evolution of the moving interface is performed using either the fast-marching scheme, which solves the time-independent boundary-value formulation of the Hamilton–Jacobi equation (or Eikonal equation), or the *full level-set*, which is a time dependent, initial-value formulation of the same equation. The fast-marching scheme computes the new boundary location for all times in a single step. The nature of the equation is such that it captures and handles collisions. However, the equation cannot identify when the collision actually occurred. The full level-set formulation is used for multimaterial, Fourier, and crystallographic etching, and for handling shadowing effects. Its time-stepping algorithm allows for recalculating the front velocity at every time step.

In MGOALS, the fast-marching scheme and the level-set equations are solved on a separate Cartesian mesh that is independent of the simulation grid. For a description of the parameters that control the Cartesian mesh, see [MGOALS Boundary-Moving Parameters on page 266](#).

12: Structure Generation

MGOALS

After solving the level-set equations, the newly created boundary is extracted from the level-set function on the Cartesian mesh and then incorporated into the simulation mesh. The exact replication of the extracted boundary in the mesh can be expensive and can transfer unwanted noise from the level-set solution into the structure. To resolve these issues, MGOALS allows a certain smoothing to be performed on the extracted boundary.

In two dimensions, to incorporate the new boundary into the simulation grid, a simplified meshing step is performed. A simple mesh is created for the modified regions and connected to the mesh in unchanged regions. Since this mesh is not suitable for process simulation, by default, a full remesh is performed after each etching and deposition step.

In three dimensions, almost all boundary-modifying operations performed by MGOALS use the analytic method. The only exceptions are isotropic deposition and etching, which are performed using the fast level-set method. The new material boundary is integrated into the structure using a set of polyhedral Boolean operations.

MGOALS Boundary-Moving Parameters

Parameters to specify the resolution of the Cartesian mesh and the interface fidelity are defined in an `mgoals` command before the `etch` or `deposit` command. These parameters are applied to the entire structure. The interface quality and resolution are controlled by `accuracy`, `resolution`, and `full.resolution`. The actual size and the placement of the Cartesian mesh bounding box is calculated starting with the initial interface being etched, extended based on the time and rates given by the user, or in the case of etch stops, extended based on the distance from the initial front to the etch stops.

The accuracy Argument

The `accuracy` argument controls the noise and features at an interface. A small value of `accuracy` allows only small deviations between the boundary extracted from the level-set method and the piecewise linear segments incorporated into the simulation mesh. As a result, a large number of small segments may be created. In addition, a value of `accuracy` that is too small may *interpret* numeric noise as surface features, which MGOALS requires to reproduce in the simulation mesh. The default value and unit for `accuracy` is $1.0 \times 10^{-5} \mu\text{m}$.

NOTE The default value for `accuracy` works well for small structures that are approximately $1 \mu\text{m} \times 1 \mu\text{m}$ in size. For larger structures, you can scale up this argument to reduce geometric noise. However, using too large a value could lead to self-intersection around sharp features, to meshing failure, and to noticeable loss of curvature in rounded areas.

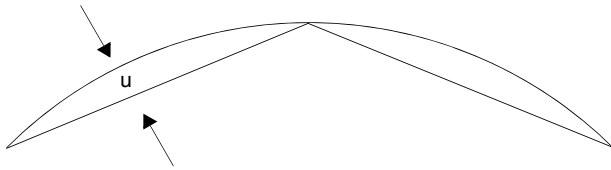


Figure 79 The curved surface represents the extracted new boundary and the piecewise linear segments represent the simplified boundary incorporated into the simulation mesh; the accuracy argument ensures that $u \leq \text{accuracy}$

The resolution Argument

The value of the `resolution` argument controls the element size in the Cartesian mesh used to perform level set-based etching and deposition. Since the thickness of the layer to be deposited or etched is user specified, the grid size is defined as a fraction of the thickness of the modified layer. The size of each grid element of the level-set mesh is given by the product of the value of `resolution` and the etching or deposition thickness.

The `resolution` is specified in the `mgoals` command, and the etching or deposition thickness is specified in the `etch` or `deposit` command, respectively. This scheme usually provides a good approximation of the required level-set resolution and is computationally efficient. The default value for `resolution` is 0.1.

NOTE Providing a small `resolution` for thin layers may lead to excessive time and memory consumption. For example, an isotropic deposition of 1 nm thickness with `resolution=0.1` will result in facets of approximately 1 Å in size. In this case, `resolution=0.3` would lead to coarser facets in rounded areas.

NOTE For a thick etching or deposition, it may be necessary to reduce `resolution`. For example, `resolution=0.1` for a 1 μm deposition leads to a level-set grid size of 100 nm, which may lead to a poor approximation of sharp corners and rounded areas in the new boundary.

NOTE It is not recommended to set the `resolution` argument to a value greater than 0.3. Doing so may compromise the integrity of the layers being deposited or etched, since they may not be resolved by the level-set grid.

General Time-Stepping Level-Set Parameters

The general time-stepping level-set method has some additional parameters to control and balance accuracy, simulation time, and memory use.

12: Structure Generation

MGOALS

Usually, the full time-stepping level-set method is used in situations where more intricate boundaries will be generated. This method is needed for Fourier, crystallographic, and multimaterial etching, and for etching with shadowing. It is also used if the option `force.full.levelset` is specified.

The `full.resolution` argument can be used for the time-stepping level-set method in the same way `resolution` is used for the fast level-set method. You also can specify the actual spacing of the Cartesian mesh in the x-direction, or y-direction, or both directions, with the `dx` and `dy` arguments (and the z-direction in three dimensions with `dz`). Reducing the mesh size causes the time-stepping level-set method to allocate more memory, to take smaller time steps, and to increase the solve time for each time step, thereby increasing the overall simulation time.

NOTE In previous releases, the *Upwind* formulation of the time-stepping level-set method provided users with the arguments `reinitfrequency` and `reinititerations` of the `mgoals` command to control the frequency and quality of level-set reinitialization. The currently implemented Lax–Friedrichs formulation does not provide these parameters to users, since reinitialization is performed at each time step.

Level-Set Cartesian Mesh and Resolution: Internal Calculations

The Cartesian mesh extent or bounding box, the `resolution` and `full.resolution` criteria, and the grid spacing criteria `dx`, `dy`, and `dz` interplay in the following ways.

The Cartesian mesh encompasses the initial interface between the Gas and all the materials the user has defined to be etched. It also encompasses the entire movement of the etching front expected throughout the entire etching process. In the case of time-based etching with the `time` argument, an etching distance is computed, based on the requested etching time multiplied by the maximum expected etching rate. The `resolution` or `full.resolution` argument is then used as the approximate mesh spacing unless overridden by `dx`, `dy`, or `dz`. The number of resulting Cartesian mesh lines follows as required to achieve the required mesh spacing within the Cartesian mesh bounding box.

In the case of an etch stop, the bounding box of the Cartesian mesh is calculated based on the initial gas–etching material interface, and its extent is determined by the position of etch stops in the simulation domain, that is, the estimated maximum etching distance. The target mesh spacing in two dimensions is the `resolution` or `full.resolution` multiplied by the estimated maximum etching distance. In three dimensions, the target mesh spacing is set to `min.levelset.size`, because the estimated maximum etching distance is not calculated in three dimensions for this purpose. From the size of the Cartesian mesh bounding box and the target mesh spacing, possibly overridden by `dx`, `dy`, and `dz`, the number of Cartesian mesh lines is determined.

Limitations of Level Set

As a general approach, while the level-set method is especially useful for shadowing, multimaterial, sophisticated etch velocity functions (for example, surface normal dependent or crystal direction dependent), and for complex evolution of etch surfaces (that is, complicated structures), it is generally not a good choice when sharp or exact corners, and straight or exact etch walls, are required, such as in anisotropic etching.

This limitation is due to the implicit representation of the structure as a rectilinear grid of distance functions used to calculate the evolution of the moving surfaces.

MGOALS 3D Boundary-Moving Algorithms

In three dimensions, a combination of level set, fast marching, and analytic techniques are used to perform geometric operations similar to the 2D mode. In three dimensions, MGOALS can reliably handle complicated polyhedral boundaries. This mode can perform any of the geometric operations contained in the `deposit`, `etch`, `photo`, `polyhedron`, and `transform` commands.

NOTE After any one of these process steps is performed in three dimensions, all subsequent geometric operations should be performed using MGOALS.

The use of the level-set method for thin etches or deposits can be prohibitively CPU intensive and memory intensive, especially for large structures or for very thin etching or deposition steps. To address this issue, etches and deposits less than 1 nm use the analytic method by default. The thickness of this cutoff can be modified using `analytic.thickness` of the `mgoals` command.

To produce meshes with the highest quality elements and the fewest points, you should reduce the number of interfaces in the structure. This is especially true for 3D simulations.

NOTE When the command `sde off` is specified, region-merging is switched on regardless of the previous setting.

Summary of MGOALS Etching

Table 23 summarizes the methods used internally to implement etching:

- Level set (LF) – General time-stepping level-set method with Lax–Friedrichs formulation.
This is the most general method. All level-set methods have the disadvantage of a certain amount of rounding at corners and edges. The LF formulation has added stability, which can result in slightly more rounding at corners and edges.
- Fastmarch – The fast level-set method is used in simple 1D or 2D directional and anisotropic etching.
- Geometric – Three-dimensional etching algorithm inserts an analytically calculated etching shape into the device structure.
- Analytic – One-dimensional or 2D etching algorithm calculates and inserts an analytically calculated etching shape into the device structure.

Table 23 Summary of etching algorithms used for different etching input parameters

Material	Shadowing	Etching type	Structure dimension	Etchstop mechanism			
				Time and rate	Material Etchstop	Thickness	
Single material	No shadowing	Isotropic	1D	Analytic	Level set (LF)	Analytic	
			2D	Fastmarch	Level set (LF)	Fastmarch	
			3D	Geometric	Level set (LF)	Geometric	
		Fourier	1D/2D/3D	Level set (LF)	Level set (LF)	Not supported	
		Directional	1D/2D	Analytic	Analytic	Analytic	
			3D	Geometric	Level set (LF)	Geometric	
		Anisotropic	1D/2D	Analytic	Analytic	Analytic	
			3D	Geometric	Level set (LF)	Geometric	
		Shadowing	Isotropic	2D	Level set (LF)	Level set (LF)	Level set (LF)
				3D	Level set (LF)	Level set (LF)	Level set (LF)
	Fourier		2D/3D	Level set (LF)	Level set (LF)	Not supported	
	Directional		2D/3D	Level set (LF)	Level set (LF)	Level set (LF)	
	Anisotropic		2D/3D	Level set (LF)	Level set (LF)	Level set (LF)	

Table 23 Summary of etching algorithms used for different etching input parameters

Material	Shadowing	Etching type	Structure dimension	Etchstop mechanism		
				Time and rate	Material Etchstop	Thickness
Multimaterial	No shadowing	Isotropic	2D	Level set (LF)	Level set (LF)	Fastmarch
			3D	Level set (LF)	Level set (LF)	Geometric
		Fourier	2D/3D	Level set (LF)	Level set (LF)	Not supported
		Directional	2D	Level set (LF)	Analytic	Analytic
			3D	Level set (LF)	Level set (LF)	Geometric
		Anisotropic	2D	Level set (LF)	Analytic	Analytic
	3D		Level set (LF)	Level set (LF)	Geometric	
	Shadowing	Isotropic	2D	Level set (LF)	Level set (LF)	Level set (LF)
			3D	Level set (LF)	Level set (LF)	Geometric
		Fourier	2D/3D	Level set (LF)	Level set (LF)	Not supported
		Directional	2D	Level set (LF)	Level set (LF)	Level set (LF)
			3D	Level set (LF)	Level set (LF)	Geometric
		Anisotropic	2D	Level set (LF)	Level set (LF)	Level set (LF)
			3D	Level set (LF)	Level set (LF)	Geometric

MGOALS Backward Compatibility

Default parameters and algorithm settings used by MGOALS can change from release to release in the pursuit of more accurate, more realistic, and more stable structure generation results. To use the default parameters and settings from a previous release, type the required release as a string in the `mgoals` command, for example:

```
mgoals "M-2016.12-SP1"
```

Partial support for this backward compatibility is available starting with Version D-2010.03 with more complete support starting with Version E-2010.12.

Boundary Repair Algorithm

Anisotropic or directional operations can produce residual material when the walls of the etched material are not perfectly vertical or aligned to the etching beam. These residual materials usually cause problems for the mesh generator since they contain sharp angles and

12: Structure Generation

MGOALS

small features that cannot be meshed. To correct this problem, the boundary repair algorithm can analyze the structure and eliminate small unwanted features.

The boundary repair algorithm can be used with the `deposit`, `etch`, and `photo` commands. These commands include the `repair` option that controls this algorithm. The boundary repair algorithm is enabled by default in 3D and disabled in 2D.

To activate or deactivate the boundary repair algorithm, include `repair` or `!repair` in the command specification, for example:

```
etch material= {Silicon} anisotropic rate= 0.001 time= 1.0 !repair
```

The repair algorithm also can be controlled globally for the `etch` and `deposit` commands. To disable or enable repairs, use the following options in the `mgoals` command (see [mgoals on page 510](#)):

- `repair.2d`, `repair.2d.deposit`, `repair.2d.etch`, `repair.2d.photo`
- `repair.3d`, `repair.3d.deposit`, `repair.3d.etch`, `repair.3d.photo`

Structure Assembly in MGOALS Mode

Sentaurus Interconnect can read a 2D or 3D structure from a file and paste it into the current 2D or 3D simulation, respectively.

To perform structure assembly, use:

```
paste (direction= back | front | left | right) tdr=<c>
```

where:

- In two dimensions, `direction` can be `left` or `right`.
- In three dimensions, `direction` can be `back`, `front`, `left`, or `right`.
- `tdr` specifies the TDR file from which a structure will be read and pasted into a simulation.

Sentaurus Interconnect automatically shifts the structure read from the file to the appropriate quantity in `x`, `y`, and `z` to fit to the current structure. Nevertheless, Sentaurus Interconnect will not automatically stretch the incoming structure. Consequently, for the operation to succeed, the sizes of the pasting planes of the incoming structure and the existing one must be the same.

The values of the fields are conserved for each structure and are interpolated at the interface between the structures.

NOTE Structure assembly requires that the structure read from the file must have the same dimensionality, 2D or 3D, as the existing structure.

NOTE In three dimensions, structure assembly must be performed in MGOALS mode. That is, structure assembly is not available when `sde on` is specified.

Multithreading

Some of the more sophisticated etching and deposition types require the use of the level-set method (such as multimaterial etching, crystallographic etching and deposition, and Fourier etching and deposition). This can be time consuming, especially for 3D summations. To minimize simulation time, MGOALS allows for a multithreaded solution of the level-set equations.

The multithreaded operation can be invoked using:

```
math numThreads=<i>
```

or:

```
math numThreadsMGoals=<i>
```

where `<i>` is an integer. It is suggested to keep `<i>` at or below 4 to obtain reliable speed improvement.

Insertion: Internal Mode

This section describes different insertion operations.

Inserting Segments

The `insert` command defines and inserts regions defined by segments in one dimension (see [insert on page 464](#)). You can choose which materials or regions are replaced, and the name and the material of the new region.

Multiple regions can be inserted in one step. However, to insert multiple regions, the name cannot be specified. If multiple regions are inserted, machine-generated names are used.

Inserting Polygons

Two-dimensional regions defined by polygons can be created and inserted directly into a 2D simulation.

Polygons are created with the `polygon` command, which accepts several arguments to specify how to create the polygon:

- `points` specifies a list of points defining the polygon.
- `rectangle`, with `min` and `max`, specifies the rectangle limits.
- `segments` specifies pairs of numbers, which are the coordinates where each segment starts and the previous one finishes.
- `xy` specifies the polygon will be created in the `xy` plane.

Since the standard use of the `polygon` command (see [Masks and Lithographic Patterning on page 251](#)) is to create masks for the `deposit`, `etch`, and `photo` commands, the default coordinates are `y` and `z` for the `segments` and `min` and `max` arguments. Consequently, the option `xy` must be specified in order for the polygon to be created in the `xy` plane instead of the `yz` plane.

- `tdr` specifies the name of a TDR file from which to import the polygon, and the argument `materials` specifies the material or list of materials that will be read from the TDR file specified.

The `insert` command takes the mandatory `polygon` argument that specifies the name of the polygon and inserts it into the structure. It allows you to specify the `replace.materials`, `new.material`, `replace.regions`, and `new.region` arguments in a very similar way to the insertion in three dimensions (see [Inserting Polyhedra on page 276](#)).

Inserting Polyhedra

Regions defined by polyhedra can be inserted into 3D structures. The `polyhedron` command creates a polyhedron and adds it to the internal polyhedron list.

You can build a polyhedron in different ways. However, only one of them can be used at a time in one `polyhedron` command (see [polyhedron on page 563](#)):

- Reading polyhedra from a TDR boundary file.
- Creating a cuboid (brick) polyhedron.
- Extruding a 2D polygon in the `x`-dimension.
- Creating a polyhedron from the beginning using its constituent polygonal faces `<pol1>` to `<poln>`.

Reading Polyhedra From TDR Boundary File

To read all the polyhedra included in a TDR boundary file called `<c>`, use the command:

```
polyhedron name=<c> tdr=<c> [materials= {<mat1> ...<matn>}]  
[regions= {<reg1> ...<regn>}]
```

The optional argument `materials` specifies which materials are included. In addition to explicit material names, you can use `materials=bulk.materials` to specify all nongaseous materials.

The optional argument `regions` specifies which regions of the boundary are included.

If neither `regions` nor `materials` is specified, all regions are assumed to be included. If both `regions` and `materials` are specified, the union of the two is assumed.

Several polyhedra can be included in a TDR file. Any valid TDR boundary file is allowed, regardless of the tool used to create it.

Creating a Cuboid (Brick) Polyhedron

To create a cuboid polyhedron given the coordinates of two diametrically opposite corners, use the command:

```
polyhedron name=<c> brick= {<minx> <miny> <minz> <maxx> <maxy> <maxz>}
```

Extruding 2D Polygons

To take a 2D polygon (created with the `polygon` command) and to extrude it in the x-direction from `min` to `max` to build a 3D polyhedron, use the command:

```
polyhedron name=<c> polygons= <list> min=<n> max=<n>
```

The command expects the polygon to be planar. Only one polygon name is expected in the `polygons` list.

Creating Polyhedron From Its Constituent Polygonal Faces

To build a polyhedron given its definition as a set of polygons, use the command:

```
polyhedron name=<c> polygons= {<pol1> <pol2> ... <poln>}
```

The polygons are `<pol1>` to `<poln>`. Obviously, the command expects the polygon list to form a valid polyhedron, that is, a compact, enclosed, nonintersecting 3D space. The polygons can be created with the `polygon` command.

Insertion: External Mode

This mode differs from the standard Sentaurus Structure Editor mode (see [Sentaurus Structure Editor Interface on page 277](#)) in that a structure can be created inside Sentaurus Structure Editor independently of the existing Sentaurus Interconnect structure. The minimum syntax needed for creating an external structure is:

```
sde external {<Sentaurus Structure Editor commands>}
```

where <Sentaurus Structure Editor commands> are Scheme commands that are sent directly to Sentaurus Structure Editor. As an option, a polyhedron can be specified to initialize the structure, and after `sde external`, further geometric commands such as `etch`, `deposit`, and `transform` operate on the external structure until the command `sde off` is specified. See [sde on page 606](#).

To create a polyhedron from an external structure, the option `external.sde` of the polyhedron command must be given. In the following example, an aluminum sphere is inserted into a Sentaurus Interconnect structure:

```
math coord.ucs
sde external {
  (sdegeo:create-sphere (position 0.4 0.0 0.0) 0.9 "Aluminum" "Aluminum_1")
}
polyhedron name= sphere external.sde
sde off
insert polyhedron= sphere
```

Inserting Polyhedra

The insert command is:

```
insert polyhedron=<c> [replace.materials= {<mat1> <mat2> ...<matn>}]
  [replace.regions= {<reg1> <reg2>...<reg2>}] [new.material=<c>]
  [new.region=<c>]
```

NOTE The argument `polyhedron` is mandatory and specifies the name of the polyhedron with which to operate. This polyhedron must have been previously defined with the `polyhedron` command (see [Inserting Polyhedra on page 274](#)).

The optional argument `replace.materials` lists the materials to be replaced by the polyhedron. In addition to explicit material names, you can use `replace.materials=bulk.materials`. If `replace.materials=bulk.materials` is specified, all materials in the structure, except gas, will be replaced.

The optional argument `replace.regions` lists the regions to be replaced by the polyhedron. If neither `replace.regions` nor `replace.materials` is specified, all materials are replaced. If both `replace.regions` and `replace.materials` are specified, the union of the two is assumed.

If the optional argument `new.material` is specified, all the regions in the polyhedron will change to the specified material. This argument does not change the polyhedron information except temporarily for the duration of the `insert` command. The material name in the inserted polyhedron is inserted, but not in the original polyhedron.

The optional argument `new.region` is valid only when there is one region. When set, the region name is set to the specified one after insertion. The region name in the inserted polyhedron is affected, but not the original polyhedron.

The `insert` command can perform *polyhedron etching* and *polyhedron deposition* as well as more general polyhedron insertion. Polyhedron etching is performed by specifying `new.material=gas` in the `insert` command or by creating a gas polyhedron. Polyhedron deposition is performed by specifying `replace.materials=gas` in the `insert` command as well as choosing one or more bulk regions or materials in the `polyhedron` command, such as `materials=bulk.materials` or `new.material=Silicon`.

NOTE The boundaries of the polyhedra to be inserted must not overlap any interfaces or outer boundaries of the structure. Otherwise, it is likely the operation will fail.

Sentaurus Structure Editor Interface

Sentaurus Structure Editor can perform 3D etching, deposition, and geometric transformation operations. It uses the ACIS solid geometry modeling kernel and the Scheme scripting language. Structures are created using CAD operations and process emulation operations. All 3D `etch`, `deposit`, `strip`, `photo`, `mask`, and `transform` commands are translated into appropriate Scheme commands that are then dispatched to Sentaurus Structure Editor.

Sentaurus Interconnect provides an interface to Sentaurus Structure Editor.

Sentaurus Structure Editor also can be used as a standalone tool to build the final structure using both its user interface and scripting capability. Then, the final structure can be used in Sentaurus Interconnect either as a boundary file or after remeshing the structure. The mesh or the boundary for the final structure is loaded and, before each solve step, the material of all regions, not yet present in the structure for the process step, is changed to gas.

12: Structure Generation

Sentaurus Structure Editor Interface

In addition, the Sentaurus Structure Editor external mode allows independent (that is, external) structures built in Sentaurus Structure Editor to be inserted into structures created with MGOALS (see [Insertion: External Mode on page 276](#)).

Hereafter, the standard Sentaurus Structure Editor mode (`sde on`) is referred to as the *SDE mode*.

As usual, simulations may start in one or two dimensions. If a 3D mask is encountered and if z-lines have been defined, the structure will be extruded to three dimensions, and if the SDE mode is switched on, the Sentaurus Structure Editor interface will be initialized. All subsequent structure-modifying steps in the `etch`, `deposit`, `strip`, `photo`, and `transform` commands are dispatched to Sentaurus Structure Editor.

NOTE Reading a discretized 3D structure in Sentaurus Structure Editor can be unstable. Most isotropic operations (deposition or etching) will fail if this method to initialize Sentaurus Structure Editor is used. Therefore, when initializing a 3D simulation, you should store and load `.sat` files, rather than simply loading a 3D TDR boundary or grid file.

When the 3D structure has been initialized in Sentaurus Structure Editor, structure generation commands (`mask`, `etch`, `deposit`, `photo`, `strip`, and `transform`) are translated by Sentaurus Interconnect into appropriate Scheme commands and then dispatched to Sentaurus Structure Editor.

NOTE Some arguments of the `etch` command cannot be translated into appropriate Scheme constructs: Fourier etching, trapezoidal etching, crystallographic etching, and shadowing effects are not supported in three dimensions. The argument `etchstop` only works with CMP not with other etching types.

The modified structure will be retrieved from Sentaurus Structure Editor and remeshed when a command that requires the geometry and the mesh to be synchronized (for example, `solve` and `struct` commands that write the mesh to a file) is found in the Sentaurus Interconnect command file.

This ‘lazy’ remeshing (only when needed) minimizes the number of 3D remeshing operations and, therefore, increases both the robustness and speed of the 3D structure generation and remeshing.

The `sde` command is used to configure and control the Sentaurus Structure Editor interface and to specify Scheme commands directly (see [sde on page 606](#)).

You must specify the following command in each 3D simulation so that the simulation is performed using the Sentaurus Structure Editor interface:

```
sde on
```

To switch off the SDE mode, use:

```
sde off
```

The argument `logfile` of the `sde` command specifies a file name in which to record all Scheme commands that are dispatched to Sentaurus Structure Editor. At the end of the simulation, a complete Scheme script is generated that can be used in a standalone run for debugging, testing different algorithms, or fine-tuning a few command parameters for Sentaurus Structure Editor without rerunning the Sentaurus Interconnect simulation, for example:

```
sde -l logfile.scm
```

These modified parameters and algorithm selections can later be incorporated into the `etch`, `deposit`, and other commands by specifying the argument `sde` in these commands:

```
deposit oxide thickness= 5<nm> isotropic \  
  sde= {"algorithm" "lopx" "adaptive" #t "radius" 0.075}  
  
etch silicon thickness= 0.2<um> isotropic \  
  sde= {"algorithm" "lopx" "radius" 0.07 "vexity" "convex" \  
    "blend-global" "steps" 1 "overetch" 0.2}  
  
deposit oxide thickness= 5<nm> isotropic sde= {"algorithm" "lopx"}
```

NOTE The Scheme language is incompatible with the Tcl used by Sentaurus Interconnect. Therefore, all Scheme commands and parameter settings must be enclosed by a pair of braces. The opening brace must be on the same line as the `sde` argument.

In the `sde` command, the braces can contain any number of Scheme commands, each of which starts on a new line.

Since the braces protect the Scheme commands and parameter settings from being parsed by Tcl, they must not contain any calls to Tcl procedures in Tcl expressions. The Scheme language provides its own set of expressions, parameter definitions, and other language constructs.

You should increase the default verbosity level when working with the `sde` command:

```
pdbSet InfoDefault 1
```

Sentaurus Structure Editor does not provide any error-processing facility for errors that have occurred during the solid modeling operations. This can be time consuming if a structure generation step fails and a long `solve` simulation is performed for an incorrect structure. To

12: Structure Generation

Sentaurus Topography Interface

avoid this, use a few runs with the `-f` command-line option to adjust the commands and to verify that the proper structure is created. In addition, by default, all boundary files that are written in `struct` commands in the fast mode and before remeshing are read and checked for any geometric inconsistencies. If any defects are observed, the simulation is stopped with an error. To prevent this checking, specify `!SdeCheck`.

NOTE By default, Sentaurus Interconnect performs stress relaxation at the end of each etching and deposition step. This requires that a boundary-fitted mesh be constructed at the end of each step. If you do not want to track the stress through all the process steps, use the following command before starting 3D structure generation:

```
pdbSet Mechanics EtchDepoRelax 0
```

NOTE To prevent adjacent regions of the same material (for example, gas regions) from merging, switch off region-merging using the command `pdbSet Grid No3DMerge 1`. During the process, as more regions need to be considered (for example, nitride spacer), appropriate materials must be reverted from gas to the required materials.

The option `Grid Auto3DMergeAndSeparate` (off by default) adds the following commands at the end of `photo` and `depo` when switched on:

```
(sdegeo:bool-unite (find-material-id 'depositedMaterial'))  
(sde:separate-lumps)
```

Only the second command is added after etching.

Sentaurus Topography Interface

Sentaurus Interconnect provides an interface to Sentaurus Topography, which is a 2D physical etching and deposition simulator.

Each `sptopo` command first transfers the current 2D geometry from Sentaurus Interconnect to Sentaurus Topography. Then, it dispatches the command to Sentaurus Topography. Finally, it retrieves the modified 2D geometry from Sentaurus Topography and creates a mesh for it.

Sending a new geometry from Sentaurus Interconnect to Sentaurus Topography has been restricted to cases where the geometry has actually been modified in Sentaurus Interconnect after last retrieving the structure from Sentaurus Topography, for example, when using the Sentaurus Interconnect `etch` or `deposit` command. Provisions also are made to detect whether Sentaurus Topography has actually modified the structure or simply a definition of it, for example, a new machine has been added to Sentaurus Topography. Remeshing is restricted to the commands that actually have changed the structure.

During syntax checking, Sentaurus Topography commands are dispatched to Sentaurus Topography and are checked for syntactical correctness. The syntax of the `sptopo` command is (see [sptopo on page 632](#)):

```
sptopo <Sentaurus Topography command>
sptopo <Sentaurus Topography command>
sptopo <Sentaurus Topography command>
```

or:

```
sptopo {
  <Sentaurus Topography command>
  <Sentaurus Topography command>
  ...
}
```

The first form of the `sptopo` command allows use of all the usual Sentaurus Interconnect Tcl constructions in the parameter specifications of `<Sentaurus Topography command>`. This form of the command is parsed through the Tcl interpreter. Otherwise, the syntax used for the `<Sentaurus Topography command>` is the same as in each of the commands for a standalone Sentaurus Topography run. In Sentaurus Interconnect command files, each Sentaurus Topography command must start with `sptopo`.

In the second form of the command, the pair of braces prevents the Sentaurus Topography commands from being parsed by the Tcl interpreter. No Tcl expressions must be used in this form of the `sptopo` command. On the other hand, any number of Sentaurus Topography commands can be provided in the second form of the command, each on a separate command line. If necessary, the structure will be sent from Sentaurus Interconnect to Sentaurus Topography once at the beginning, and retrieved and remeshed once at the end of the entire command sequence.

Example 1

The following command performs two planar deposition steps in Sentaurus Topography, where the first step fills the structure with oxide up to 5 nm above the top material position, and the second step adds a planar layer of 180 nm polysilicon:

```
sptopo {
  deposit material= Oxide thickness= 0.005
  deposit material= PolySilicon thickness= 0.180
}
```

The structure is sent to Sentaurus Topography once, retrieved, and remeshed once at the end of both deposition steps.

12: Structure Generation

Sentaurus Topography 3D Interface

Example 2

If masks are required in a Sentaurus Topography simulation, segments can be specified in the `sptopo` command as shown here. Alternatively, you can use the Sentaurus Interconnect `photo` command with a mask to define a photoresist layer that will protect certain areas from being etched in an `sptopo etch` command.

The Sentaurus Interconnect `strip` command, or the following `sptopo` command, can be used later to remove the entire photoresist layer:

```
sptopo etch material=Photoresist complete
```

Example 3

The following example defines a mask in Sentaurus Topography including:

- Geometric etching of polysilicon, which is strictly vertical and restricted to the outside of the specified mask segments.
- An anisotropic etching machine.
- Execution of an anisotropic oxide etching in Sentaurus Topography.

```
sptopo {  
  mask name=m1 s0=-1.1 e0=-0.3 s1=0.3 e1=1.1  
  etch material=PolySilicon depth=0.185 mask=m1  
  
  machetch name=oxe1 material=Oxide anisotropy=1 rate=1  
  etch machname=oxe1 time= 0.02 dx=0.03 dy=0.03 mask=m1  
}
```

To increase the default verbosity level when working with the `sptopo` command, use:

```
pdbSet InfoDefault 1
```

For details about the commands, parameters, and syntax rules of Sentaurus Topography, refer to the *Sentaurus™ Topography User Guide*.

Sentaurus Topography 3D Interface

Sentaurus Interconnect provides an interface to Sentaurus Topography 3D, which is a three-dimensional physical etching and deposition simulator.

The Sentaurus Topography 3D interface makes advanced etching and deposition models of Sentaurus Topography 3D available from within Sentaurus Interconnect.

The subset of 3D commands that are needed for etching and deposition is available through the interface. The `topo` command in Sentaurus Interconnect enables all the interface functionality (see [topo on page 660](#)). The `topo` command is followed by the respective Sentaurus Topography 3D commands:

```
topo {<Sentaurus Topography 3D commands>}
```

For a list of the supported Sentaurus Topography 3D commands, refer to the *Sentaurus™ Topography 3D User Guide*.

References

- [1] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, Cambridge: Cambridge University Press, 1999.

12: Structure Generation

References

CHAPTER 13 ICWBEV Plus Interface for Layout-Driven Simulations

This chapter describes how to use the IC WorkBench Edit/View Plus–TCAD Sentaurus interface.

Overview

The IC WorkBench Edit/View Plus (ICWBEV Plus)–TCAD Sentaurus interface drives the TCAD simulations from the GDSII or OASIS layout file provided by designers, which could be at any level of integration in the hierarchy: full chip, test chip, or a single cell.

The TCAD simulation domain can be conveniently chosen using specific markups in the layout file. A single process flow can be defined for all devices in the layout and can be applied easily with minimal adjustments for 1D, 2D, and 3D simulation domains. For meshing, it provides the unique feature of layout-driven meshing. Electrical contacts can be defined easily using auxiliary masks.

This chapter includes the following sections:

- [Introducing ICWBEV Plus for TCAD Users on page 286](#) provides basic ICWBEV Plus training, especially with relevance to TCAD.
- [Sentaurus Markup Files and TCAD Layout Files on page 297](#) introduces the files and file formats used in the ICWBEV Plus–TCAD Sentaurus interface.
- [Starting ICWBEV Plus in Batch Mode and Using Macros on page 304](#) introduces working with macros and running ICWBEV Plus in batch mode.
- [TCAD Layout Reader on page 305](#) presents the TCAD layout reader of Sentaurus Interconnect that provides a file-based interface between ICWBEV Plus and Sentaurus Interconnect.

Introducing ICWBEV Plus for TCAD Users

Before discussing the ICWBEV Plus–TCAD Sentaurus interface, it is important to have an understanding of ICWBEV Plus itself.

The general ICWBEV Plus training is a good starting point. Here, the focus is mainly on ICWBEV Plus operations that are most relevant to TCAD Sentaurus users.

For details, refer to the ICWBEV Plus documentation available from the **Help** menu. For all user documentation, choose **Help > Documentation**.

The first step consists of opening a layout file, which is typically in GDSII format.

Opening GDSII Layout Files

To open a GDSII or an OASIS layout file:

1. On the command line, set the environment variable `ICWBEV_USER` to activate the ICWBEV Plus Sentaurus User Mode by typing:

```
setenv ICWBEV_USER SENTAURUS
```

2. Launch ICWBEV Plus by typing:

```
icwbev
```

3. Select the file to be opened by choosing **File > Open**.
4. In the Open File dialog box, browse to the file you want to load.
5. Click **Open**.

User Interface of ICWBEV Plus

Figure 80 shows the user interface of ICWBEV Plus and illustrates the typical layout of work, panes, and toolbars. The panes can be moved and reconfigured as needed.

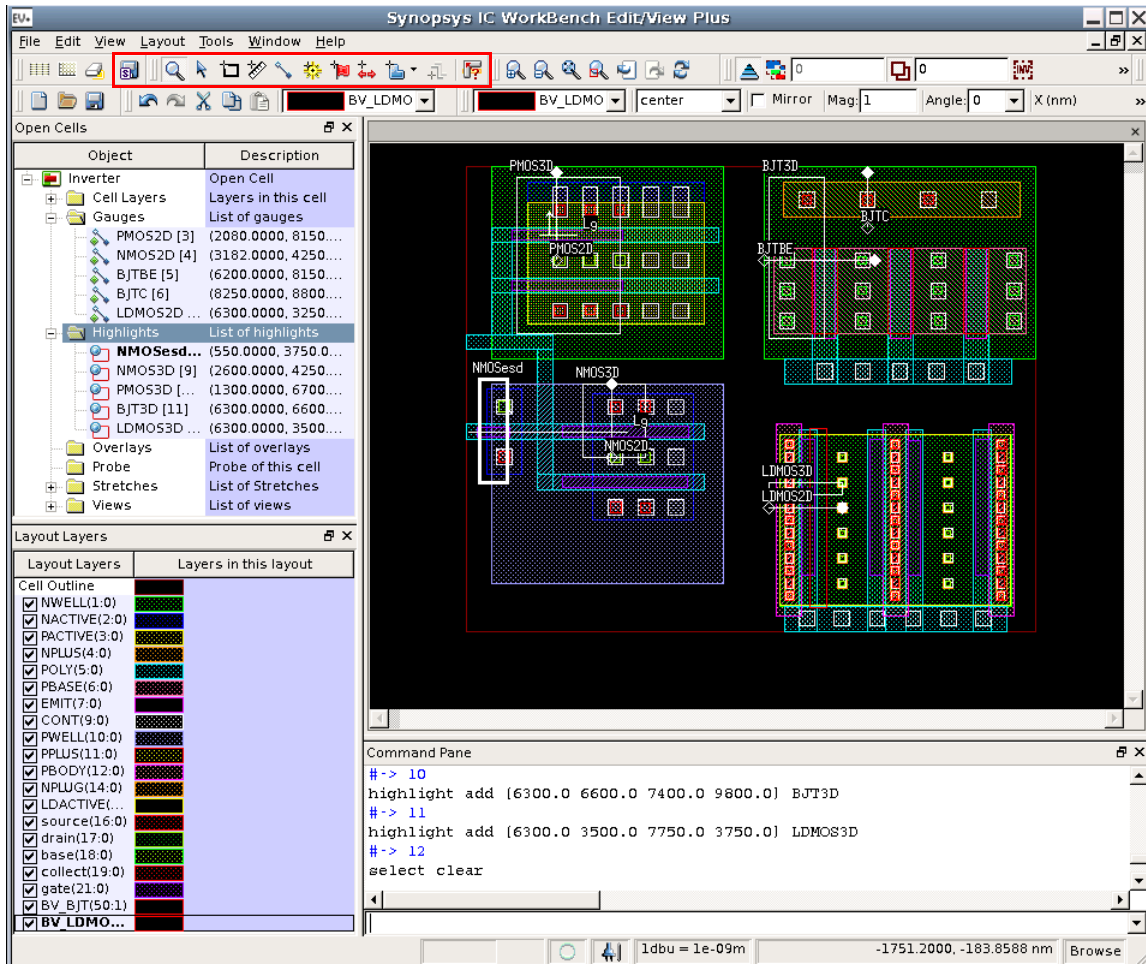


Figure 80 ICWBEV Plus main window with toolbar buttons specific to TCAD Sentaurus shown in red box at top

Figure 80 includes the following TCAD-relevant items in the GUI:






- The Open Cells pane contains details about layers and markups including the TCAD-relevant markups.
- The Layout Layers pane shows the list of layers.
- The Command Pane shows commands after GUI operations. Commands also can be entered directly in this pane.

13: ICWBEV Plus Interface for Layout-Driven Simulations

Introducing ICWBEV Plus for TCAD Users

Table 24 describes the relevant buttons of the TCAD Sentaurus toolbar.

Table 24 TCAD Sentaurus-specific toolbar buttons

Button	Description	Button	Description
	Highlight (3D simulation domain)		Stretch utility
	Gauge (2D simulation domain)		Sentaurus Export for saving TCAD layout files or markup files
	Point (1D simulation domain)		

Sentaurus Markups

Sentaurus markups are used to add the simulation domain in 1D, 2D, and 3D domains as needed. The Command Pane in Figure 81 shows the commands after adding Sentaurus markups in the layout using GUI actions.

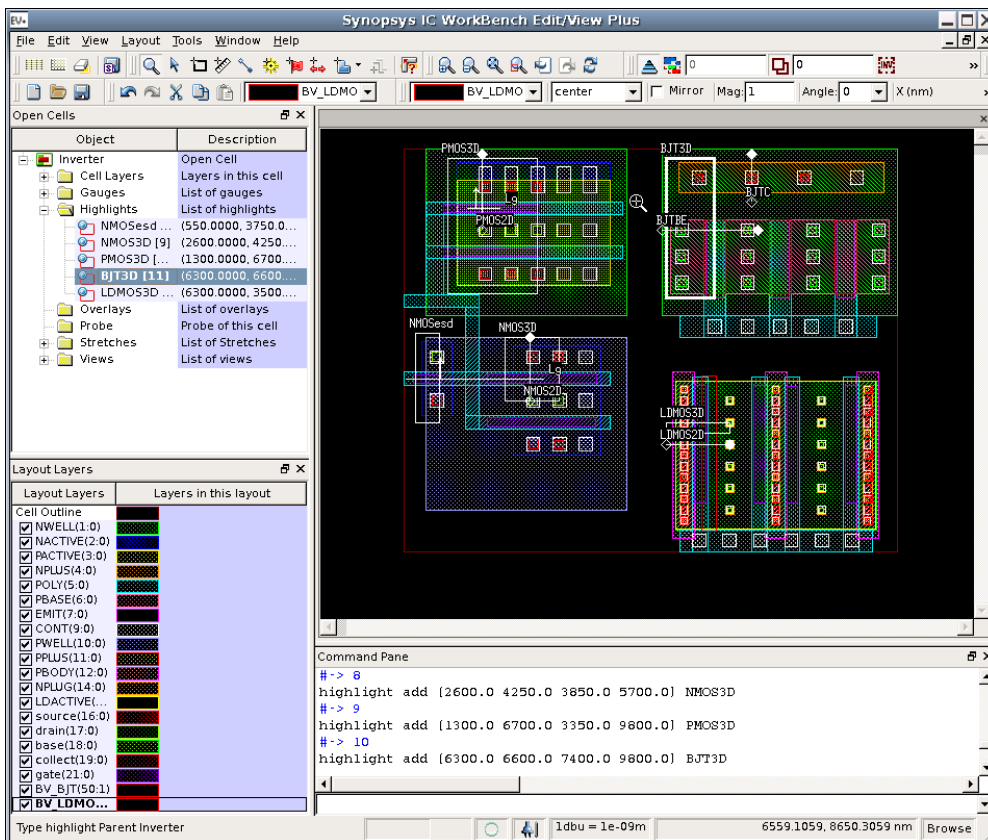


Figure 81 Adding Sentaurus markups to a layout

To add a 3D simulation domain (highlight), a 2D simulation domain (gauge), or a 1D simulation domain (point), click the respective toolbar button and draw a rectangle, a line, or a point on the layout, respectively.

NOTE A 2D simulation domain (gauge) has a direction. The starting point is given as an open diamond, and the endpoint is given as a filled diamond. A gauge that runs parallel to an edge of a layer must have a finite orthogonal distance to that edge. If a gauge is collinear with the edges of a layer, this edge might not be included in the 2D mask.

The default naming convention is the following, where <n> is an automatically incremented number:

- For a highlight: SIM3D<n>
- For a gauge: SIM2D<n>
- For a point: SIM1D<n>

For 2D TCAD simulations, it can be useful to work with composite simulation domains, for example, when the different contacts in a device layout cannot be connected by a single straight line. In this case, it is not possible to perform a 2D device simulation after a 2D process step using a single 2D simulation domain. However, a 2D TCAD simulation using a composite 2D simulation domain is feasible. In this case, the various 2D cuts in the layout are joined to form a composite 2D simulation domain.

An example of a composite simulation domain is shown in [Figure 82](#), which shows a close-up of the layout of a bipolar transistor with two 2D TCAD simulation domains. The simulation domain labeled BJTB E cuts through two base-contact fingers and one emitter finger. The simulation domain labeled BJTC cuts through the collector contact.

NOTE The two simulation domains are orthogonal and not contiguous.

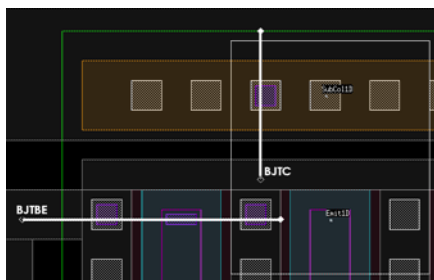


Figure 82 Layout of a bipolar transistor with two 2D simulation domains: BJTB E cuts through two base-contact fingers and one emitter finger, and BJTC cuts through the collector contact

13: ICWBEV Plus Interface for Layout-Driven Simulations

Introducing ICWBEV Plus for TCAD Users

Figure 83 shows the 2D TCAD simulation results obtained with a composite simulation domain consisting of both the BJTBE and BJTC domains. Using a composite simulation domain allows simulating a functional bipolar junction transistor (BJT) even for a 2D TCAD simulation.

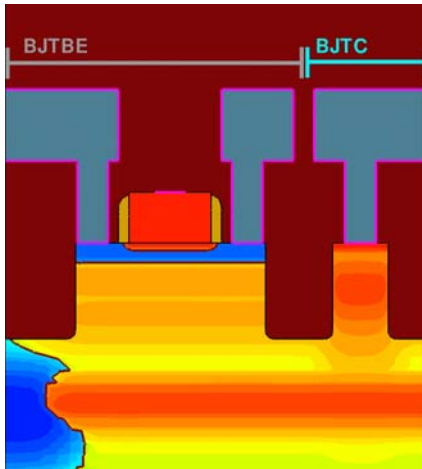


Figure 83 Two-dimensional TCAD simulation results using composite simulation domain consisting of the 2D domains BJTBE and BJTC

Stretch Utility

The stretch utility provides a convenient way to parameterize a layout by inserting a uniformly stretched segment into the layout. For example, this feature can be used to generate a set of transistors that have different gate lengths but are otherwise identical.

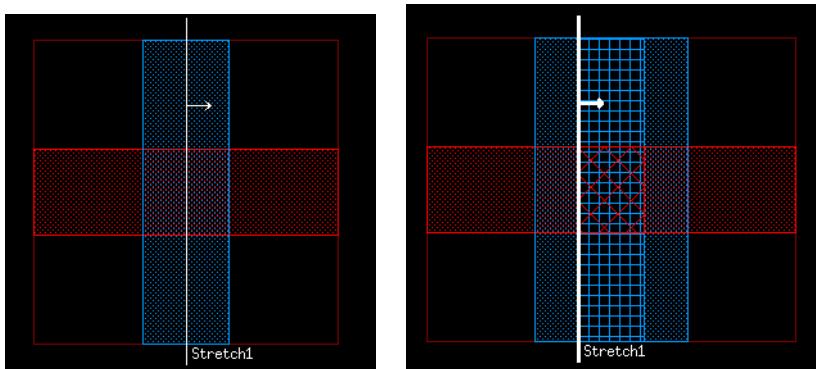


Figure 84 (Left) Snapshot of sample ICWBEV Plus layout with stretch utility line and (right) effective layout seen by Sentaurus Interconnect when the layout is loaded with a positive stretch amount

A stretch line must be defined in ICWBEV Plus first. The stretch amount is set after loading the TCAD layout with the Sentaurus Interconnect command:

```
icwb stretch name= "<stretch-name>" value= <stretch-amount>
```

This feature can be used for simple parameterization of layouts for quantities such as wire length (Joule heat). [Figure 85](#) shows a close-up of the layout containing an NMOS transistor. In addition, two stretch lines are shown. The stretch line labeled `NMOS_W` is used to vary gate width, and the one labeled `NMOS_L` is used to vary the gate length in an NMOS.

To add a stretch line:

- Click the stretch utility toolbar button, and draw a line across the required region in the layout.

NOTE The stretch line must cross the entire simulation domain to which it should be applied. Stretch lines can be used for 2D and 3D simulation domains.

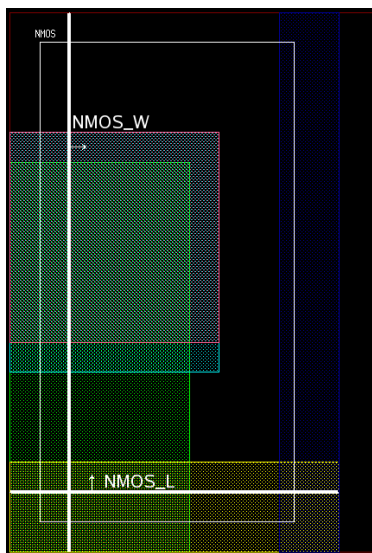


Figure 85 Adding stretch lines in a layout to vary gate width and gate length

[Figure 86 on page 292](#) shows the resulting changes in the NMOS gate width and gate length. The default naming scheme for a stretch line is `Stretch<n>`.

For example, to apply a stretch at runtime in Sentaurus Interconnect, use a command such as:

```
icwb stretch name= "NMOS_W" value= @Stretch@
```

13: ICWBEV Plus Interface for Layout-Driven Simulations

Introducing ICWBEV Plus for TCAD Users

where `NMOS_w` is the name of the stretch variable. Here, the amount of stretching is defined using the Sentaurus Workbench variable `@Stretch@`. A positive stretch value is used for expansion; a negative value leads to shrinkage.

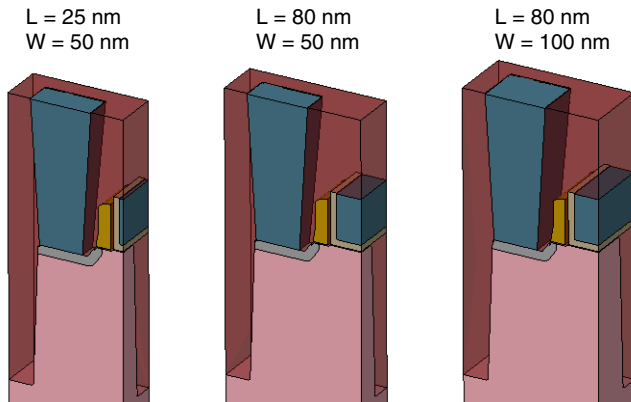


Figure 86 Effect of stretch utility on 3D NMOS structure showing variation in width and length

Renaming Markups

Markups can be renamed and edited.

To rename markups:

1. Choose **View > Views > Open Cells**.
2. Expand the markup type, for example, **Highlights**.
3. Click the respective Sentaurus Interconnect markup to edit the name.
4. Click the coordinates to edit the coordinate values.

[Figure 87 on page 293](#) shows the Open Cells pane, displaying the list of Sentaurus Interconnect markups and their coordinates that can be edited as required.

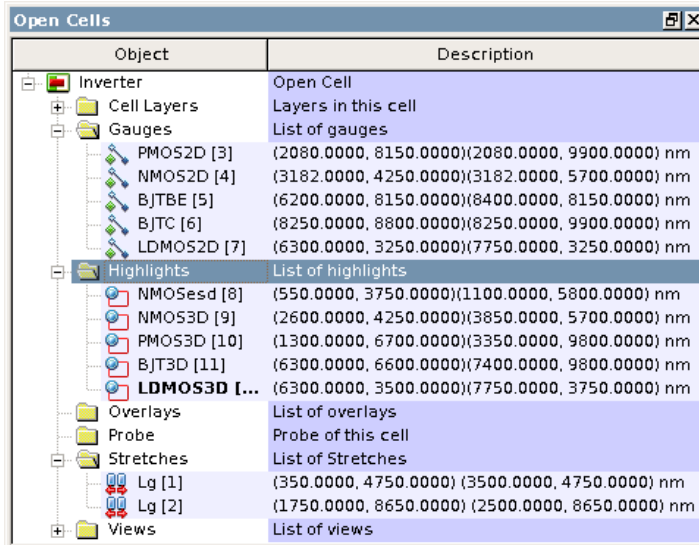


Figure 87 Open Cells pane showing list of objects with descriptions

Auxiliary Layers

Auxiliary layers are used, for example, to denote the position of electrical contacts in a layout. To add auxiliary layers, first a layer must be declared and attributes must be defined.

To add auxiliary layers, draw a polygon defining the region of the layer:

1. Open the layout for editing by choosing **Edit > Cell**.
2. Select the active layer.
3. Select the shape tool.
4. Draw a polygon.

13: ICWBEV Plus Interface for Layout-Driven Simulations

Introducing ICWBEV Plus for TCAD Users

Figure 88 illustrates how to define a layer and its attributes.

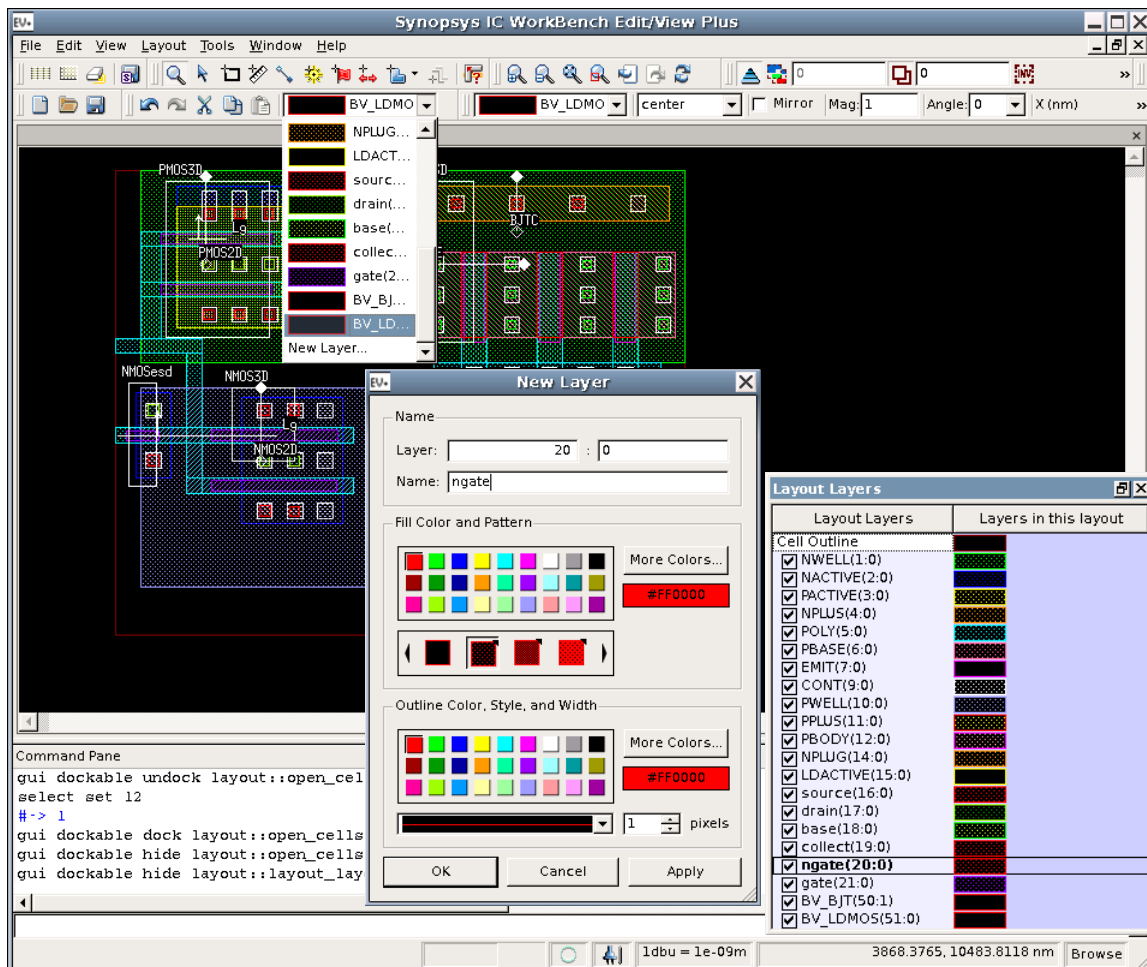


Figure 88 Defining a layer and its attributes: selecting New Layer, specifying attributes of layer, and Layout Layers panel showing new layer

Text Labels

As an alternative to auxiliary layers, you can use text labels to denote the position of electrical contacts in a layout.

To add a text label:

1. Open the layout for editing by choosing **Edit > Cell**.
2. Select the active layer.
3. Select the text-adding tool by choosing **Tools > Create > Text**.

4. In the layout window, click at the location where you want to add the text label.
5. Type the name of the text label in the entry box.

Editing Polygons

If required, polygons can be edited. You can edit polygons by either:

- Resizing a rectangle.
- Converting a rectangle to a polygon.

Resizing a Rectangle

To resize a rectangle:

1. If not already open, open the layout for editing by choosing **Edit > Cell**.
2. Activate the selector tool.
3. Click the polygon edge to select it.
4. Move the edge as needed.

[Figure 89](#) shows a rectangle highlighted for editing.

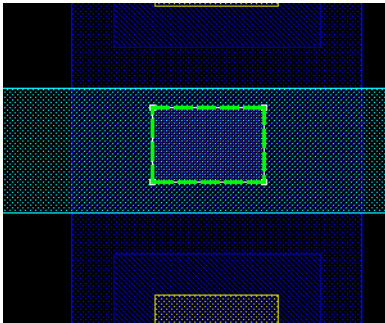


Figure 89 Moving the edge of a rectangle: select the rectangle and drag an edge

Converting a Rectangle to a Polygon

To convert a rectangle to a polygon:

1. Click the polygon edge to select it.
2. Right-click and select **Split Edge**.
3. Move the edge as needed.

Figure 90 illustrates the procedure.

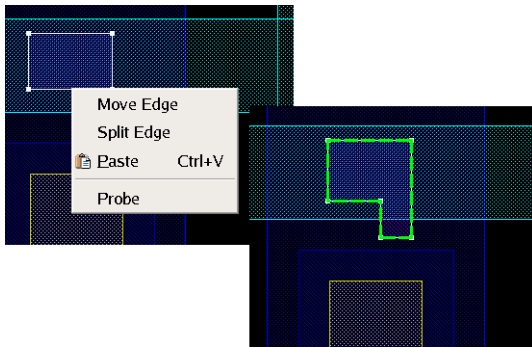


Figure 90 Converting a rectangle to a polygon

Nonaxis-Aligned Simulation Domains

The ICWBEV Plus–TCAD Sentaurus interface supports nonaxis-aligned domains. To realize nonaxis-aligned simulation domains, the GDSII layout is rotated by a given angle, and the TCAD simulation domain is added as discussed in [Sentaurus Markups on page 288](#). [Figure 91 on page 297](#) shows the transformation of a GDSII layout and the transformation parameters.

To rotate a GDSII layout:

1. Choose **Layout > Transform**.
2. In the Transform dialog box, enter the values of the fields as required.
3. Click **OK**.

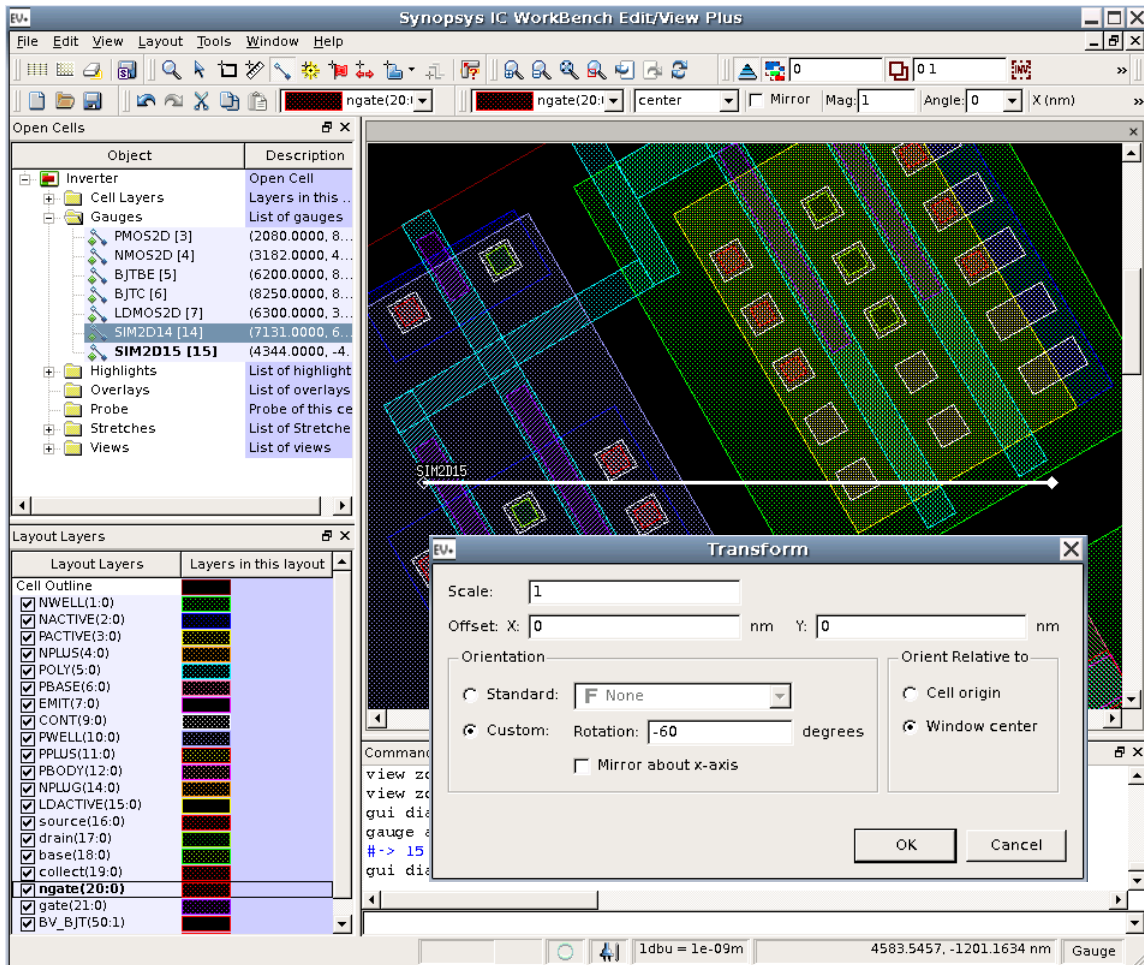


Figure 91 Rotation of layout with transformation parameters

Sentaurus Markup Files and TCAD Layout Files

After adding Sentaurus Interconnect markups in ICWBEV Plus, the markup information is saved in two different files:

- *Sentaurus markup file* (*_mkp.mac): This file format is based on the standard ICWBEV Plus macro language. It can be used to reload and re-edit Sentaurus Interconnect markup. It also contains a reference to the original, potentially large, GDSII file.
- *TCAD layout file* (*_lyt.mac): This file format is used as an internal file format for the exchange of layout information between ICWBEV Plus and TCAD Sentaurus. It is based on the standard ICWBEV Plus macro language. This file is flat and does not contain a

13: ICWBEV Plus Interface for Layout-Driven Simulations

Sentaurus Markup Files and TCAD Layout Files

reference to the GDSII file. It is a small file because it contains only the parts of the layout needed for TCAD Sentaurus.

After performing the necessary operations on the layout file, save the resulting Sentaurus markup file.

Saving the Sentaurus Markup File

To save a Sentaurus markup file:

1. Click the Sentaurus Export button (see [Table 24 on page 288](#)) or choose **Layout > Sentaurus Export**.
2. In the Sentaurus Export dialog box, select the **Markups** option (see [Figure 92 on page 299](#)).
3. If you have added auxiliary layers, which you do not want to store in the GDSII file, select the **Include new layers** option.

NOTE Including new layers in the markup file keeps the original GDSII file intact.

4. To open a specific layout file when reloading the markup file, select the **Open a layout file** option:
 - a) **Active layout:** Select this option when using a centrally located GDSII layout. This option is particularly useful when working with a very large full-chip layout.
 - b) **Layout:** Select this option when working with an edited or a local version of the GDSII layout. Specify the name of the layout.
5. Type the file name in the **Output file** field.

The recommended extension for saving the file is `_mkp.mac`, for example, `BiCMOS_mkp.mac`.

6. Click **OK**.

The corresponding script command is:

```
sentaurus export markups <name>_lyt.mac [-newLayers]
[-active | -reference <gdsfilename>]
```

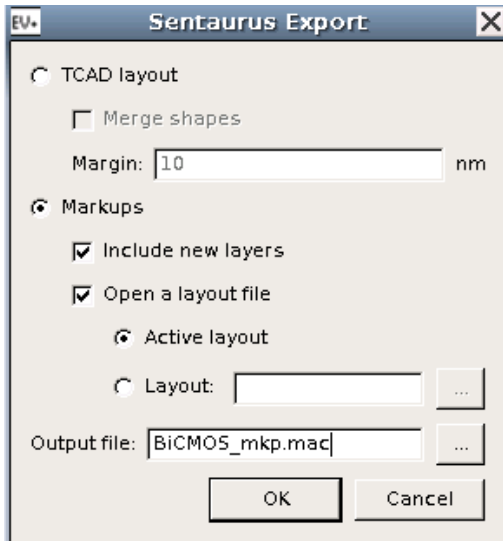



Figure 92 Sentaurus Export dialog box, showing options for saving markup file

Contents of Sentaurus Markup File

This section describes a typical markup file with a brief explanation. For a description of keywords, refer to the *IC WorkBench Edit/View Plus User Guide*.

Version information:

```
# Sentaurus markups information - Tue Jul 18 15:43:47 2017  
# Version - M-2016.12 (513586)
```

Setting for treating self-intersection: By default, all layers are ORed. This convention is expected by all subsequent tools and, therefore, this setting should not be altered.

```
default winding 1
```

Pointer to layout file:

```
layout open [<path>]/BICMOSinverter.gds Inverter
```

Global transformations:

```
cell transform 1 0 0 0 0
```

Layer declarations and display settings:

```
layer configure 1:0 -name NWELL -fill #00ff00 -pattern fill12-a  
-outline #00ff00 -lineStyle solid -lineWidth 1
```

13: ICWBEV Plus Interface for Layout-Driven Simulations

Sentaurus Markup Files and TCAD Layout Files

Open cell for editing (here, for adding new polygons):

```
cell edit_state 1
```

Auxiliary layers:

```
polygon -layer 20:0 {950 5150 950 5350 1150 5350 1150 5150}  
...
```

Simulation domains:

```
point add {8900 8100} Emit1D  
gauge add {2100 7900 2100 10000} PMOS  
highlight add {7900 7550 9900 9900} BJT3D  
...
```

Text labels:

```
cell object add text {layer 9:3 string "gate" coords {750 150}  
mag 0.3 anchor c angle 0 mirror 0}  
...
```

File end:

```
select clear  
catch {view default}
```

Reloading the Markup File

To edit a markup file, you must reload it.

To reload a markup file:

1. Choose **File > Open**.

The Open File dialog box is displayed (see [Figure 93 on page 301](#)).

2. In the **Files of type** field, select **Flags Files (*.mac)** or **Macro Files (*.mac)**.
3. Select the file required.
4. Click **Open**.

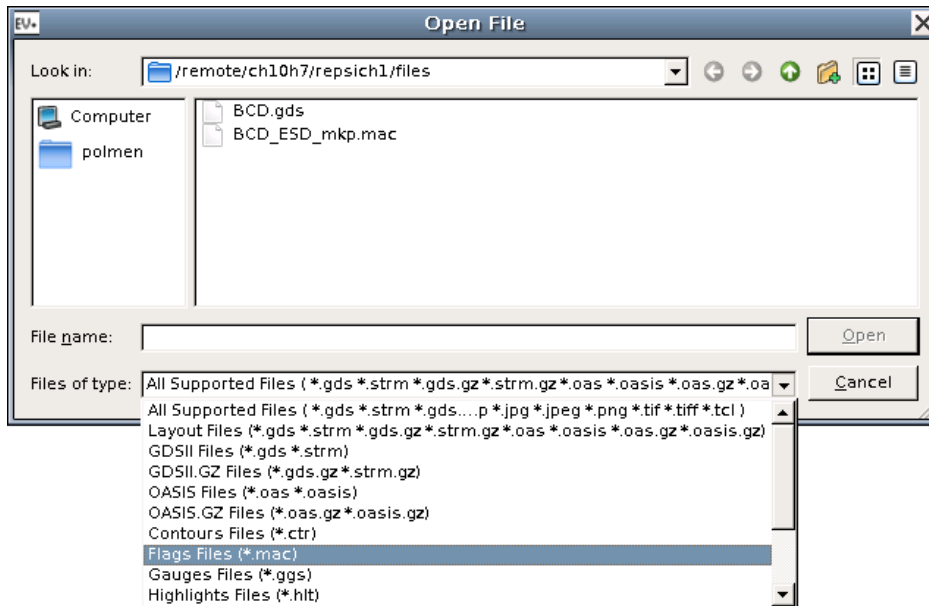


Figure 93 Open File dialog box for reloading markup file

Saving the TCAD Layout File

To save the TCAD layout file:

1. Click the Sentaurus Export button (see [Table 24 on page 288](#)) or choose **Layout > Sentaurus Export**.
2. In the Sentaurus Export dialog box, select the **TCAD layout** option (see [Figure 94 on page 302](#)).
3. A layer in a layout can be defined by a large number of touching or overlapping polygons. To merge all these polygons into a smaller number of possibly more complex polygons, select the **Merge shapes** option.
4. For better viewing, layers are padded 10 units (nm). To change the padding value, edit the **Margin** field.
5. Type the file name in the **Output file** field.

The recommended extension for the TCAD layout file is `_lyt.mac`, for example, `BiCMOS_lyt.mac`.

6. Click **OK**.

13: ICWBEV Plus Interface for Layout-Driven Simulations

Sentaurus Markup Files and TCAD Layout Files

The corresponding script command is:

```
sentaurus export layout <name>_lyt.mac [-merge | -noMerge] [-margin <number>]
```

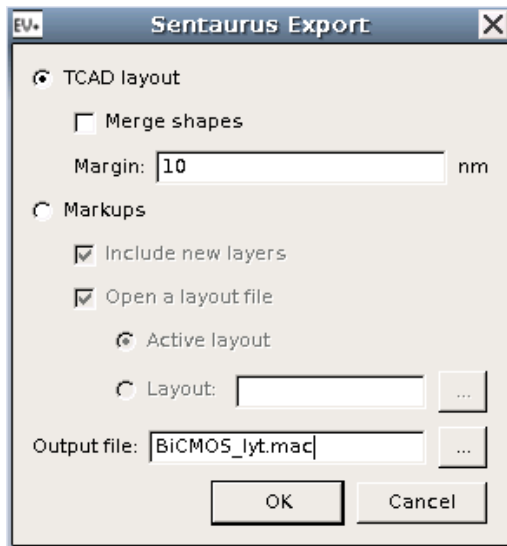


Figure 94 Sentaurus Export dialog box, showing options for saving TCAD layout file

Contents of TCAD Layout File

This section describes the contents of the TCAD layout file and the differences between the contents of the Sentaurus markup file and TCAD layout file.

Version information: Same as [Contents of Sentaurus Markup File on page 299](#).

Setting for treating self-intersection: Same as [Contents of Sentaurus Markup File](#).

Pointer to layout file: Commented out.

Global transformations: Commented out.

Initialization of this self-contained layout:

```
layout new <cell name> -dbu 1e-09
```

Layer declarations and display settings: Same as [Contents of Sentaurus Markup File](#).

File end: same as [Contents of Sentaurus Markup File](#).

Simulation domains in a layout file are described below. Point, gauge, and highlight coordinates are mentioned first, and all polygons associated with the given simulation domains are listed:

- 1D simulation domains:

```
point add {8900 8100} Emit1D
polygon -layer 1:0 {8890 8090 8910 8090 8910 8110 8890 8110}
...
```

- 2D simulation domains:

```
gauge add {2100 7900 2100 10000} PMOS
rectangle -layer 1:0 {2090 10010 2110 7890}
...
```

- 3D simulation domains:

```
highlight add {7900 7550 9900 9900} BJT3D
polygon -layer 1:0 {7890 7540 9910 7540 9910 9910 7890 9910}
...
```

- Text labels:

```
cell object add text {layer 9:3 string "gate" coords {750 150}
mag 0.3 anchor c angle 0 mirror 0}
...
```

Reloading the TCAD Layout File

For debugging purposes, reload the TCAD layout file.

To reload the file:

1. Choose **File > Open**.

The Open File dialog box is displayed (see [Figure 93 on page 301](#)).

2. In the **Files of type** field, select **Macro Files (*.mac)**.

3. Select the required file.

4. Click **Open**.

NOTE Do not extract a TCAD layout from a reloaded TCAD layout. TCAD layout files should always be extracted from the Sentaurus markup file.

Starting ICWBEV Plus in Batch Mode and Using Macros

To extract the TCAD layout file in batch mode, add the following command to the end of the markup file:

```
sentaurus export layout <name>_lyt.mac [-merge | -noMerge] [-margin <number>]
exit
```

To start ICWBEV Plus in batch mode, run the following command from the shell prompt:

```
> icwbev -nodisplay -run <name>_mkp.mac
```

ICWBEV Plus Macros

ICWBEV Plus macros can be used to create simple layouts. An example of a macro is:

```
default winding 1
layout new a -dbu 1e-09
cell transform 1.0 0.0 0 0.0 0.0
layer add 0:0
layer configure 0:0 -name {} -fill #ff0000
layer add 1:0
layer configure 1:0 -name {} -fill #ff0000
cell edit_state 1
polygon -layer 0:0 {0 0 0 100 50 100 50 0}
polygon -layer 1:0 {-25 25 75 25 75 75 -25 75}
select clear
catch {view default}
```

Tcl-Based Macros for Layout Parameterization

The macro language of ICWBEV Plus is Tcl based. [Figure 95 on page 305](#) shows a rectangle that has been replicated four times. The following Tcl command performed the replication:

```
layout new a -dbu 1e-09
layer configure 0:0 -name {}
cell edit_state 1
set SHIFTS [list 0 100 200 300 400]
foreach SHIFT $SHIFTS {
    eval rectangle -layer 0:0 { $SHIFT 0 [expr 50+$SHIFT] 50 }
}
```

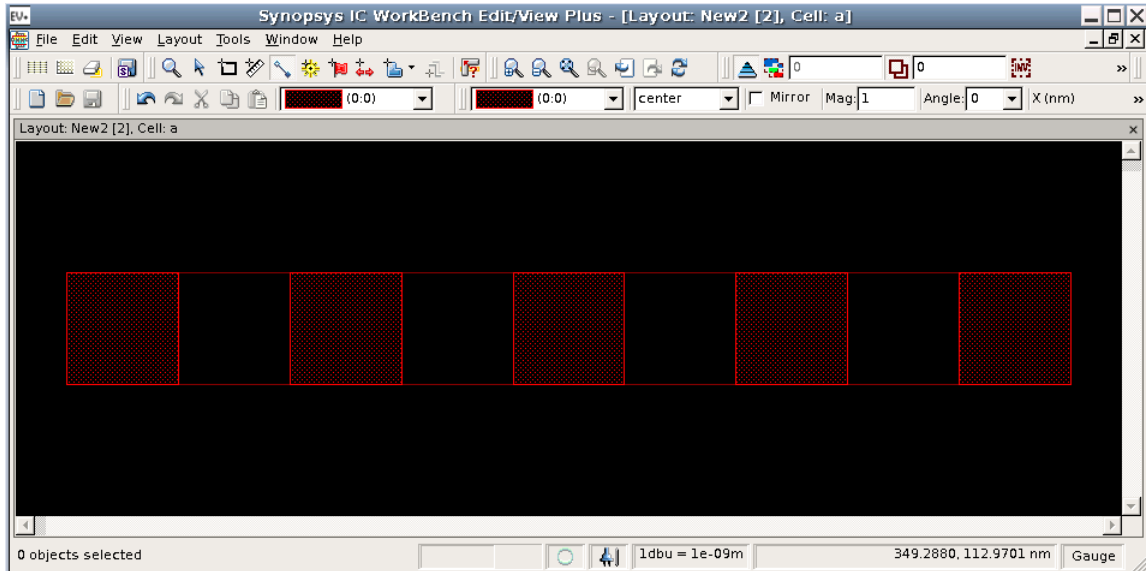


Figure 95 Shift operation using macros

TCAD Layout Reader

The TCAD layout reader of Sentaurus Interconnect provides a file-based interface between ICWBEV Plus and Sentaurus Interconnect. Some of its key features include:

- Loading the TCAD layout (optional rescaling)
- Layout query functions
- Selecting a simulation domain
- Applying stretches
- Creating masks
- Mask-driven meshing
- Mask-driven contact assignment

The following sections discuss these features in detail.

Loading the TCAD Layout

To load a TCAD layout in Sentaurus Interconnect, use the command:

```
icwb filename=<c> [scale=<n>]
```

Coordinates found in the TCAD layout file are multiplied by the value of the optional argument `scale` as the file is read.

For example, to load the TCAD layout file `BiCMOS_lyt.mac` and to apply a rescaling factor of 1/1000 to convert the ICWBEV Plus default unit of nanometer to the Sentaurus Interconnect default unit of micrometer, use:

```
icwb filename= "BiCMOS_lyt.mac" scale= 1e-3
```

Finding Simulation Domains

To generate a list of the simulation domains, use:

```
icwb list domains
```

For example:

```
set Domains [icwb list domains]  
-> icwb: Domains -> Emit1D NBODY NMOS BJT3D PMOS3D
```

Finding Layer Names and Layer IDs

Each layer in the TCAD layout file has a unique ID of the form `<integer>:<integer>`, for example `3:0`. A layer also can have an optional explicit layer name such as `NWELL`. If no explicit layer name has been set in ICWBEV Plus, the TCAD layout reader uses the layer ID as the default layer name. The TCAD layout reader refers to layers always by the layer name.

To find the layer names, use:

```
icwb list layerNames
```

For example:

```
set LNames [icwb list layerNames]  
-> icwb: LNames -> NWELL NPDIFF POLY EMIT METAL CONT ndrain ngate nsource base  
emitter collect
```


To find the layer IDs, use:

```
icwb list layerIDs
```

For example:

```
set LIDs [icwb list layerIDs]  
-> icwb: LIDs -> 1:0 2:0 3:0 4:0 5:0 6:0 7:0 8:0
```

Selecting the Simulation Domain

To select a single or a composite simulation domain, use one of the following commands:

```
icwb domain=<c> | <list-of-2d-domain-names>  
icwb domain= { <domain_name1> <domain_name2> ...<domain_namen> }
```

For example, to select a single simulation domain (which can be 1D, 2D, or 3D), use:

```
icwb domain= {PMOS}
```

To define a composite simulation in two dimensions, use:

```
icwb domain= { NBODY NMOS PMOS BJTBE BJTC }
```

Loading a GDSII Layout

To load a GDSII layout directly in Sentaurus Interconnect, use the command:

```
icwb gds.file=<c> cell=<c> layer.names= {<list>} layer.numbers= {<list>}  
sim2d | sim3d= {<n>} [domain.name=<c>] [scale= <n>]  
[stretches= {<c>= {<n>}}]
```

For example:

```
icwb gds.file= BCD.gds cell= Inverter \  
layer.numbers= "1:0 2:0 3:0 4:0 5:0 6:0 7:0 8:0" \  
layer.names= "NWELL NACTIVE PACTIVE NPLUS POLY PBASE EMIT CONT" \  
sim3d = "6300 3500 7750 3750" \  
stretches= {lgate= {7025 3400 7025 3850}} scale= 1e-3
```

The `domain.name` argument defines the name of the simulation domain. If no name is specified, `SIM3D` is used for a 3D domain, and `SIM2D` is used for a 2D domain.

The domain will be set to be the current domain automatically, so you do not need to call `icwb domain=<c>` before using other `icwb` commands.

NOTE The simulation domain and the stretches are defined using layout coordinates. This option does not require access to ICWBEV Plus. Two-dimensional composite simulation domains are supported using a variation of the `icwb` command (see [icwb.composite on page 449](#)).

Finding Domain Dimensions

To find the domain dimensions, use:

```
icwb dimension
```

This command returns 3 for 3D simulation domains (highlight), 2 for 2D simulation domains (gauge), and 1 for 1D simulation domains (point).

For example:

```
set DIM [icwb dimension]
-> icwb: dimension -> 3
```

Finding Bounding Box of Domain

To find the coordinates of the bounding box of the simulation domain in the global layout coordinates, use:

```
icwb bbox (xmin | xmax | ymin | ymax)
```

For example:

```
set LXmin [icwb bbox xmin] ; set LXmax [icwb bbox xmax]
set LYmin [icwb bbox ymin] ; set LYmax [icwb bbox ymax]
-> icwb: Layout Bounding Box -> 7.9 9.9 7.55 9.9
```

To find the coordinates of the bounding box that automatically recenters the simulation domain to start at the origin, use:

```
icwb bbox (left | right | back | front)
```

For example:

```
set Ymin [icwb bbox left] ; set Ymax [icwb bbox right]
set Zmin [icwb bbox back] ; set Zmax [icwb bbox front]
-> icwb: Centered Bounding Box -> 0 2.35 0 2
```

NOTE Sentaurus Interconnect works with the centered coordinates. The coordinates in Sentaurus Interconnect correspond to `left`, `right`, `front`, and `back`. Whereas, GDS layout coordinates correspond to `xmin`, `ymin`, `xmax`, and `ymax`. In other words, (0, 0) corresponds to (7.9 9.9) on the layout. The option `!recenter` can be given with the `gds.file` or `filename` argument to switch off recentering, which is supported in 3D simulations.

Interface With line Commands

After storing the bounding box of the simulation domain in the Tcl variables such as `Ymin`, `Ymax`, `Zmin`, and `Zmax`, these variables can be used in `line` commands to define the initial substrate and mesh in Sentaurus Interconnect. For example:

```
if { $DIM == 3 } {
    line y location= $Ymin spacing=100.0 tag=left
    line y location= $Ymax spacing=100.0 tag=right
    line z location= $Zmin spacing=100.0 tag=back
    line z location= $Zmax spacing=100.0 tag=front
    set Ydim "ylo=left yhi=right"
    set Zdim "zlo=back zhi=front"
} elseif { $DIM == 2 } {
    line y location= $Ymin spacing=100.0 tag=left
    line y location= $Ymax spacing=100.0 tag=right
    set Ydim "ylo=left yhi=right"
    set Zdim ""
} else {
    line y location=-0.5 spacing=100.0 tag=left
    line y location= 0.5 spacing=100.0 tag=right
    set Ydim "ylo=left yhi=right"
    set Zdim ""
}
eval region silicon xlo=top xhi=bottom $Ydim $Zdim
```

Creating Masks

To create a mask from a layer, use the command:

```
icwb.create.mask
    layer.name= <c> | <list>
    [name= <string>] [polarity= positive | negative]
    [save.to.tdr= 0 | 1]
    [shift= {<dy> <dz>}]
    [stretchypos= {<y0> <dy>}] [stretchyneg= {<y0> <dy>}]
```

13: ICWBEV Plus Interface for Layout-Driven Simulations

TCAD Layout Reader

```
[stretchzpos= {<z0> <dz>}] [stretchzneg= {<z0> <dz>}]  
[reflecty= <y0>] [reflectz= <z0>]
```

Masks can be created in the following ways:

- Mask name defaults to the layer name:

```
icwb.create.mask layer.name= POLY
```

- Give an explicit name to the mask. For example, to distinguish between the positive mask and negative counterparts:

```
icwb.create.mask layer.name= NWELL name= NWELL polarity= positive  
icwb.create.mask layer.name= NWELL name= NOTNWELL polarity= negative
```

Several layers can be *OR*ed to create a single mask. The following command illustrates the OR procedure:

```
icwb.create.mask layer.name= "NPDIFF PPDIFF NPLUG PBASE" name= STI info= 1
```

The `info` argument directs more detailed information about the mask creation process to the log file.

To automatically create mask layers with both polarities, use the command:

```
icwb.create.all.masks
```

The resulting mask names are `<layername>_p` for the positive version and `<layername>_n` for the negative version.

The optional arguments `shift`, `stretchypos`, `stretchyneg`, `stretchzpos`, and `stretchzneg`, `reflecty`, and `reflectz` allow you to modify individual layers during the mask generation with `icwb.create.mask`. For example, to generate a mask that corresponds to layer `1:0` shifted by $0.25\ \mu\text{m}$ along the *y*-direction and by $-0.1\ \mu\text{m}$ along the *z*-direction, use:

```
icwb.create.mask layer.name= 1:0 shift= {0.25 -0.1}
```

For 2D simulation domains, the *z*-shift can be omitted.

The arguments starting with the word `stretch` allow you to stretch individual layers in a manner similar to the `icwb.stretch` command. (The latter, however, is applied to all layers and takes the location of the stretch from the TCAD layout file.) The remaining part of the argument determines whether the stretch is applied along the *y*- or *z*-direction, and whether the layer is stretched to the positive or negative side of the stretch position. For example, to move the vertices of layer `1:0`, which have a *y*-coordinate less than 1.2 by $-0.25\ \mu\text{m}$, use:

```
icwb.create.mask layer.name= 1:0 stretchyneg= {1.2 -0.25}
```

This command operates on layer vertex coordinates and does not check if the resulting polygon is valid. When using these commands to shrink layers, you must ensure that the resulting polygons are still well defined, for example, not self intersecting.

More than one `shift`, `stretch*`, and `reflect*` argument can be used in the `icwb.create.mask` command. As these operations may not commute, you must note the order in which these operations are applied if more than one is used. First, `shift` is applied, and then `stretchypos`, `stretchyneg`, `stretchzpos`, `stretchzneg`, `reflecty`, and finally `reflectz` are applied.

NOTE This order is hard coded and not influenced by the order the arguments appear on the command line.

NOTE If you have a large layout with masks containing many polygons, it can take some time until the Tcl function `icwb.create.all.masks` parses and creates the masks. In this situation, you can use the command `icwb create.all.masks` to create all the masks (see [icwb on page 445](#)). This command works like the Tcl version but creates the masks much faster.

By default, all masks are saved into a TDR file and can be reused after reloading the TDR file in a different input file. Certain GDSII files can, however, contain layers with a very large number of points. Saving masks created from such GDSII layers in a TDR file is usually undesirable and is automatically suppressed. The default of the cut-off limit is 200 points per layer. If needed, you can increase this limit with:

```
pdbSet GDS Max.Points <n>
```

In addition, you can suppress the saving *any* mask created from GDSII layers by setting:

```
pdbSetBoolean GDS Save.To.TDR 0
```

To overwrite the default saving behavior for an individual mask, use the `save.to.tdr` argument, for example:

```
icwb.create.all.masks save.to.tdr= 0 | 1  
icwb.create.mask layer.name= L012 name= L012_n polarity= negative \  
save.to.tdr= 0 | 1
```

NOTE If you have a large layout with masks containing many polygons, it can take some time until the `icwb.create.all.masks` Tcl function parses and creates the masks. In this situation, you can use the command `icwb create.all.masks` to create all the masks (see [icwb on page 445](#)). This command works like the Tcl version but creates the masks much faster. Masks created with the `icwb create.all.masks` command are not saved to the TDR file.

Mask-Driven Meshing

To create a refinement box that is tied to a mask, use:

```
refinebox  
  [(mask=<c> extrusion.min=<n> extrusion.max=<n>) [extend=<n>]  
   ([mask.corner.mns=<n>] [mask.corner.ngr=<n>]  
    [mask.corner.refine.extent=<n>] |  
    [mask.edge.mns=<n>] [mask.edge.ngr=<n>] [mask.edge.refine.extent=<n>])]  
  [<other_arguments>]
```

Mask-driven meshing can be particularly useful when meshing in critical regions, such as the channel and emitter areas of BiCMOS devices. The following example illustrates the use of the POLY mask for meshing placement:

```
refinebox name= UnderPoly mask= POLY extend= 0.1 \  
  extrusion.min= -1.51 extrusion.max= -1.35 \  
  xrefine= 0.02 yrefine= 0.02  
  
refinebox name= SiOxPo mask= POLY extend= 0.1 \  
  extrusion.min= -1.51 extrusion.max= -1.35 min.normal.size= 0.005 \  
  interface.mat.pairs= {Silicon Oxide Silicon Polysilicon}
```

The extend argument provides the option to mesh in areas wider than the mask.

[Figure 96](#) and [Figure 97](#) on [page 313](#) demonstrate the use of the extend argument in the emitter region of a BJT and the channel region of an NMOS, respectively.

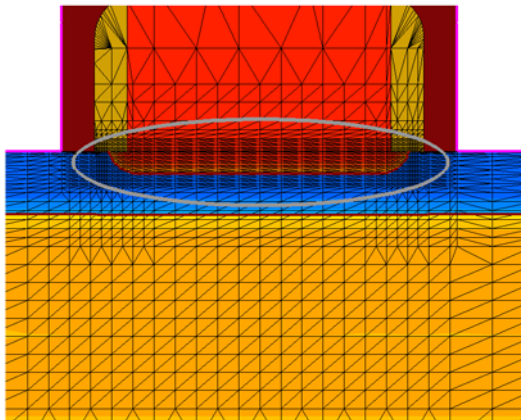


Figure 96 Meshing in emitter region of BJT; the extend parameter is set to 0.1 μm

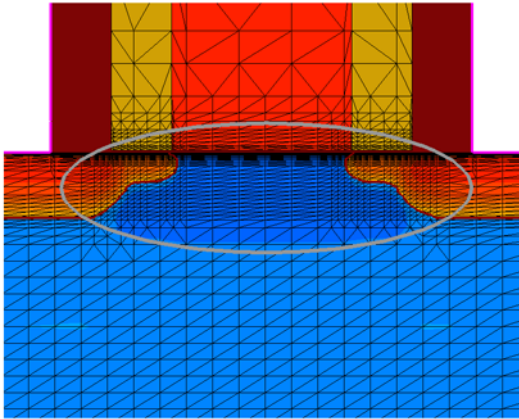


Figure 97 Meshing in channel region of NMOS; the extend parameter is set to 0.1 μm

Layout-Driven Contact Assignment

The `icwb.contact.mask` command creates contacts for subsequent device simulations that are tied to a layer or a text label in the TCAD layout file. The command serves as an interface between the TCAD layout and the Sentaurus Interconnect `contact` command by automatically obtaining the lateral placement of the contact from the specified layout layer or text label, taking the vertical placement from the argument list and passing all other options directly to the `contact` command. The syntax of the command is:

```
icwb.contact.mask
  (label.name=<c> [distance=<n>] | layer.name=<c>)
  (box <material> [adjacent.material=<c>] [boxheight=<n>] [xhi=<n>] [xlo=<n>]
  |
  point <material> [replace] [x=<n>])
  [name=<c>] [<other_arguments>]
```

The `icwb.contact.mask` command supports both box-type and point-type contacts:

- A *box-type* contact consists of elements at the surface of one region or material inside the box. The lateral extent of the box is determined automatically from the layer segment (two dimensions) or the layer polygons (three dimensions), while the vertical extent is taken from the `xlo` and `xhi` arguments of the `contact` command. If the `xlo` argument is not explicitly given, the vertical extent is determined automatically, based on the topmost interface with the material mentioned in the command call. The vertical extent of the box is controlled by `boxheight`. If a text label name is given, the layer associated with that text label is used.
- A *point-type* contact contains all the boundary elements of one region. The lateral position of the point is determined automatically as a point inside the layer segment (two dimensions) or the polygon (three dimensions), while the vertical position is taken from the

13: ICWBEV Plus Interface for Layout-Driven Simulations

TCAD Layout Reader

`x` argument of the `contact` command. If the `x` argument is not explicitly given, the vertical position is determined automatically, based on the interfaces with the material given in the command call. If a text label name is given, the anchor point of the text label is used.

Often, there is no layer or text label in the layout provided by designers that can be used readily for the creation of contacts. In this case, add auxiliary layers or text labels in ICWBEV Plus to be used as markups for device contacts.

The following example demonstrates the assignment of gate and drain contacts using layout-driven contact assignment:

```
icwb.contact.mask layer.name= ndrain name= drain point aluminum \  
  replace x= -2.0  
  
icwb.contact.mask layer.name= ngate name= gate box polysilicon \  
  adjacent.material= oxide xlo= -2.05 xhi= -1.95  
  
icwb.contact.mask label.name= "drain" point Aluminum replace
```

Any other commands not explicitly handled by the `icwb.contact.mask` command are passed to the `contact` command. The argument `name` is optional. If no `name` is given, the label name or the layer name is used as the contact name, depending on whichever has been specified.

[Figure 98 on page 315](#) shows a layout on which auxiliary layers have been added for layout-driven contact assignment. [Figure 99 on page 315](#) shows the 2D boundary after the process simulation with Sentaurus Interconnect depicting the gate, drain, and source contacts.

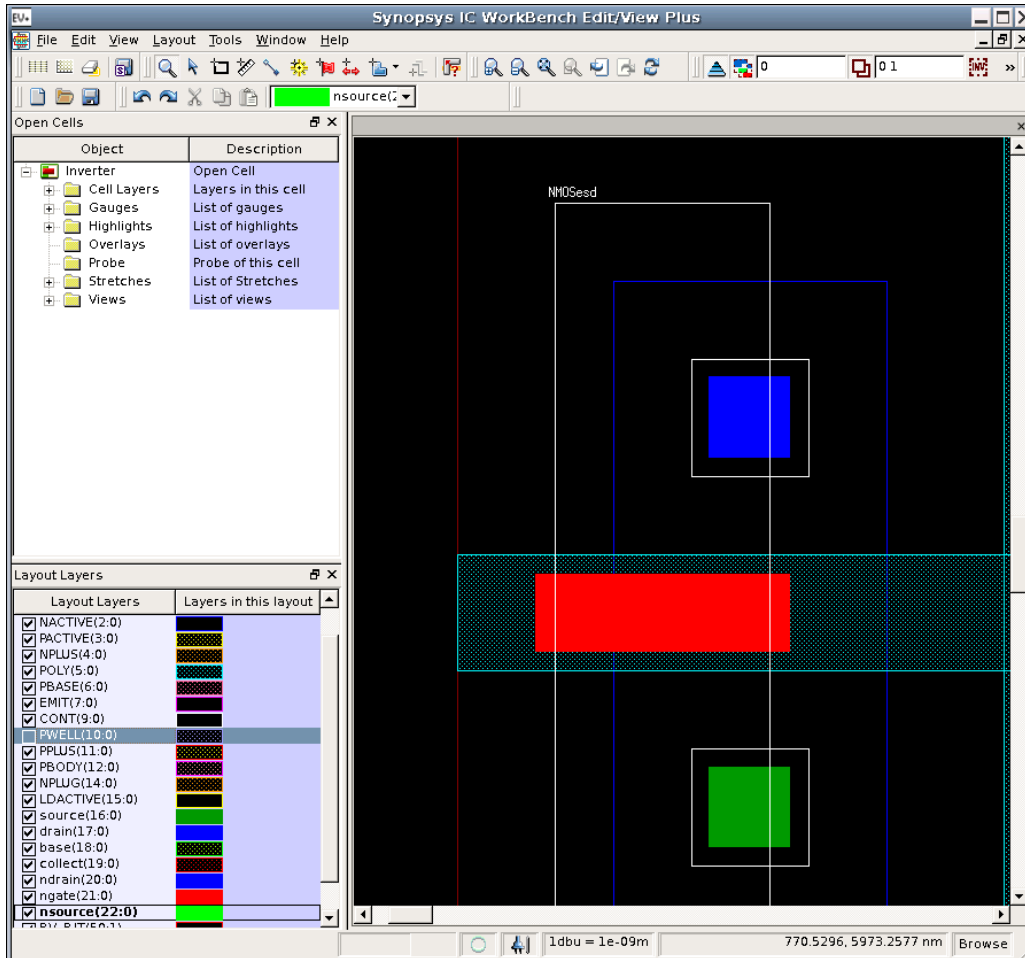


Figure 98 Auxiliary layers added for gate, source, and drain contacts are represented by rectangles of solid color in the layout

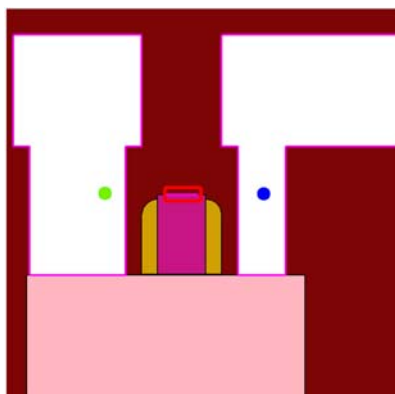


Figure 99 Final boundary after TCAD simulation showing gate (red), drain (blue), and source (green)

Aligning Wafer and Simulation Domain

To correctly support tilted process steps for 2D and 3D simulation domains, the alignment between the wafer and the simulation domain must be declared using the `slice.angle` argument of the `init` command.

The TCAD layout reader command `icwb slice.angle.offset` returns the relative angle of the active simulation domain so that the slice angle can be adjusted as needed.

Table 25 lists the returned `slice.angle` offset values for:

- A 3D simulation domain (SIM3D1).
- A 2D domain along the layout x-axis extending from left to right (SIM2DXLR) and from right to left (SIM2DXRL), and along the layout y-axis from top to bottom (SIM2DYTD) and from bottom to top (SIM2DYDT).

Table 25 Values of `slice.angle` offset for different domains

Domain	Offset value returned
SIM3D1	90
SIM2DXLR	0
SIM2DXRL	180
SIM2DYDT	90
SIM2DYTD	-90

The following commands realize a tilted process:

```
set SliceAngle -90
set SliceOffset [icwb slice.angle.offset]
init silicon field= Vacancy concentration= 1e13 \
  slice.angle= [expr $SliceAngle+$SliceOffset]
```

Additional Query Functions

The TCAD layout reader of Sentaurus Interconnect provides additional layout query functions. For example, the following command returns a list of segments in the given layer for a 2D simulation domain:

```
set Segments [icwb list.segments layer.name= "<layer-name>"]
```

For a 3D simulation domain, the following command returns a list containing the bounding boxes for all polygons in the given layer (this command also can be used for 2D):

```
set PolyBBoxes [icwb list polygon.bounding.bboxes layer.name= "<layer-name>"]
```

For a 3D simulation domain, the following command returns a list containing a tessellated representation of polygons in the given layer (this command also can be used for 2D):

```
set PolyTessel [icwb list polygon.tessellations layer.name= "<layer-name>"]
```

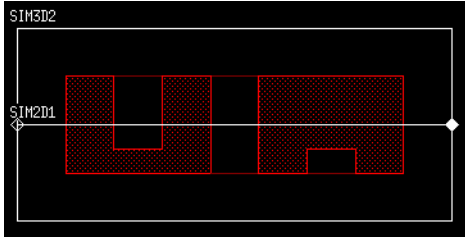


Figure 100 Sample layout containing two polygons

For example, [Figure 100](#) shows a simple layout containing two polygons in layer 0:0. The following commands:

- Load the TCAD layout file (here, called `ORG_lyt.mac`).
- Select the 2D simulation domain `SIM2D1`.
- Query the segment, the bounding boxes, and the tessellations:

```
icwb filename= "ORG_lyt.mac"
icwb domain= "SIM2D1"

set Segments [icwb list.segments layer.name= "0:0"]
LogFile "Segments $Segments"
# -> Segments 100 200 300 400 500 800

set BBoxes [icwb list polygon.bounding.bboxes layer.name= "0:0"]
LogFile "BBoxes: $BBoxes"
# -> BBoxes: {100 0 200 0} {300 0 400 0} {500 0 800 0}

set Tessellations [icwb list polygon.tessellations layer.name= "0:0"]
LogFile "Tessellations: $Tessellations"
# -> Tessellations: {100 0 200 0} {300 0 400 0} {500 0 800 0}
```

NOTE The bounding box and tessellation queries are supported for 2D, and they return flat rectangles. The returned y-values are the same as for the segment query; however, zeros are padded for the z-direction.

When loading the 3D simulation domain `SIM3D2`, the set of rectangles returned by the `polygon.bounding.bboxes` query and the `polygon.tessellations` query are different:

```
icwb filename= "ORG_lyt.mac"
```

13: ICWBEV Plus Interface for Layout-Driven Simulations

TCAD Layout Reader

```
icwb domain= "SIM3D2"

set BBoxes [icwb list polygon.bounding.bboxes layer.name= "0:0"]
LogFile "BBoxes: $BBoxes"
# -> BBoxes: {100 100 300 400} {100 500 300 800}

set Tessellations [icwb list polygon.tessellations layer.name= "0:0"]
LogFile "Tessellations: $Tessellations"
# -> Tessellations: {100 100 150 200} {100 200 150 300} {100 300 150 400}
# {100 500 150 600} {100 700 150 800} {150 100 300 200} {150 300 300 400}
# {150 500 300 600} {150 600 300 700} {150 700 300 800}
```

The `polygon.bounding.bboxes` query returns the bounding box rectangle for each polygon in the layer, while the `polygon.tessellations` query breaks each polygon into a set of rectangles and then returns these rectangles. The set of rectangles covers the same area as the original polygon, while the bounding box rectangles may cover a larger area.

[Figure 101](#) shows the rectangles returned by the two query functions as an ‘effective/ equivalent’ layout for better comparison with the original layout shown in [Figure 100](#) on [page 317](#).

NOTE The tessellation procedure supports only polygons with axis-aligned edges.

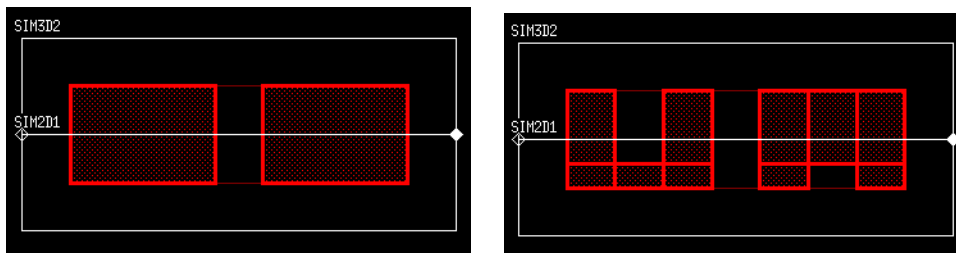


Figure 101 Set of rectangles returned by (*left*) `polygon.bounding.bboxes` query and (*right*) `polygon.tessellations` query for polygons shown in [Figure 100](#)

CHAPTER 14 Extracting Results

This chapter presents strategies for analyzing simulation results.

Overview

This chapter covers basic tasks such as obtaining a list of materials currently in the structure, and obtaining 1D solution profiles from 2D or 3D structures to more complex ones, such as looping through all materials and extracting parameters for each material. The following commands perform these tasks: `interface`, `interpolate`, `layers`, `mater`, `print.1d`, `select`, `slice`, `FitArrhenius`, and `FitLine`.

All these commands are built-in procedures designed to work with the tool command language (Tcl). These commands allow you to take full advantage of the programmability of the Sentaurus Interconnect input language and provide a powerful framework for performing complex customized tasks.

These commands return or accept a Tcl list to perform their respective functions. The Tcl list can be viewed and processed by the user, passed to another function, written to a file, or read from a file. For example, the `slice` command returns a Tcl list of xy pairs where the x value gives the depth [μm] and the y value gives the value chosen with the `select` command. This list can be viewed with the Tcl `puts` command, written to a file with the Tcl `open` and `puts` commands, or processed with another command.

An understanding of basic Tcl commands and Tcl lists is helpful to utilize fully the flexibility of these commands. For convenience, some basic aspects of Tcl are described to enable you to work efficiently with these commands, and examples of basic results analysis are provided.

Saving Data Fields

Sentaurus Interconnect automatically saves all solutions along with mechanical stress results in a TDR file. You can control the fields that will be saved in the TDR file using the `SetTDRList` command (see [SetTDRList on page 620](#)).

You can create new data fields and store them in a TDR file. The following example divides the `DataField` by 2 and saves the results in a data field called `MyField`:

```
sel z= "DataField/2" name= MyField store
```

14: Extracting Results

Selecting Fields for Viewing or Analysis

The parameter `store` ensures that the newly created field will be saved in a TDR file. To see whether a field will be saved in a TDR file, the `select` command with the option `permanent` can be used (see [select on page 609](#)). The following example returns 1 if the field will be saved in a TDR file (otherwise, it returns 0):

```
sel name= MyField permanent
```

Selecting Fields for Viewing or Analysis

Most analytic tasks begin with the `select` command, which is used to select a data field to be viewed or operated on (see [select on page 609](#)). A data field in Sentaurus Interconnect is a quantity that varies over the simulation domain, such as thermal or electrostatic potential distribution. The value of the data field is set with the `z` parameter of the `select` command.

The expression can be simply the name of a solution variable (such as `Temperature` or `Potential`) or it can be a complex expression depending on what is required. If the expression is the simple name of an existing data field, the `select` command selects this data field.

If it is more complex expression, the `select` command creates a corresponding data field and then selects it, for example:

```
select z= Potential                ;# Select Potential concentration
select z= "Potential+2*Potential"  ;# Create and select a data field using
                                   ;# solution variable
```

The list of available data fields can be retrieved by using `select list`. The name that can appear in the expression of the `z` parameter can be either a data field or a term.

A term is defined with the `term` command and is also an expression containing solution variables, data fields, constants, and so on (see [term on page 658](#)). Numerous terms are created automatically in the `solve` command (see [solve on page 629](#)), and any of these terms can be selected.

When a data field is selected (or created and selected) with the `select` command, the data field can be viewed or operated on. The following commands can operate on the selected field: `interpolate`, `layers`, `plot.1d`, `print.1d`, `print.data`, and `slice`.

Obtaining 1D Data Cuts

After a `select` command has been issued, you can obtain 1D cuts through the data along one of the principal axes using the `slice` or `print.1d` command. The `slice` command returns

a list of coordinate data pairs. To make a cut perpendicular to x , specify the x parameter, similarly for y and z .

The `print.1d` command returns a list of data-point lists. Each data-point list contains the coordinate, data value, and the material name at that coordinate. Again, a cut perpendicular to x is made by specifying the x parameter and, similarly, for the y and z cuts.

The `plot.1d` command can be used to view profiles with a temporary X11 graphics tool.

Examples

Sentaurus Interconnect can run in interactive mode if there is no command file given on the command line. In this case, you are prompted with the `sinterconnect>` prompt for commands. If a command file is given, commands are read from this file. In interactive mode, the return value of the commands is always displayed. You can set variables to the return value of a command by using the syntax:

```
set var [command]
```

In this case, `command` is executed and a Tcl variable `var` is created if it does not already exist, and the value of `var` is set to the return value of `command`. In addition, the return value of `command` is displayed. It is also possible to write the return value to a user-defined file. The following examples demonstrate the differences and functionality of the `slice` and `print.1d` commands:

```
sinterconnect> select z= MyData
sinterconnect> slice y=0.6
{-1.000000e-02 4.804720e+16
-9.340278e-03 5.869015e+16
...
0.000000e+00 5.969905e+17
0.000000e+00 7.075867e+17
7.421875e-04 7.618894e+17
...
sinterconnect> print.1d y=0.6
{ Distance      Value      Material }
{ -1.00000e-02  4.80472e+16  Oxide }
{ -9.34028e-03  5.86902e+16  Oxide }
...
{ 0.00000e+00   5.96991e+17  Oxide }
{ 0.00000e+00   7.07587e+17  Silicon }
{ 7.42188e-04   7.61889e+17  Silicon }
...
```

14: Extracting Results

Examples

Here, the `slice` command returns raw coordinate data pairs, whereas `print.1d` returns a header and coordinate data-material triplets. In both cases, the coordinates are given in micrometers and the concentration is in cm^{-3} .

To illustrate how data from these functions can be manipulated with Tcl, suppose you require a 1D profile of `MyData`, which starts with 0.0 as the first coordinate, and the `MyData` concentration to be in μm^{-3} .

First, create a Tcl list from the data returned by the `slice`, and convert data in that list to a new list, such as:

```
set myList[lindex [slice y=0.6] 0]      ;# Create a new list from slice
                                        ;# command called myList
set offset [lindex myList 0]           ;# Grab the offset, that is, the
                                        ;# first coordinate
list modList                            ;# Create new Tcl list where
                                        ;# modified data will reside

foreach { coord data } $myList {\
  lappend modList [expr $coord-$offset] ;# Convert coordinate by subtracting
                                        ;# the offset and append to modList
  lappend modList [expr $data*1.0e-12] ;# Convert data to  $\mu\text{m}^{-3}$  units and
                                        ;# append to modList
}
```

The above example uses the following Tcl commands;

- `lindex` retrieves a given element of a list.
- `list` creates a list.
- `lappend` appends an element to the end of a list.
- `expr` evaluates a math expression.
- `foreach` is used for looping.

For example, to write `modList` to a file called `xy.dat`:

```
set fileID [open xy.dat w]              ;# Use the Tcl open command
                                        ;# to open a file for writing
foreach { x y } $modList { puts $fileID "$x $y" } ;# Write modList line by line
close $fileID
```

Determining the Dose: Layers

The `layers` command computes the dose of the selected data field along one of the principal axes. The syntax to specify the cut is the same as the `slice` command (see [slice on page 623](#)). As with the other commands, the information is returned as a list of lists:

```
sinterconnect> sel z= MyData
sinterconnect> set layerInfo [layers y=0.5]; # For a 2d structure,
                                             either x or y must be specified
{
  Top          Bottom          Integral    Material}
{-2.06000e-01 -6.00000e-03  9.98843e+14 Silicon}
{-6.00000e-03  0.00000e+00  3.97970e+09e Oxide}
{ 0.00000e+00  1.00000e+00  2.81858e+05  PolySilicon}
```

The top and bottom coordinates are in micrometers. To obtain the total integrated dose along `y=0.5`, use:

```
sinterconnect> set total 0
0
# Loop over layerInfo list of lists skipping header list,
# and retrieve the 3rd element of each list (first element has 0 index)
# which corresponds to the Integral for that layer.
sinterconnect> for { set i 1 } { $i < [llength $layerInfo] } { incr i } {
> set total [expr $total + [lindex [lindex $layerInfo $i] 2]]
> }
sinterconnect> puts $total
9.991288377e+14
sinterconnect>
```

In addition to the Tcl commands used in the previous section, this example uses the following:

- `llength` returns the size of a given list.
- `incr` increases an integer by 1.

For more information about the `layers` command, see [layers on page 479](#).

Extracting Values and Level Crossings: interpolate

The `interpolate` command has two purposes: to obtain the position at which a profile crosses a particular value and to retrieve a value at a particular location in space. Interpolation is used to accomplish both tasks.

14: Extracting Results

Extracting Values During solve Step: extract

The main arguments of this command are `value`, `x`, `y`, and `z`. The combination of these arguments determines how the command operates on a selected data field:

- In 1D simulations, you must supply either `x` or `value`. If `x` is supplied, Sentaurus Interconnect returns the value at `x`. If `value` is supplied, Sentaurus Interconnect returns the locations at which the selected profile crosses `value`.
- In 2D simulations, two of these arguments must be given (not `z`).
- In 3D simulations, three of these arguments must be given.

For example, in 2D simulations, if `x` and `value` are given, the locations along `x` where `value` is crossed are returned. If `x` and `y` are given, the value at this location is returned.

For more information, see [interpolate on page 473](#).

Extracting Values During solve Step: extract

The `extract` command is used to extract historical data during `solve` steps. This command allows you to define the data extraction script with the `command` parameter. The extraction script is composed typically of the `select` command for choosing the data field for extraction and the `interpolate` command for retrieving the value at a specified location. Only values returned by the `interpolate` command, at each time step, are stored in the historical data values.

For example, to extract the potential at the position $0.04\ \mu\text{m}$ in the silicon and the `YY` component of the element stress at the position $-0.001\ \mu\text{m}$ in the oxide for each `solve` substep:

```
extract name=etest command= {
  sel z=Potential
  interpolate Silicon x=0.04
  sel z=StressEL_yy element
  interpolate Oxide x=-0.001
}
```

This command must be defined before the `solve` step. After the `solve` steps of interest, the following command retrieves the extracted data values for the defined extraction `etest`:

```
extract name=etest print
```

The values are returned as a Tcl list with the format:

```
<time1> <Potential> <Syy1> <time2> <Potential> <Syy2> ...
```

The following script demonstrates how to manipulate this list for more formatted output:

```
set extdata [extract name=etest print]
foreach { time bval sval } $extdata {
    puts "$time $bval $sval"
}
```

Output from the above script is:

```
0.000000e+00 4.738727e+15 6.000000e+09
1.000000e-04 4.738727e+15 4.793201e+09
2.503231e-04 4.738727e+15 4.793201e+09
5.509694e-04 4.738727e+15 4.793201e+09
1.152262e-03 4.738727e+15 4.793201e+09
...
```

For more information, see [extract on page 422](#).

Optimizing Parameters Automatically

Previously, users relied on the Optimizer tool in Sentaurus Workbench to perform parameter optimization. Sentaurus Interconnect provides a built-in capability for automatic optimization of parameters using the `optimize` command (see [optimize on page 522](#)).

To use this feature, a Tcl procedure must be created that takes as input the current values of the parameters to be optimized, and returns a corresponding result. Although writing the procedure is a small extra task for users, this design allows greater flexibility in the types of optimization that can be performed. The procedure can specify anything from a simple analytic function to a complete process simulation flow. The form of the result is a vector of values from which an error is computed based on user-defined target data. This generic flow allows for various applications (such as dopant profiles resulting from multiple process steps or the thickness of an oxide layer).

To use the automatic parameter optimization feature, the Tcl procedure is introduced into an input file or sourced from an external file before calling the `optimize` command.

The `optimize` command also allows you to weigh target values, to log the history of optimization steps, and to specify the maximum number of iterations and other convergence criteria.

Fitting Routines

The following commands provide fitting capabilities.

FitArrhenius Command

This command is used to find the best prefactor and energy for an Arrhenius fit of a given profile, for example:

```
sinterconnect> list dat          ;# This is the list to be passed to FitArrhenius
sinterconnect> foreach temp { 700 800 900 1000 } {
> SetTemp $temp
> lappend dat $temp             ;# dat will contain "temp" - "Arrhenius val" pairs
> lappend dat [Arrhenius 0.1 1.0];           ;# Arrhenius takes prefactor and
                                           ;# activation energy
> }
sinterconnect> FitArrhenius $dat          ;# Send the list to FitArrhenius
0.0999999308634 1.00030363776 -0.999999999866 ;# Return prefactor, energy
                                           ;# and corr factor
sinterconnect>
```

This command takes a list of temperature–function pairs. The unit of temperature is degree Celsius. The return value is a list where the first member is the prefactor, the second member is the activation energy [eV], and the third member is the correlation factor. Absolute values of the correlation factor close to one are desirable.

FitLine Command

This command is used to find the best offset and slope for a given set of data, for example:

```
sinterconnect> foreach temp { 700 800 900 1000 } {
> lappend dat $temp
> lappend dat [expr 110 + 10*$temp]
> }
sinterconnect> FitLine $dat             ;# Get the slope, offset, and correlation
factor
10.0 110.0 1.0
```

Storing Time Versus Resistance, Current, and Voltage

The `option` command allows you to store time versus resistance between two contacts, or current at a contact, or voltage at a contact for a time-dependent simulation, for example:

```
option resistance= { {upper lower} }
option current= {upper lower} voltage= {upper lower}
```

The first command stores the resistance between the contacts `upper` and `lower`. The second command stores the current and voltage at the same contacts versus time. More than one contact name can be given in a list for the `current` and `voltage` options. For `resistance`, a list of contact pairs must be given. For resistance information of multiple pairs of contacts, use:

```
option resistance= { {upper lower} {c1 c2} }
```

When the time-dependent simulation is completed, the result can be written to a `.plx` file using the `WritePlx` command (see [WritePlx on page 681](#)).

```
WritePlx TimeVsR resistance "upper lower"
WritePlx TimeVsI current "upper lower"
WritePlx TimeVsV voltage "upper lower"
```

The first command creates a file called `TimeVsR.plx` and writes time-versus-resistance values for the contacts `upper` and `lower`. The second command creates a file called `TimeVsI.plx` and writes time-versus-current values for the contacts `upper` and `lower`. The third command creates a file called `TimeVsV.plx` and writes time-versus-voltage values for the contacts `upper` and `lower`.

Optionally, `file.name` can be used in the `option` command to store the results in a file:

```
option resistance= { {upper lower} } file.name= TimeVsR.plx
```

The `option` command also can be used to store circuit information. For circuit simulations, it can store time-versus-total current between a circuit element and a node, and time-versus-total voltage at a node, for example:

```
option current= { {r1 left} {V2 grnd} }
option voltage= {left grnd}
```

The first command stores time versus total current between the pairs, circuit element `r1` and node `left`, and also between circuit element `V2` and node `grnd`. The second command stores the time-versus-voltage values for the circuit nodes `left` and `grnd`. To write the results to a `.plx` file, use `WritePlx`:

```
WritePlx TimeVsI current "r1 left"
WritePlx TimeVsV voltage "left"
```

14: Extracting Results

Storing Time Versus Resistance, Current, and Voltage

The first command creates a `TimeVsI.plx` file and writes the time-versus-current between the circuit element `r1` and circuit node `left`. The second command creates a `TimeVsV.plx` file and writes the time-versus-voltage values for the circuit node `left`.

NOTE When time-versus-variable files are written, they begin at time $t=0$. For mixed-mode simulations, `isolve` is switched on by default in the `solve` command and, therefore, a steady-state solution at $t=0$ is obtained, which is written to the file. However, when performing structure-only simulations, `isolve` is not switched on by default and, therefore, the solution at $t=0$ is the initial guess supplied to the simulation. If a more accurate representation of the starting point is needed using an initial steady-state solution, `isolve` must be switched on. However, consider that this could be computationally expensive.

This chapter discusses numerics-related issues, time integration methods, and the linear solvers used in Sentaurus Interconnect.

Overview

In Sentaurus Interconnect, during the simulation of solve steps, different sets of nonlinear partial differential equations must be solved:

- Thermal
- Electrical
- Stress equations

These equations are solved on the simulation mesh using a trapezoidal rule/backward differentiation formula (TRBDF) time discretization, a finite volume (box) method for the spatial integration, and a Newton method to solve the nonlinear equations.

For the discretization of the nonlinear stress equations, piecewise linear finite elements are used. If stress history is tracked, the stress equations are solved, not only during the simulation of thermal and electrical solution steps, but also at the end of etch and deposit steps.

Various direct and iterative solvers are integrated in Sentaurus Interconnect to solve the large systems of linear equations in each Newton iteration. By default, for all equations in 1D simulations and for mechanics equations in 2D simulations, the parallel direct solver PARDISO is used. For diffusion equations in two dimensions and for all equations in three dimensions, the iterative solver ILS is used. The solver can be selected using the `math` command:

```
math ils
```

This command selects the solver ILS for all types of equation in 1D, 2D, and 3D. Separate selections can be made for the various spatial dimensions and for the solution of mechanics, thermal, and electrical equations.

The parameters `Flow` and `compute` select the type of equation, and the parameter `dim` specifies the spatial dimension:

```
math Flow    dim=2 ils
math compute dim=2 pardiso
math Flow    dim=3 pardiso
```

15: Numerics

Setting Parameters of the Iterative Solver ILS

If a direct solver is used, a modified Newton method is used by default; Sentaurus Interconnect tries to avoid the recomputation and factorization of a new matrix and will reuse the last factorized matrix, as long as the convergence rate remains sufficiently high. For the iterative solvers, by default, a modified Newton scheme is used as well.

The `math` command is used to specify various parameters for the Newton iterations and to define resources and specifications for the linear solvers (see [math](#) on page 501).

For the default settings for ILS, refer to the Parameter Database (PDB). More detailed settings for ILS can be made using `pdbSet` commands as described in the next section.

Setting Parameters of the Iterative Solver ILS

The iterative solver ILS is used by default to solve the linear systems for mechanics, thermal, and electrical analyses in 3D simulations. Default parameters for ILS have been added to the parameter database. To specify modified parameters for ILS, such as the type of iterative scheme, the number of iterations, the output verbosity, or the memory resources, use the `pdbSet` commands.

NOTE In the `pdbSet` commands, parameters must be specified separately for each type of problem (Flow or compute) and for each dimension (1D, 2D, or 3D).

NOTE ILS is not recommended for use in 1D simulations because of the simple structure of matrices arising in 1D cases. The default direct solver PARDISO is the correct choice for 1D simulations.

Different ILS parameters can be specified both in 2D or 3D. In general, the `pdbSet` command for the ILS parameters has the form:

```
pdbSet Math [compute | Flow] [2D | 3D] ILS.[command] [value]
```

The following ILS commands are available:

<code>ILS.compact</code>	Boolean
<code>ILS.fgmres.restart</code>	Double
<code>ILS.gmres.restart</code>	Double
<code>ILS.fit</code>	Double
<code>ILS.ilut.tau</code>	Double
<code>ILS.leftPreconditioner</code>	Boolean
<code>ILS.maxit</code>	Double
<code>ILS.method</code>	String
<code>ILS.nonsymmOrdering</code>	String
<code>ILS.okayForModNewton</code>	Boolean
<code>ILS.preconditioner</code>	String

ILS.recompute.ordering	Double
ILS.refine.residual	Double
ILS.refine.iterate	Double
ILS.scaling	String
ILS.symmOrdering	String
ILS.tolabs	Double
ILS.tolrel	Double
ILS.tolunprec	Double
ILS.useILSRCFile	Boolean
ILS.verbose	Double

To select the GMRES method, for example, `gmres(60)`, use:

```
pdbSet Math compute 3D ILS.method          gmres
pdbSet Math compute 3D ILS.gmres.restart 60
```

To specify the FlexibleGMRES method, for example, `fgmres(40)`, to solve the stress equations, use:

```
pdbSet Math Flow 3D ILS.method          fgmres
pdbSet Math Flow 3D ILS.fgmres.restart 40
pdbSet Math Flow 3D ILS.fit             5
```

To select the efficient reuse mode (on each 3D solution time step, the costly reordering is applied only once to a first Jacobian system), specify:

```
pdbSet Math compute 3D ILS.recompute.ordering 2
```

and to return to reordering for every system, use:

```
pdbSet Math compute 3D ILS.recompute.ordering 1
```

To improve the accuracy and convergence of iterative linear solvers, use an enhanced option by specifying (default value is 0):

```
pdbSet Math [compute | Flow] [2D | 3D] ILS.refine.iterate 1
```

To select the solvers for mechanics, STS2 or STCG2 for 2D, and STS3 or STCG3 for 3D, for example, `sts3`, use:

```
pdbSet Math Flow 3D ILS.method          sts3
pdbSet Math Flow 3D ILS.tolrel          1e-10
pdbSet Math Flow 3D ILS.ilut.tau        5e-4
pdbSet Math Flow 3D ILS.scaling          diagsym
pdbSet Math Flow 3D ILS.nonsymmOrdering none
```

For the mechanics solvers STS2, STS3, STCG2, and STCG3, it is mandatory to specify `ILS.scaling` as `diagsym`, and `ILS.nonsymmOrdering` as `none`. It is also recommended to specify the value for the parameter `ILS.ilut.tau` in the range of 5×10^{-4} – 5×10^{-5} .

15: Numerics

Partitioning and Parallel Matrix Assembly

To improve convergence of the mechanics solver STS2 or STS3, use an enhanced version of the solver by specifying, respectively (default value is 0):

```
pdbSet Math Flow 2D ILS.refine.sts 1
```

or:

```
pdbSet Math Flow 3D ILS.refine.sts 1
```

The enhanced version takes advantage of results from previous solve steps, so the actual performance gain can vary depending on the simulation setup, and it performs best when there is a sequence of mechanical solve steps, as in temperature ramps.

To active the STS solver which improves robustness and speed, use:

```
pdbSet Math Flow 3D ILS.refine.sts 2
```

Partitioning and Parallel Matrix Assembly

Sentaurus Interconnect can assemble the solution matrix in parallel on multicore machines. To switch on the parallel assembly, use:

```
math numThreads=<i> | numThreadsAssembly=<i>
```

where `numThreads` is the number of threads that would be used during the matrix assembly. `numThreads` is a general keyword (see [math on page 501](#)) used by linear solvers. If you want to use a different number of threads for matrix assembly, use the keyword `numThreadsAssembly`. If the number of threads is greater than 1, Sentaurus Interconnect first creates the threads.

To modify the thread stack size, use:

```
math threadStackSize=<i>
```

NOTE It is recommended that `numThreadsAssembly` equals the number of actual CPUs that the computer contains. Parallel assembly of the matrix is not recommended for moving-boundary problems.

Sentaurus Interconnect then partitions the mesh structure into levels, and each level is divided into different domains at the beginning of the solve step. For example, [Figure 102 on page 333](#) shows a structure with three levels. The first level (blue) (L0) has four domains: D0, D1, D2, and D3. Elements belonging to each domain on the same level do not cross over to the other domains. The second level (orange) (L1) also has four domains: D0, D1, D2, and D3. Again, the elements on the same level do not cross over to the other domains. The third level (green)

(L2) has only one domain: D0. This is the last level and contains all the elements not included in the previous levels.

To control the partitioning, use:

```
math maxNumberOfDomains=<i> | NumberOfElementsPerDomain=<i>
```

NOTE The `maxNumberOfDomains` argument is the maximum number of domains that each level of partition can have. It is recommended that `maxNumberOfDomains` is equal to or greater than the number of threads used. The argument `NumberOfElementsPerDomain` is the number of elements that should go to each domain.

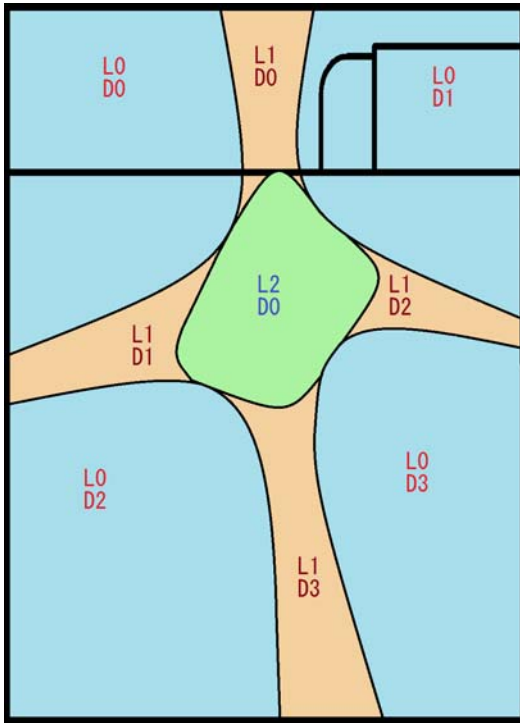


Figure 102 Partitioned mesh structure

The final number of domains at each level is determined by:

$$domains = \min(maxNumberOfDomains, \frac{Number\ of\ Edges}{NumberOfElementsPerDomain}) \quad (117)$$

To partition the mesh, based on material type, give weight to each material using the command:

```
pdbSet <material> PartitionWeight <n>
```

15: Numerics

Matrix Size Manipulation

For example:

```
pdbSet Silicon PartitionWeight 10
pdbSet Gas PartitionWeight 0
```

gives more weight to silicon mesh elements than the gas mesh elements during partitioning. This allows Sentaurus Interconnect to distribute the work among the threads more evenly since there is no matrix assembly for the gas mesh.

Partition weights for mechanics assembly can be specified separately with:

```
pdbSet <material> Mechanics PartitionWeight <n>
```

This allows balancing the workload among threads according to the stress analysis methods for different material behaviors. If the partition weights for mechanics assembly are not defined, the partition weights for PDE assembly are used by default.

Matrix Size Manipulation

The size of the matrix used during PDE assembly is automatically determined based on the number of solution variables and nodes in the structure. In most cases the allocated matrix size is more than sufficient. If the matrix size becomes insufficient during the assembly, the matrix size will be increased automatically by 10%. You can change the default 10% value by using the command:

```
pdbSet Math Matrix.Size.Scale <n>
```

where the value <n> should be greater than one.

You also can increase the automatically determined matrix size using the command:

```
pdbSet Math Assembly.Matrix.Size.Scale <n>
```

where the value <n> should be greater than one.

NOTE Be careful when choosing the matrix scaling values because it can exhaust the computer memory for large scaling values.

Node and Equation Ordering

Because the order of nodes in meshes does not follow a specific order by default, adjacent nodes may be far from each other in the internal node list. The order may not have much effect

on simulation time for small examples (such as 2D), but it may degrade 3D results. The nodes in the structure can be ordered meshwise or globally using the command:

```
pdbSet Math <1D | 2D | 3D> Reorder.Nodes <model>
```

where <model> is None (default), Mesh, or Global.

The default order of equation numbering in the structure is based on the meshes. Each node in the mesh receives an equation number from a solution variable and the same is repeated for the next solution. This may create many distributed entries in the assembly matrix. Again, the order may not greatly affect the simulation time for small examples (such as 2D), but it may degrade 3D results. It is possible to number equations based on solutions by taking a node in the mesh, numbering it for each solution variable, and moving to the next node in the mesh. This creates better-distributed entries in the assembly matrix. The order can be changed using the command:

```
pdbSet Math <1D | 2D | 3D> Reorder.Equations <model>
```

where <model> is None (default) or Solution.

Time Integration

The TRBDF method [1] is used for time integration by default for time-dependent problems. It also is possible to choose the backward Euler method for the time integration. The following command can be used to switch between methods:

```
math [tr_bdf | euler]
```

A TRBDF integration step consists of a trapezoidal step followed by a backward difference step. A second trapezoidal solution is used to estimate the local truncation error and to determine the size of the next time step.

The local truncation error can be estimated by either a Milne's device (the default method) or the divided difference method. The following command switches between methods:

```
math [milne | difference]
```

The local truncation error for the next time-step estimation can be modified using the command:

```
pdbSet Math Time.Step.Function {<model>}
```

where <model> is Damped, UnDamped, or Linear. The Damped model applies a logarithmic damping function to the truncation error if the error is greater than 1.0. The UnDamped model does not modify the error. The Linear model applies a linear damping function to the

truncation error if the error is greater than 1.0. The `Linear` model matches that of `TSUPREM-4`.

When the geometry of a simulation structure evolves, one cycle of the TRBDF time integration requires the geometric coefficients at three incidents, that is, $t = t_0, t_0 + \Delta t_{TR}, t_0 + \Delta t$. To reduce the computational time to calculate the geometric coefficients, especially in three dimensions, the geometric coefficients at $t = t_0 + \Delta t_{TR}$ can be set to the interpolated values by assuming that the coefficients change linearly during Δt , which reduces the number of the box method calls by one third:

```
pdbSet Math 3D Use.Interpolated.Geom.Coeff 1
```

or:

```
math dimension=3 use.interpolated.geom.coeff
```

Time-Step Control

This section discussed different time-step controls.

Time-Step Control for PDEs

Sentaurus Interconnect provides automatic time-step control. You can modify some of the control parameters.

The first time step of the `solve` command uses the initial time given with the `solve` command (see [solve on page 629](#)).

In an ideal situation, mechanics and PDE solve time steps are equal to each other, and the next time step is increased by the `IncreaseRatio`:

$$t_{n+1} = I_{\text{ratio}} t_n \quad (118)$$

where t_{n+1} is the next time step, t_n is the current time step, and I_{ratio} is the `IncreaseRatio`. The default for `IncreaseRatio` is 2.

Use the following command to change `IncreaseRatio`:

```
pdbSet Compute IncreaseRatio {<n>}
```

In some cases, the ideal time step can be solution limited or reduced:

- *Solution limited* is the case when the time step is shortened to decrease the local truncation error; in a log file, such steps are marked by (s).

- *Reduced* is the case when the time step is reduced to prevent overstepping of mechanics steps; in log files, such steps are marked by (r).

If convergence is not achieved, the next time step is reduced by the `ReduceRatio`:

$$\tilde{t}_{n+1} = R_{\text{ratio}} t_{n+1} \quad (119)$$

The following command can be used to change `ReduceRatio`:

```
pdbSet Compute ReduceRatio {<n>}
```

For more information about the convergence during solve steps, use the command:

```
pdbSet Compute Convergence.Info <1 | 0>
```

Typical output with information `level=2` will look like:

```
Iter Lin Potential Vacancy
  1   8 1.526e-03 1.007e+01
-----
Mesh: Copper_1      Mater: Copper
      Org. Val. Org. Updt. Org.-Updt Apld. Updt. Error Location
Vac   3.5e+14  5.5e+13   2.9e+14  2.9e+14  1.5e+03 (4.82e-01 1.0e+00 1.00e+00) (UCS)
                                           (1.00e+00 1.0e+00 -4.82e-01) (DFISE)
Largest update: Vac in Silicon @ (5.01e-01 1.00e+00 1.00e+00) (UCS)
                                           (1.00e+00 1.00e+00 -5.01e-01) (DFISE)
```

where:

- `Org. Val` is the original value at the node.
- `Org. Updt` is the original update at the node.
- `Org. -Updt` is the Original Value – Original Update.
- `Apld. Updt.` is the applied update.
- `Location` is the location (in μm) of the node in both the unified coordinate system (UCS) and the DF-ISE coordinate system.

Different time-step control models are available:

- Two history-based models involving all previous time steps within the `solve` command: BPTS and NGLTS.

BPTS uses the *biggest previous time step* from the history, such that [Eq. 118](#) is modified as:

$$t_{n+1} = I_{\text{ratio}} t_{\text{BPTS}, n} \quad (120)$$

NGLTS uses the latest *nongrid limited time step*, and [Eq. 118](#) is modified as:

$$t_{n+1} = I_{\text{ratio}} t_{\text{NGLTS}, n} \quad (121)$$

Use the following command to choose the time-step control model:

```
pdbSet Compute TSEM <model>
```

where <model> can be None, BPTS, or NGLTS.

Error Control for PDEs

To control errors during transient simulation, Sentaurus Interconnect uses the following to calculate the error:

$$e = \left(\frac{1}{N} \sum_{i=1}^N \left(\frac{|u_i|}{|sol_i| \times \text{TransRelErr} + \text{AbsErr}} \right)^2 \right)^{1/2} \quad (122)$$

where the sum is taken over all solution variables, and u_i is the update for solution sol_i . TransRelErr and AbsErr are the transient relative error and absolute error for the solution variables, respectively. They can be set using the commands:

```
pdbSetDouble <mater> <solution> Transient.Rel.Error <n>  
pdbSetDouble <mater> <solution> Abs.Error <n>
```

where <mater> is the material name.

To control errors during nonlinear Newton iterations, Sentaurus Interconnect uses the following to calculate the error:

$$e = \left(\frac{1}{N} \sum_{i=1}^N \left(\frac{|u_i|}{|sol_i| \times \text{RelErr} + \text{AbsErr}} \right)^2 \right)^{1/2} \quad (123)$$

where RelErr is the relative error for the solution variables. It can be set using the command:

```
pdbSetDouble <mater> <solution> Rel.Error <n>
```

NOTE If the error control parameter is not defined in the PDB for a material or a solution, the long-hand command `pdbSetDouble` must be used.

Time-Step Control for Mechanics

Automatic time-step control for mechanics is activated only if the structure contains certain nonlinear features that require Newton iterations, such as CZM, plasticity, viscoplasticity, and creep. The size of the time step is adjusted based on satisfaction of certain convergence criteria.

Convergence Criteria

To check the convergence of Newton iterations for mechanics equations [2][3], the criteria are:

- Force residual
- Energy
- Displacement

The force residual criterion checks the satisfaction of force equilibrium by comparing the maximum norm of the residual ($R = F^{\text{external}} - F^{\text{internal}}$) against a reference value:

$$\|R_{n+1}^i\|_{\infty} \leq \varepsilon_R \|R_{n+1}^{-1}\| \quad (124)$$

The reference value of the force residual is computed automatically by taking a norm of the element force residual vector at the first Newton iteration in a time step.

The energy criterion checks the satisfaction of the minimization of energy at equilibrium by comparing the change in energy against a reference value:

$$\sqrt{|\Delta u_{n+1}^i \cdot R_{n+1}^i|} \leq \varepsilon_E \sqrt{|\Delta u_n \cdot R_{n+1}^1|} \quad (125)$$

The reference value of energy is computed automatically by taking a dot product of the force residual and the displacement increment ($\Delta u = v\Delta t$) vector at the first Newton iteration in a time step.

The displacement criterion checks the satisfaction of the solution accuracy by comparing the maximum norm of the displacement increment against a reference value:

$$\|\Delta u_{n+1}^i\|_{\infty} \leq \varepsilon_U \|\Delta u_n\| \quad (126)$$

The reference value of displacement is computed automatically by taking a norm of the displacement increment vector at the start of the first Newton iteration in a time step.

The force residual and the energy criteria are checked by default. Optionally, the energy criterion may be replaced by the displacement criterion. Use the following parameters to activate or deactivate any of the convergence criteria:

```

pdbSet Mechanics Convergence.Force.Check <n>
pdbSet Mechanics Convergence.Energy.Check <n>
pdbSet Mechanics Convergence.Displacement.Check <n>

```

where <n> is either 0 (deactivate) or 1 (activate).

15: Numerics

Time-Step Control

The choices for the force residual and the displacement reference value norms that can be set using the following command are:

```
pdbSet Mechanics Convergence.Check.Norm [RMS | ABS]
```

where RMS refers to the root mean square value (default) and ABS refers to the mean absolute value.

The reference values for any of the convergence criteria can be changed by using the commands:

```
pdbSet Mechanics Convergence.Force.RefVal <n>  
pdbSet Mechanics Convergence.Energy.RefVal <n>  
pdbSet Mechanics Convergence.Displacement.RefVal <n>
```

where <n> is a suitable positive value.

By default, the tolerance for each of the convergence criteria is set to 0.001 and can be changed by using the commands:

```
pdbSet Mechanics Convergence.Force.Tolerance <n>  
pdbSet Mechanics Convergence.Energy.Tolerance <n>  
pdbSet Mechanics Convergence.Displacement.Tolerance <n>
```

where <n> is a value between 0.0 and 1.0.

Convergence criteria are checked in every iteration until either they are satisfied or the maximum number of Newton iterations is reached. The default value for the maximum number of Newton iteration is 8, and this can be changed using the command:

```
pdbSet Mechanics MaxIterations <n>
```

where <n> is a value greater than zero.

Time-Step Adjustment

The first time step for solving mechanics equations is set to the initial time given with the `solve` command. The size of subsequent time steps is decided based on the convergence history of the preceding time step. The time-step size is extended if the preceding time step converges quickly. The maximum value is limited to the maximum time given with the `solve` command. The time-step size is reduced if the preceding time step takes too many iterations to converge; it remains unchanged if the preceding time step takes a moderate number of iterations to converge.

The time-step size also is adjusted to keep the viscoplastic or creep strain error or value within tolerance when using such material models. The viscoplastic strain error or value is checked

for the Anand model (see [Anand Model on page 59](#)), while the creep strain error or value is checked for power law creep and the standard linear solid model (see [Power Law Creep on page 62](#) and [Standard Linear Solid Model on page 48](#)). Use the following commands to change the default settings for the viscoplastic or creep strain error or value:

```

pdbSet Mechanics StrainVP.Error.Tolerance <n>
pdbSet Mechanics StrainVP.Value.Tolerance <n>
pdbSet Mechanics StrainCr.Error.Tolerance <n>
pdbSet Mechanics StrainCr.Value.Tolerance <n>

```

where <n> is a value between 0.0 and 1.0 for both. By default, only the viscoplastic or creep strain error criterion is used with a tolerance of 0.02.

Time-Step Cutback

An automatic time-step cutback procedure interrupts the Newton iteration loop and restarts the time step with a smaller size when any of the following issues is encountered:

- Convergence criteria are not satisfied within the maximum number of Newton iterations.
- The solution converges very slowly over several iterations.
- The solution diverges over several iterations.
- Convergence criteria are satisfied, but viscoplasticity equations fail to converge.
- Convergence criteria are satisfied, but the viscoplastic or creep strain error or value is greater than the tolerance.

To check convergence details during solve steps, specify `info=1` in the `solve` command.

For more information about convergence during solve steps, use the command:

```

pdbSet Mechanics Convergence.Info <1 | 0>

```

Typical output with information `level=2` is:

```

-Iter 1
  lin norm 4.6088e-13  vel err 27.792
-----
      Mesh: region_1      Mater: Metal
      Org. Val. Update   New Val.   Error     Location
Velocity_y  1.95e-05  -5.31e-05  -3.35e-05  5.20e+01  (0.00e+00 0.00e+00 4.64e-03) (UCS)
                                           (4.64e-03 0.00e+00 0.00e+00) (DFISE)
-----
Largest update: Velocity_y in Metal @ (-3.94e-02 0.40e+00 0.00e+00) (UCS)
                                           ( 0.00e-00 0.40e+00 3.94e-02) (DFISE)
-----
Mechanics Assembly time: 0.09s
Maximum Residual 2.7020e+07 Reference Value 3.4748e+07 Tolerance 1.0000e-03
Energy 1.5095e+03 Reference Value 5.7383e+03 Tolerance 1.0000e-03

```

15: Numerics

References

where:

- `Org. Val.` is the original value at the node.
- `Update` is the update at the node.
- `New Val.` is `Org. Val. + Update`.
- `Maximum Residual` is the maximum norm of the force residual over the entire structure.
- `Energy` is the square root of the maximum norm of change in energy over the entire structure.
- `Location` is the location (in μm) of the node in both the UCS and the DF-ISE coordinate system.

References

- [1] R. E. Bank *et al.*, “Transient Simulation of Silicon Devices and Circuits,” *IEEE Transactions on Electron Devices*, vol. ED-32, no. 10, pp. 1992–2007, 1985.
- [2] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, Butterworth-Heinemann: Oxford, 5th ed., 2000.
- [3] K. J. Bathe and A. P. Cimento, “Some Practical Procedures for the Solution of Nonlinear Finite Element Equations,” *Computer Methods in Applied Mechanics and Engineering*, vol. 22, no. 1, pp. 59–85, 1980.

CHAPTER 16 Writing Partial Differential Equations Using Alagator

This chapter discusses how to specify partial differential equations and boundary conditions for use in simulations using the Alagator scripting language.

The partial differential equations (PDEs) are expressed in a Newton iteration–ready form and are specified as strings that are assumed to equal zero. Most mathematical operators are supported, and operators for differential terms are available.

Care must be exercised with the Tcl expansion of variables and strings because, usually, users want variables to be evaluated at runtime, not when they are read.

Available Operators and Variables

The operators and variables of the Alagator scripting language include binary operators, simple functions, differential functions, string names, solution names, subexpressions, constants, and parameters.

Binary and Unary Operators

Most common binary algebraic operators are supported. Unary negation is also supported with the usual mathematical rules applying. [Table 26](#) lists the supported binary and unary operators in order of precedence. Parentheses are supported for grouping operations.

NOTE The unary minus (-) and power (^) operators have nonstandard precedence.

Table 26 Precedence of supported binary and unary operators

Operator	Description
-	Unary minus
^	Power; for example, to raise a to the b power, use: a^b
*, /	Multiplication, division

16: Writing Partial Differential Equations Using Alagator

Available Operators and Variables

Table 26 Precedence of supported binary and unary operators

Operator	Description
+, -	Addition, subtraction
<, <=, >=, >, ==, !=	Equality, inequality NOTE Care must be used with <i>equals</i> and <i>not equals</i> , since a comparison of floating-point values in this way can be problematic.
&&	Logical <i>and</i>
	Logical <i>or</i>
?:	Conditional operator
,	Comma operator for lists

Many comparison operators are implemented. These do not support derivative operations, so they cannot be used with the `grad` operator (see [Differential Functions on page 345](#)). However, they can be used in the `select` command (see [select on page 609](#)).

Logical operators *and* (`&&`) and *or* (`||`) are provided for use with callback procedures and initialization. When these operators are used as part of a PDE, care must be taken as the Newton method does not ensure convergence for problems that are not first-order continuous.

The conditional operator (`?:`) takes three operands:

- The first operand is a condition.
- The second operand is the value of the entire conditional expression if the condition is true.
- The third operand is the value of the entire conditional expression if the condition is false.

NOTE When using the conditional operator (`?:`), use parentheses around the expressions to ensure the correct order of evaluation.

For example, the command:

```
select z= "(MyData>1e15) ? (1e15) : (MyData)" name= MyData
```

sets the value of `MyData` to 1×10^{15} on mesh points where `MyData` is greater than 1×10^{15} and does not change `MyData` on mesh points where `MyData` is smaller than 1×10^{15} . Since the `select` command works on mesh points, the conditional operator is very useful for truncating profiles.

Simple Functions

All simple functions take one argument that must be enclosed in parentheses. The argument can be any expression. Most common functions are available, including:

- 'exp' natural exponentiation
- 'log' natural log
- 'log10' log base 10
- 'sqrt' square root

In addition, the complementary error function 'erfc' and error function 'erf' are provided to help build initial doping profiles. All of these functions have supported derivatives and can be used in the specification of PDEs.

The 'abs' function and the 'sign' function provide an absolute value and a sign operation, respectively. The sign operation returns +1 if the argument is greater than zero and returns -1 if the argument is less than zero. These functions do not provide derivatives and cannot be used as part of a PDE.

Differential Functions

Differential functions are used in PDEs only and are not evaluated with the `select` command. The differential operators are `ddt` and `grad`.

Time derivatives are supported with the `ddt` operator. It takes a single argument and computes the first-time derivative of the argument for use in a PDE. Time-step integration is provided automatically using the Bank–Rose trapezoidal rule/backward differentiation formula (TRBDF) method [1].

A simple gradient is supported with the `grad` operator. Implied is the evaluation in a discrete sense and the integral around a control volume. For this reason, the divergence (`div`) operator is not required. For example, `A*B*grad(C)` is treated as `div(A*B*grad(C))`.

NOTE Sentaurus Interconnect automatically calculates the divergence if the term includes the `grad` operator.

String Names

Strings that are not recognized as real numbers, operators, or functions are compared to four sets of possible matches:

- The first set is valid solution names created with the `solution` command (see [solution on page 626](#)).
- The second set is named subexpressions created with the `term` command (see [term on page 658](#)).
- The third set is the data field name.
- Finally, any remaining strings pass to the Tcl expression function to see whether they can be parsed to a real number constant. This allows parameters from the parameter database to be used in PDEs.

Solution Names and Subexpressions

Solution names must match exactly the string specified in the `solution name=<c>` command (see [solution on page 626](#)). This is important because solution names link PDEs to the variables to be solved. Derivatives are taken automatically of all equations with respect to each solution name found in the PDE.

Terms are useful for common subexpressions (see [Using Terms on page 350](#)). The term name used in the PDEs must match exactly the string specified in the `term name=<c>` command (see [term on page 658](#)).

NOTE Terms and data fields with the same name can coexist in a structure. During equation parsing, the term name has higher priority than the data field name for a region.

Constants and Parameters

Anything that does not match the name of a term or solution passes to the Tcl expression parser to see whether it evaluates as a valid expression. The result is treated as a constant. The evaluation of the expression is performed again in the code if the temperature or time changes, so that parameters can have explicit dependencies on these values.

When defining parameters, care must be given to nested declarations. Especially when parameters are derived using the `pdbDelayDouble` command from other parameters, the Tcl expression parser may be unable to expand the entire expression and evaluate it correctly.

For example:

```

pdbSetDouble Si Test Param1 {[Arrhenius 1 2]} (1)
pdbSetDouble Si Test Param2 {2.0*[pdbDelayDouble Si Test Param1]} (2)
pdbSetDouble Si Test Param3 {2.0*[pdbGetDouble Si Test Param1]} (3)
pdbGet Si Test Param2 (4)
pdbGet Si Test Param3 (5)

```

Lines 1–3 set Param1, Param2, and Param3. Both Param2 and Param3 are derived parameters from Param1. While Param2 uses `pdbDelayDouble` to obtain the value of Param1, Param3 uses `pdbGetDouble`. When retrieving data, line 4 will return an error message, and line 5 will return a valid double number without an error message.

The error message is issued because `pdbDelayDouble` returns an expression of Param1, which is treated as a string by the Tcl expression parser during the evaluation of Param2.

To prevent such errors, Param2 can be encapsulated with the `expr` command, for example:

```

pdbSetDouble Si Test Param2 {[expr 2.0*[pdbDelayDouble Si Test Param1]]}

```

NOTE If Sentaurus Interconnect cannot evaluate an expression correctly, it assumes that it is zero.

Basics of Specifying Partial Differential Equations

In this section, an example is used to illustrate how to specify PDEs using the Alagator scripting language. The general expression for diffusion of species C_X is given by:

$$\frac{\partial C_X}{\partial t} = \nabla \cdot D \nabla C_X \quad (127)$$

This will be translated to the Alagator language as:

```

ddt (CX) -D*grad (CX) = 0

```

where CX is the solution variable and D is the diffusivity term (see [Using Terms on page 350](#)).

NOTE The divergence (`div`) operator is not specified in front of `D*grad (CX)` because it is implied and computed as part of the discretization of PDEs. Whenever the `grad` operator is in a term, it is assumed that the divergence will be taken of that term during assembly.

NOTE The argument of the `grad` operator must be a function of the solution variable.

NOTE Since the examples in this section use parameters that are not in the parameter database, long-hand `pdbSet` commands are used (see [pdbSet and Related Commands on page 540](#)).

Building Partial Differential Equations

The simplest diffusion equation uses a constant diffusivity and can be described by Fick's first law and second law. Two main steps are required to initialize and solve this equation:

1. A solution must be defined (see [solution on page 626](#)).
2. An equation must be entered into the parameter database.

A minimum of two commands can accomplish this, for example:

```
solution name=CX add !negative !damp solve
pdbSetString Silicon CX Equation "ddt(CX) - [Arrhenius 0.138 1.37]*grad(CX) "
```

The first command creates a new solution named `CX` and adds it to the solution list. The solution cannot take negative values, and numeric damping is not applied to the updates of the Newton iteration. The solution must always be solved.

NOTE Aliases are defined only for `pdb` commands. The solution name used in PDEs must match exactly the one specified with the `solution` command.

The second command creates an entry in the parameter database for the material `Silicon` and the solution variable `CX`. An entry is made for an `Equation`, which is the predefined entry that Alagator looks for to find a PDE. The string value is the PDE that will be solved for this variable in this material.

In this example, the PDE to be solved is:

$$d(CX)/dt - \text{div}(0.138 * \exp(-1.37/kT) * \text{grad}(CX)) = 0$$

Here, k is the Boltzmann constant and T is the temperature.

NOTE In the second command, `[Arrhenius 0.138 1.37]` is evaluated and converted to `"(0.138*exp(-1.37/((Temperature)*8.617383e-05)))"`. The Arrhenius function is a predefined helper function that allows for the simple creation of Arrhenius expressions. It uses the local temperature.

Setting the Boundary Conditions

After defining a solution and entering the equation into the parameter database, you need to set boundary conditions.

Dirichlet Boundary Condition

The previous example can be enhanced by adding a boundary condition. For simplicity, it is assumed that the surface concentration of variable CX is $5 \times 10^{19} \text{ cm}^{-3}$ at the Gas_Silicon interface.

The following commands create this boundary condition, and both commands work on the gas-silicon interface for the CX variable:

```
pdbSetBoolean Gas_Silicon CX Fixed_Silicon 1
pdbSetString Gas_Silicon CX Equation_Silicon "CX_Silicon - 5.0e19"
```

The first command indicates that the value will be set on the silicon side (with the `_Silicon` option on `Fixed`), that is, a Dirichlet boundary condition will be applied. The keyword `Fixed` is used only with the Dirichlet boundary condition. Fluxes will be ignored at this node and the boundary condition will control the concentration. Specifying `Fixed_Silicon` is critical because there can be three components on any interface: one for each material and one for the interface.

The second command sets the boundary condition equation. The CX variable has `_Silicon` appended to indicate that the concentration of $5 \times 10^{19} \text{ cm}^{-3}$ is set on the silicon side.

NOTE Equations are set to zero by definition.

NOTE Interface names are ordered alphabetically. Most interface names are set to the correct order using the `alias` command (for example, `Silicon_Gas` will be interpreted as `Gas_Silicon`). If a new interface name is introduced, the order must be followed.

Segregation Boundary Condition

If a segregation-type boundary condition is needed, for example, at the oxide–silicon interface, use the following commands:

```
pdbSetString Oxide_Silicon CX Equation_Oxide "(1.6e-7*(CX_Oxide - \  
CX_Silicon/0.28)) "  
pdbSetString Oxide_Silicon CX Equation_Silicon "-(1.6e-7*(CX_Oxide - \  
CX_Silicon/0.28)) "
```

This boundary condition assumes that an equation for CX is solved in the oxide region as well. Otherwise, the equations would be unbalanced at this interface.

Both commands work on the oxide–silicon interface. The `_Silicon` and `_Oxide` options on `Equation` indicate the side of the interface to which the given flux will be applied. The same options on the solution variable `CX` indicate whether the solution variable value at this interface is taken from the oxide side or the silicon side.

NOTE The fluxes have opposite signs. The first number (1.6×10^{-7}) in the flux equation is the transfer coefficient, and the second number (0.28) is the segregation coefficient.

Using Terms

A *term* is a common subexpression that can be used in multiple instances or to retrieve information. When the term appears in multiple equations, it reduces the maintenance of equations, and the values are easily retrieved from memory and accumulated. The previous example can be modified in the following way (see [term on page 658](#)):

```
term name= D Silicon eqn= "[Arrhenius 0.138 1.37]" store  
pdbSetString Silicon CX Equation "ddt(CX) - D * grad(CX) "
```

The first command creates the `D` term, not a local variable. The dollar sign (indicating a Tcl variable) is no longer needed for `D` in the subsequent equation.

The terms are retained until you exit the simulator, so they can be used in other PDEs or in a `select` command (if you want to monitor the diffusivity of `CX`).

Modifying Built-in Equations and Terms

Sentaurus Interconnect builds equations and terms for known solution variables by default. If you want to add a new expression to an existing equation or term, or to subtract a new expression from an existing equation or term, you can use one of the following commands:

- `UserAddEqnTerm`
- `UserSubEqnTerm`
- `UserAddToTerm`
- `UserSubFromTerm`

NOTE Do not use these commands within callback procedures. They are designed to change PDEs without using callback procedures (see [Using Callback Procedures to Build Models on page 353](#)).

NOTE All commands are saved to TDR files. If the command file is split, the commands must not be included in the new command file. However, if a user-defined variable or term is used with these commands, the variable or term must be included in the new command file, or the variable must be saved with the `define` command (see [define on page 398](#)) and the term must be stored using the `store` argument (see [term on page 658](#)).

UserAddEqnTerm and UserSubEqnTerm

The `UserAddEqnTerm` and `UserSubEqnTerm` commands allow you to add a new expression to an existing solution variable equation or to subtract the new expression from an existing solution variable equation. The commands have the format:

```
UserAddEqnTerm <material> <solution> <expression> <side> \  
  overwrite | !overwrite  
  
UserSubEqnTerm <material> <solution> <expression> <side>  
  overwrite | !overwrite
```

where:

- `<material>` is any valid material name.
- `<solution>` is any valid solution variable name.
- `<expression>` is the new expression to be added to or subtracted from the solution variable.

16: Writing Partial Differential Equations Using Alagator

Modifying Built-in Equations and Terms

- `<side>` is the side of the interface material where the new expression will be added or subtracted.
- `overwrite` | `!overwrite` specifies whether or not to overwrite the previous setting. The default is *not* to overwrite (`!overwrite`).

For example, the following command adds the expression " $\{2e-15*(CY*CY-1e-16*CX)\}$ " to the `CY` equation in silicon by overwriting the previous settings during the PDE solve:

```
UserAddEqnTerm Silicon CY "{2e-15*(CY*CY-1e-16*CX)}" overwrite
```

Since the equations can be set in three different ways for interfaces, you have the option to specify the side to which the expression will be added or subtracted.

For example, the commands:

```
UserAddEqnTerm Oxide_Silicon CX "(CX_Oxide - CX_Silicon)" Silicon
UserSubEqnTerm Oxide_Silicon CX "(CX_Oxide - CX_Silicon)" Oxide
```

add the expression " $(CX_Oxide - CX_Silicon)$ " to the `Oxide_Silicon` interface equation for `CX` on the `Silicon` side, and subtract the same expression from the `Oxide_Silicon` interface equation for `CX` on the `Oxide` side, respectively. If no side information is given, the expression will be added to the `Oxide_Silicon` interface equation for `CX`.

NOTE The `UserAddEqnTerm` and `UserSubEqnTerm` commands apply to materials that are *like* the one given with the `<material>` option.

UserAddToTerm and UserSubFromTerm

The `UserAddToTerm` and `UserSubFromTerm` commands add a new expression to an existing term or subtract the new expression from an existing term. The commands have the format:

```
UserAddToTerm <material> <term> <expression>
```

```
UserSubFromTerm <material> <term> <expression>
```

where:

- `<material>` is any valid material name.
- `<term>` is an existing term name.
- `<expression>` is the new expression to be added to or subtracted from the existing term.

For example, the following command adds the expression " $2*CX$ " to the term `MyTerm` in silicon:

```
UserAddToTerm Silicon MyTerm "2*CX"
```

In the same way, the following command subtracts the expression "2*CX" from the term MyTerm in silicon:

```
UserSubFromTerm Silicon MyTerm "2*CX"
```

NOTE The `UserAddToTerm` and `UserSubFromTerm` commands do not apply to materials that are *like* the one given with the `<material>` option.

Using Callback Procedures to Build Models

NOTE Callback procedures involve complex operations. Only advanced users should use them.

Callbacks allow additional *intelligence* to be built into PDEs by allowing procedures to be called at runtime. These procedures build the Alagator equation strings according to user-specified options. By selecting model switches, you can choose between different physical models to be represented in the equation strings. By having callback procedures that use a material name and a solution name as arguments, the same type of equation can be built for several materials and solutions.

In Sentaurus Interconnect, all frequently used equations are built-in Tcl callback procedures. Using the `pdbSet` command, you can instruct Alagator to use various callback procedures.

You use specific keywords to create callback procedures. [Table 27](#) lists the callback procedure–related keywords in Alagator.

Table 27 Keywords used to create callback procedures

Keyword	Description
EquationProc	Specifies the name of the callback procedure to set up the PDEs (see Constructing Equation Strings: EquationProc on page 358).
InitProc	Specifies the name of the callback procedure that usually resets or deletes existing parameter database equations or terms when the <code>solve</code> command is called (see Cleaning Up Equation Strings: InitProc on page 357).

Callbacks During Execution of the solve Command

In addition to calling the callback procedures at various stages during the execution of a `solve` command, Sentaurus Interconnect calls the `solvePreProcess` procedure *before* executing

16: Writing Partial Differential Equations Using Alagator

Using Callback Procedures to Build Models

the `solve` command and the `solvePostProcess` procedure *after* executing the `solve` command. Figure 103 shows the flowchart of the process.

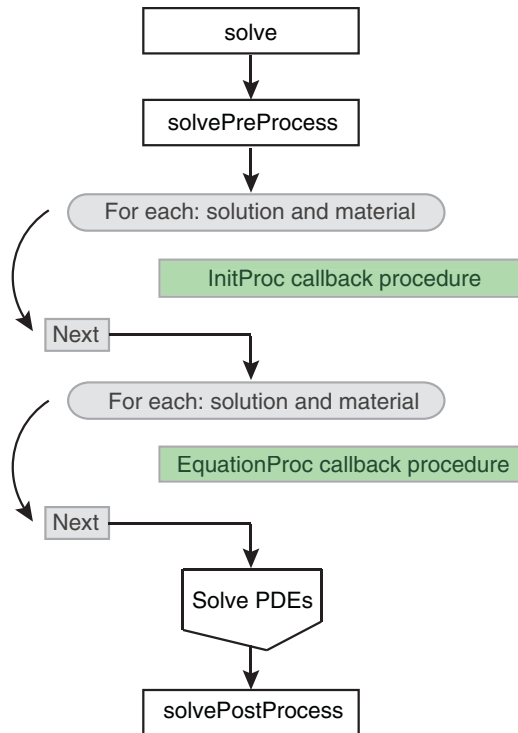


Figure 103 Flowchart with calls to callback procedures during execution of a solve command

The solvePreProcess Procedure

Sentaurus Interconnect can initialize solution variable fields on the command line using commands such as `select` (see [select on page 609](#)) and `profile` (see [profile on page 583](#)). If the initialization can be standardized, it is better to use the `solvePreProcess` procedure because, by default, it initializes the data fields for potential and temperature.

The execution of each `solve` command starts with a call to the `solvePreProcess` procedure that initializes various data fields or preprocesses existing data fields. For example, the `solvePreProcess` procedure initializes solution variables. In this procedure, the potential and temperature equations are switched on or off according to the simulation models selected. In addition, the appropriate boundary conditions for each simulation model are set.

To enforce additional actions to be performed upon solve preprocessing, it is not necessary to overwrite the default implementation of the `solvePreProcess` procedure. Instead, to preserve the default initialization of data fields, you can redefine the `UserSolvePreProcess` procedure. By default, `UserSolvePreProcess` is an empty procedure and is called from the `solvePreProcess` procedure as one of the last commands.

Building and Solving PDEs

After the execution of the `solvePreProcess` procedure, Sentaurus Interconnect checks all material and solution names to see whether a callback procedure has been specified using the `InitProc` keyword (see [Cleaning Up Equation Strings: InitProc on page 357](#)). If a callback procedure is defined, it is called for the specified material name and solution name. Such a callback procedure usually sets the equation strings to empty strings and removes terms defined in previous steps. By having empty equation strings, the equations and terms can be built up piecewise, by adding expressions for each selected model that contributes to an equation or a term. This is necessary because different models may be used for different solve steps, and because additional solution variables may be added between solve steps, which may require terms to be added to the equations for existing species.

In the next step, Sentaurus Interconnect checks all material and solution names to see whether a callback procedure has been specified using the `EquationProc` keyword (see [Constructing Equation Strings: EquationProc on page 358](#)). If a callback procedure is defined, it is called with the material name and solution name as arguments. Such a procedure is used to set the PDEs for the solution variable. Alternatively, if no callback procedure is defined for a material and a solution, the equation string can be set on a command line without specifying any callback procedures.

NOTE If a callback procedure has been specified using the `EquationProc` keyword for a material name and solution name, this callback procedure typically overwrites any equation specified on the command line for that material and solution.

After the equations are set, Sentaurus Interconnect solves the equations using the local temperature.

The `solvePostProcess` Procedure

Finally, the `solvePostProcess` procedure is called when the `solve` command is finished. By default, it deletes the data fields that are no longer needed and stores the conduction current density, the electric fields, and the temperature gradient.

The last command line of the `solvePostProcess` procedure calls the procedure `UserSolvePostProcess`, which is empty by default. To add commands to be executed after the `solve` command is finished, you can redefine the `UserSolvePostProcess` procedure.

NOTE The `solvePreProcess` and `solvePostProcess` procedures can be found in the `TclLib` directory in the `SolveProcess.tcl` file.

Specifying Callback Procedures Using Keywords

This section demonstrates how to use keywords to specify callback procedures.

Common Features of Using Keywords

To avoid unnecessary repetition of content, this section describes common features that apply to all keywords described in the next sections.

Using the `InitProc` keyword as an example, here is a simple demonstration of how you would use this keyword to create a callback procedure:

```
pdbSetString Silicon CX InitProc CleanEquations

proc CleanEquations { Mat Sol } {
    LogFile "This callback procedure unsets $Sol equation in $Mat."
    pdbUnSetString $Mat $Sol Equation
}
```

where:

- The first line specifies the name of the callback procedure in silicon for CX, which follows the keyword instance. In this case, the callback procedure is named `CleanEquations`.
- The second line defines the `CleanEquations` callback procedure.

The callback procedure takes two arguments: a material name (`Mat`) and a solution name (`Sol`). In this example, Sentaurus Interconnect calls the `CleanEquations` procedure with these two arguments: The first will be `Silicon` and the second will be `CX`.

NOTE The argument names `Mat` and `Sol` are arbitrary. They can be any valid Tcl variable but the first argument is *always* the material name and the second argument is *always* the solution name.

- The third line defines the message string that will be printed to the log file when the callback procedure is executed. This is to inform users that the callback procedure is called. The `$Sol` variable and `$Mat` variable will be replaced with the solution name and the material name, respectively. In this case, the message would be:

```
This callback procedure unsets CX equation in Silicon.
```

- The fourth and last line unsets the equation if it exists.

Generic Procedure

When neither the material name nor the solution name is used in the implementation of a callback procedure, it is a *generic* procedure and can be used for several materials and solutions.

The examples in the next sections demonstrate generic procedures.

Cleaning Up Equation Strings: InitProc

The `InitProc` keyword specifies the name of the callback procedure that cleans up equation strings.

For example, the `InitProc` keyword defines the `ResetEquations` callback procedure of the solution variable `CX` in silicon:

```
pdbSetString Silicon CX InitProc ResetEquations

proc ResetEquations { Mat Sol } {
  LogFile "This callback procedure unsets $Sol equation in $Mat."
  pdbUnSetString $Mat $Sol Equation
}
```

Sentaurus Interconnect will call the `ResetEquations` procedure with two arguments: The first will be `Silicon` and the second will be `CX`.

The procedure is called every time the solutions are checked during the PDE solve. The `ResetEquations` procedure prints the message to the log file and removes the parameter database equation if it was defined.

This example can be extended with the following commands:

```
pdbSetString Silicon CX InitProc ResetEquations
pdbSetString Silicon CY InitProc ResetEquations
pdbSetString Oxide CX InitProc ResetEquations
pdbSetString Oxide CY InitProc ResetEquations
```

In this case, the same `ResetEquations` callback procedure is used for the solution variables `CX` and `CY` in oxide and silicon. Sentaurus Interconnect will print the following messages:

```
This callback procedure unsets CX equation in Silicon.
This callback procedure unsets CY equation in Silicon.
This callback procedure unsets CX equation in Oxide.
This callback procedure unsets CY equation in Oxide.
```

The advantage of callback procedures is clear. With four new command lines, the equations for `CX` and `CY` in both oxide and silicon can be unset. At the same time, there is only one callback procedure to maintain. If you change the callback procedure, the changes will apply to all four settings.

16: Writing Partial Differential Equations Using Alagator

Summary

Constructing Equation Strings: EquationProc

The `EquationProc` keyword specifies the name of the callback procedure that constructs the equation string in the parameter database.

For example, the first command defines `SetEquations` as the callback procedure of the solution variable `CX` in silicon:

```
pdbSetString Silicon CX EquationProc SetEquations

fproc SetEquations { Mat Sol } {
    LogFile "This callback procedure sets $Sol equation in $Mat."
    pdbSetString $Mat $Sol Equation "ddt($Sol) - \[Arrhenius 0.138 1.37\] \
        *grad($Sol)"
}
```

Sentaurus Interconnect will call `SetEquations` with two arguments: The first will be `Silicon` and the second will be `CX`.

The `SetEquations` callback procedure is called every time the solutions are checked during the simulation. The `SetEquations` procedure prints the message to the log file and sets the parameter database equation for `CX` in silicon.

NOTE In this example, the `SetEquations` callback procedure is a generic procedure. The equation setting is similar to that explained in [Building Partial Differential Equations on page 348](#). The only difference is that, instead of using a material name and solution name, only Tcl variables are used.

This example can be extended with the following commands:

```
pdbSetString Silicon CX EquationProc SetEquations
pdbSetString Silicon CY EquationProc SetEquations
```

In this case, the same `SetEquations` callback procedure is used for the solution variables `CX` and `CY` in silicon. Sentaurus Interconnect will print the following messages:

```
This callback procedure sets CX equation in Silicon.
This callback procedure sets CY equation in Silicon.
```

Summary

You have been shown how to set up PDEs and boundary conditions, and to initialize solution variables. The example can be divided into three parts.

The first part is the `pdb` entries for the models such as diffusivities and conductivity. These parameters correspond to the parameter database. All default Sentaurus Interconnect model parameters are stored in the parameter database.

The second part defines the names of the solution variables and the names of the callback procedures:

```
solution add name=CX !damp !negative solve  
  
pdbSetString Silicon CX EquationProc SetEquations  
pdbSetString Silicon CX InitProc ResetEquations
```

The third part defines the models.

References

- [1] R. E. Bank *et al.*, “Transient Simulation of Silicon Devices and Circuits,” *IEEE Transactions on Electron Devices*, vol. ED-32, no. 10, pp. 1992–2007, 1985.

16: Writing Partial Differential Equations Using Alagator
References

This appendix describes the commands of Sentaurus Interconnect.

Syntax Conventions

The commands are designed to optimize the use of the tool command language (Tcl).

The following conventions are used for the command syntax:

- Each command starts on a new line.
- A backslash (\) is used to extend a command on to multiple lines if it appears as the last character on the line. It is not included on the last line of a command. For example:

```
refinebox name= SiOxPo mask= POLY extend= 0.1 \  
    extrusion.min= -1.51 extrusion.max= -1.35 min.normal.size= 0.005 \  
    interface.mat.pairs= {Silicon Oxide Silicon Polysilicon}
```

- Braces – { } – indicate a list. The braces are part of the syntax.

NOTE Usually, Tcl lists are created using braces, but Tcl does not typically evaluate expressions enclosed by braces. However, for convenience, Sentaurus Interconnect does evaluate all parameters that take numeric values even when enclosed by braces.

- Brackets – [] – indicate an optional argument. The brackets are *not* part of the syntax.
- Parentheses – () – indicate grouping of arguments. The parentheses are *not* part of the syntax.
- A vertical bar – | – indicates options, only one of which can be specified. Vertical bars are *not* part of the syntax.
- Angle brackets – < > – indicate specific type of input. The angle brackets are *not* part of the syntax.

NOTE An exception to this is that angle brackets must be included in the syntax if units are specified in a command. For example:

```
line x location= 0<um> spacing= 0.02<um> tag= surf
```

A: Commands

Syntax Conventions

- The following type identifiers are used:
 - `<c>`: Replace with a string. In general, strings are enclosed in double quotation marks (" "), for example:

```
line_edge_roughness normal= "Z" masks= {mask1} \  
correlation.length= 25.00<nm> standard.deviation= 5.00<nm>
```
 - `<i>`: Replace with an integer.
 - `<n>`: Replace with a floating-point number.
 - `<list>`: Replace with a Tcl list of values. The list must be enclosed in braces, for example, `min= {-5.0 0.0 0.0}`.
 - `<field>`: Replace with the name of a field.
 - `<material>`: Replace with the name of a material.
 - `<solution>`: Replace with the name of a solution.
- Boolean arguments are true (or on) if they appear on the command line (for example, `negative`).
- Boolean arguments are false (or off) if they appear on the command line preceded by an exclamation mark (for example, `!negative`).

Example of Command Syntax

An example of command syntax is:

```
command  
  argument1=<n>[<unit1>]  
  argument2=<i>  
  argument3=<i>  
  argument4  
  ...
```

There are two types of argument:

- *Named* arguments must be specified with a value, for example, `name=Germanium`.
- *Unnamed* arguments supply only the value, for example, `<material>` would be replaced by `Oxide`.

If you specify units for an argument:

- Omit spaces between the value and unit.
- Include the angle brackets with the unit, for example, `x10=5<um>`.

If you specify lists:

- Lists must be enclosed in braces.
- Elements in the list must be separated by space.
- You must insert space between the equal sign and the opening brace. For example:

```
transform translate= {-1 0 0}
```

Common Arguments

Nearly all Sentaurus Interconnect commands (with the exception of those implemented as Tcl procedures as well as a few others) support two common optional arguments:

- `info=<i>`

This argument sets the amount of information to be printed to the screen and the log file. It can take the value 0, 1, or 2. The default is 0, which is the minimum amount. Higher values give more details about the status of the simulation as well as model and parameter selection information. Any value greater than 2 will be interpreted as 2.

This argument can be used with any other argument combination for nearly all commands that are not Tcl procedures.

For example, to specify that more information should be printed to the screen and the log file for the `etch` command, use:

```
etch info= 2
```

- `parameters`

This argument prints all the available arguments for a command.

For example, to print the arguments for the `extract` command, use:

```
extract parameters
```

NOTE When you use the `parameters` argument, the complete list of arguments for that command is generated. However, some of the arguments in this list are not documented. These parameters are used for debugging purposes only.

A: Commands

Quantities and Units for Command Arguments

Quantities and Units for Command Arguments

Table 28 lists the quantities and units for arguments used in the commands. Many command arguments specify physical values. Such arguments are given a *quantity*, and any unit of a given quantity can be used. The factor listed is the value that is multiplied by the given value to convert to the default unit (which has a factor of 1).

Table 28 Quantities, units, and factors for command arguments

Quantity	Unit	Factor
Time	s	1
	ms	1.00e-03
	min	60
	hr	3600
Distance	cm	1
	Angs	1.00e-08
	nm	1.00e-07
	um	1.00e-04
	m	1.00e+02
Volume	cm ³	1
	Angs ³	1.00e-24
	nm ³	1.00e-21
	um ³	1.00e-12
	m ³	1.00e+06
Dose	cm ⁻²	1
Concentration	cm ⁻³	1
	m ⁻³	1.00e-06
	um ⁻³	1.00e+12
	nm ⁻³	1.00e+21
	Angs ⁻³	1.00e+24

Table 28 Quantities, units, and factors for command arguments

Quantity	Unit	Factor
Velocity	cm/s	1
	Angs/hr	2.78e-12
	Angs/min	1.67e-10
	Angs/s	1.00e-08
	nm/min	1.67e-09
	nm/s	1.00e-07
	um/min	1.67e-06
Diffusivity	cm ² /s	1
	um ² /min	1.67e-10
Flow	l/min	1
Stress	dyn/cm ²	1
	Pa	1.00e+01
	kPa	1.00e+04
	MPa	1.00e+07
	GPa	1.00e+10
	atm	1.01e+06
	torr	1.33e+03
Viscosity	poise	1
	Pa*s	1.00e+01
Energy	eV	1
	keV	1.00e+03
	MeV	1.00e+06
Temperature	C	1
	K	1
Temperature rate	C/s	1
	K/s	1
	C/min	1.67e-02
	K/min	1.67e-02
Inverted temperature	1/K	1

A: Commands

Quantities and Units for Command Arguments

Table 28 Quantities, units, and factors for command arguments

Quantity	Unit	Factor
Angle	degree	1
Current	A	1
Current density	A/cm ²	1
Current rate	A/s	1
	A/min	1.67e-02
Charge	Coulomb	1
Charge density	C/cm ³	1
	Coulomb/cm ³	1
Dose rate	cm ⁻² /s	1
	cm ⁻² /min	1.67e-02
Electric potential	V	1
Electric potential rate	V/s	1
	V/min	1.67e-02
Power	W	1
	nW	1.00e-09
	uW	1.00e-06
	mW	1.00e-03
Power density	W/cm ³	1
	nW/cm ³	1.00e-09
	uW/cm ³	1.00e-06
	mW/cm ³	1.00e-03
Power density rate	W/cm ³ /s	1
	W/cm ³ /min	1.67e-02
Resistivity	ohm-cm	1
	kohm-cm	1.00e+03
Force	dyn	1
	N	1.00e+05
Toughness	dyn/cm	1
	N/m	1.00e+03

Table 28 Quantities, units, and factors for command arguments

Quantity	Unit	Factor
Moment	dyncm	1
	Nm	1.00e+07
Mass	g	1
	amu	1.66e-24

alias

Sets and prints aliases.

Syntax

```
alias <c> [<c>] [-list]
```

Arguments

<c>

If one argument is specified, only one alias for this value is printed.

If two arguments are specified, a new alias is set.

-list

Prints a list of allowed aliases.

Description

This interactive mode command sets and prints aliases.

If the first argument is `-list`, a list of allowed aliases is printed. Otherwise, only one alias corresponding to the first argument is printed.

Examples

Set a new alias of `Temperature`:

```
alias Temp Temperature
```

Print the list of allowed aliases:

```
alias -list
```

Print an alias of `Temp`:

```
alias Temp
```

ArrBreak

Creates two Arrhenius expressions that depend on a break temperature.

Syntax

```
ArrBreak <n> <n> <n> <n> <n>
```

Arguments

<n>

The first argument is the prefactor and the second argument is the activation energy [eV] of the first Arrhenius expression.

The third argument is the prefactor and the fourth argument is the activation energy [eV] of the second Arrhenius expression.

The final argument is the break temperature [°C].

Description

This command creates two Arrhenius expressions and switches from the first expression to the second one at the given break temperature.

The first Arrhenius expression is computed when the temperature is below the break temperature. The second Arrhenius expression is computed when the temperature is equal to or greater than the break temperature.

Examples

Create two Arrhenius expressions – $3.0 e^{-0.5/k_B T}$ and $2.0 e^{-0.4/k_B T}$ – with a break temperature of 825°C. The first Arrhenius expression is computed when $T < 825^\circ\text{C}$, and the second Arrhenius expression is computed when $T \geq 825^\circ\text{C}$:

```
ArrBreak 3.0 0.5 2.0 0.4 825.0
```

A: Commands

Arrhenius

Arrhenius

Creates an Arrhenius expression.

Syntax

```
Arrhenius <n> <n>
```

Arguments

<n>

The first argument is the prefactor and the second argument is the activation energy [eV] of the Arrhenius expression.

Description

This command creates an Arrhenius expression.

Examples

Create the Arrhenius expression $4.0 e^{-0.5/k_B T}$:

```
Arrhenius 4.0 0.5
```

beam

Creates a beam for multiple-beam etching.

Syntax

```
beam
  name=<c>
  (direction= {<x> <y>} | incidence=<n>)
  factor=<n>
  [list]
```

Arguments

`direction`

Defines the angle of incidence of the beam using a direction vector. The specified direction vector is normalized automatically to unit length.

`factor`

If multiple beams are defined, `factor` defines the relative strength of each beam.

`incidence`

Defines the angle of incidence of the beam. An angle of 0 is vertical. The angle is measured counterclockwise, that is, a positive angle implies a beam ray entering from the upper left towards the lower right. A negative angle implies a beam ray entering from the upper right towards the lower left.

`list`

Returns a Tcl list of known beams.

`name`

Specifies the beam name to be referenced using `sources` of the `etch` command.

Description

This command defines the direction and relative strength of etchant beams. The beam name is referenced in the `etch` command. The angle of incidence of the beam can be given using `direction` or `incidence`. The relative strength `factor` is used to mix the strength of different beams. Etchant beams are assumed to be collimated, that is, a slight angular spread of beam direction is not taken into account.

A: Commands

beam

Examples

Define a vertical beam called `source1` and a beam called `source2` at half the strength of `source1` at an angle of 10° (positive angle implies that the beam travels from the upper left to the lower right). A third beam called `source3` at one-tenth the strength of `source1` enters from the upper right slightly towards the lower left:

```
beam name= source1 incidence= 0 factor= 1  
beam name= source2 incidence= 10 factor= 0.5  
beam name= source3 direction= {1 -0.1} factor= 0.1
```

See Also

[etch on page 414](#)

bound

Extracts the boundary of a material or region, and returns the outline as a list of coordinates.

Syntax

```
bound  
  <material> | region=<c> [list.all]
```

Arguments

<material>

Specifying a material extracts the boundary of all regions of the specified material. See [Specifying Materials on page 18](#).

list.all

This option lists all boundary coordinates of a region with no ordering. For regions with more than one boundary, only one closed boundary will be listed by default. If this option is switched on, all boundaries will be listed in one list but with no ordering. Default: false.

region

Specifying a region extracts the boundary of that region.

Description

This command is used to plot the limits of the regions for further processing. It returns a list of lists of coordinates of the boundary. The outer lists are distinct parts of the regions. Each outer list comprises a complete circle around that part. Each inner list contains coordinate pairs in order around the regions. The coordinate pairs are written in xy order around the material.

NOTE This command is not available for 3D simulations.

Examples

Return the boundary of oxide material:

```
bound oxide
```

Return the boundary of the region named Silicon_1:

```
bound region= Silicon_1
```

boundary

Controls the conversion of GC structures generated by Sentaurus Process Explorer into the boundary representation (brep) used by Sentaurus Interconnect.

Syntax

```
boundary
  [spx2brep.method= MLS | Standard]
  [spx2brep.mls.cellsize=<n>]
  [spx2brep.mls.decimateblockbreps]
  [spx2brep.mls.decimatebrep]
  [spx2brep.mls.prefilterdecimation]
  [spx2brep.mls.splinesmooth]
  [spx2brep.standard.decimation]
  [spx2brep.standard.decimation.accuracy=<n>]
  [spx2brep.standard.decimation.ridge.angle=<n>]
  [spx2brep.standard.decimation.shortest.edge=<n>]
  [spx2brep.standard.tolerance=<n>]
```

Arguments

`spx2brep.method`

Selects the conversion method from the following options:

- `MLS` uses a multimaterial level set-based conversion to smooth and filter out small features in the geometry. The MLS algorithm produces guaranteed well-defined breps that are suitable for simulation, but it is considerably slower than the standard algorithm. Internally, the algorithm first performs a standard brep conversion and then filters the resulting brep using the level-set algorithm.
- `Standard` provides a fast conversion and, by default, it does not apply any smoothing, resulting in a *blocky* appearance of the structure. The argument `spx2brep.standard.tolerance` controls this conversion method. After the conversion, the brep is decimated if the `spx2brep.standard.decimation` option is specified.

Default: `Standard`.

`spx2brep.mls.cellsize`

Sets the resolution of the cells used by the MLS algorithm. By default, the cells are four times the size of the resolution used in the GC structure.

`spx2brep.mls.decimateblockbrep`

If specified, the MLS algorithm decomposes the domain into blocks to improve the performance of the conversion. Default: false.

`spx2brep.mls.decimatebrep`

If specified, a decimation step is performed after the brep has been processed by the MLS algorithm. Default: false.

`spx2brep.mls.prefilterdecimation`

If specified, a decimation step is performed before sending the brep to the MLS algorithm. Default: false.

`spx2brep.mls.splinesmooth`

If specified, a b-spline filter is applied to smooth the geometry during the MLS conversion. The smoothing radius is defined as $16 \times h$, where h is the height of the GC column. Default: true.

`spx2brep.standard.decimation`

If specified, a decimation step is performed after the GC to brep conversion. Default: true.

`spx2brep.standard.decimation.accuracy`

Sets the accuracy used during decimation. Points are removed from the brep only if the deformation induced in the structure is less than the value specified by this argument. Default value and unit: $1.0 \times 10^{-8} \mu\text{m}$.

`spx2brep.standard.decimation.ridge.angle`

Sets the criterion used to detect feature ridges in the structure. Feature ridges are preserved and, if possible, not deformed during the decimation process. Default value and unit: 150° .

`spx2brep.standard.decimation.shortest.edge`

Sets the length of the shortest edge allowed in the final brep structure. Default value and unit: $1.0 \times 10^{-8} \mu\text{m}$.

`spx2brep.standard.tolerance`

Specifies how much smoothing is performed on the columns during conversion. The default value is 0.0, that is, no smoothing.

The following example sets the tolerance to 5.0, which represents reasonable smoothing:

```
boundary spx2brep.standard.tolerance=5.0
```

A: Commands

boundary

NOTE Using the standard algorithm can result in nonconformal brep geometries for nonzero values of `spx2brep.standard.tolerance`. For this reason, it is recommended to set this argument only if the resulting brep structure is intended for visualization purposes.

Description

Conversions occur when Sentaurus Interconnect reads a TDR GC file using the `init` command. After the `init` command is executed, Sentaurus Interconnect works as usual, using the MGOALS module to produce structures.

circuit

Defines a circuit for mixed-mode simulation.

Syntax

```
circuit
  [add] [clear] [contact] [current]
  [electrodes= {<c> <c>}]
  [element]
  [instance.parameters= {<c>}]
  [interconnect= { {<c> <c>} {<c> <c>} }]
  [list] [model.name=<c>] [name=<c>] [node] [print]
  [set= { {<c> <n>} {<c> <n>} }]
  [<solution>]
  [thermodes= {<c> <c>}]
```

Arguments

add

Adds a new circuit element with specified details.

clear

Clears all circuit information.

contact

Retrieves circuit-to-contact connection information.

current

Retrieves the current between a node and an element.

electrodes

Specifies the electrodes of a circuit element.

element

Retrieves circuit element information.

instance.parameters

Specifies the parameters, such as resistance, of a circuit element.

A: Commands

circuit

interconnect

Connects a contact to an electrode or a thermode.

list

Prints a list of circuit nodes or elements.

model.name

Specifies the name of the model from which the element instance is derived.

name

Specifies the name of the circuit element or node.

node

Retrieves circuit node information.

print

Prints the node or element information requested.

set

Sets the boundary conditions on thermodes.

<solution>

Specifies the solution name for the circuit. It is used to retrieve information, such as voltage, regarding the specified solution name at the circuit.

thermodes

Specifies the thermodes of a circuit element.

Description

This command adds circuit elements to a mixed-mode simulation. It also can be used to obtain information about circuit nodes and elements.

Examples

Add a circuit element (resistor) that is derived from the model `Resistor_pset`. The name of the element is `r3`, and its electrodes are `1` and `0`. The circuit element `resistance` is `1000` units:

```
circuit add name= r3 model.name= Resistor_pset electrodes= {1 0} \  
instance.parameters= {resistance= 1000}
```


List all the circuit nodes:

```
circuit node list
```

List all the contacts of the physical device to which the circuit nodes are connected:

```
circuit node contact list
```

Print the current between the circuit element r3 and the node 1:

```
circuit node= 1 name= r3 current print
```

Clear all circuit information:

```
circuit clear
```

Set Dirichlet boundary conditions on the thermode t0 with the value 300 K:

```
circuit set= { {t0 300} }
```

Compatibility

Applies parameters consistent with the default values of a previous release.

Syntax

```
Compatibility <c>
```

Arguments

<C>

Specifies the release from which to apply parameters. Aliases are available for the release, so it is not necessary to know the release foundation letter. For example, 2016.12 can be used instead of M-2016.12.

Description

If used, this command must be the first command in the command file, so that all subsequent commands that depend on the default values take into account the compatibility setting (see [Compatibility With Previous Releases on page 20](#)).

Examples

Apply parameters consistent with Version M-2016.12:

```
Compatibility 2016.12
```

contact

Defines a contact for subsequent simulation.

NOTE If the contacts are defined before the `init` command (see [init on page 458](#)) in mixed-mesh mode, you must switch on contacts in the boundary representation using the command:

```
pdbSet Grid Contacts.In.Brep 1
```

Syntax

```
contact
  [add] [clear] [charge | current | power | temperature | voltage]
  [depth=<n>] [<um>]
  [left] [right] [back] [front] [top] [bottom]
  [list] [list.existing] [list.region]
  [merge= <list>]
  [name=<c>] [new.name=<c>]
  [precision=<n>]
  [print]
  [region=<c>]
  [sidewall]
  [<solution>]
  [width=<n>] [<um>]
  (
    [box] [<material>] [adjacent.material=<c>] [cut.mesh]
    [xlo=<n>] [<m>|<cm>|<um>|<nm>]
    [xhi=<n>] [<m>|<cm>|<um>|<nm>]
    [ylo=<n>] [<m>|<cm>|<um>|<nm>]
    [yhi=<n>] [<m>|<cm>|<um>|<nm>]
    [zlo=<n>] [<m>|<cm>|<um>|<nm>]
    [zhi=<n>] [<m>|<cm>|<um>|<nm>] |
    [point]
    [x=<n>] [<m>|<cm>|<um>|<nm>]
    [y=<n>] [<m>|<cm>|<um>|<nm>]
    [z=<n>] [<m>|<cm>|<um>|<nm>]
    [replace]
  )
```

A: Commands

contact

Arguments

add

If the `contact` command is called multiple times with the same name, it overwrites the previous definition by default. If `add` is specified, the `contact` command will instead add to the existing contact indicated by name or create a new contact if it does not already exist.

adjacent.material

Specifies a second material for the contact. Only elements at the interface between the two materials are allowed for the contact.

box, point

Selects one of the supported contact types:

- `box`: Specifies a box-type contact that consists of elements at the surface of one region or material inside the box, defined for the contact. When choosing a box-type contact, the mesh is cut where the box intersects the chosen region to give an accurate size for the contact (see `cut.mesh`). Occasionally, this cutting produces poor quality mesh elements. In such cases, cutting can be switched off with `pdBSet Grid Cut.At.Contacts 0`. Use the `line` command to insert lines in the mesh to retain contact size accuracy if required.
- `point`: Specifies a region-type contact that contains all the boundary elements of one region. The region can be specified or the material and the x-, y-, and z-coordinates of one point can be specified to select the region.

NOTE Box-type contacts should be used to set up mechanical boundary conditions. Region-type contacts are not recommended for setting up mechanical boundary conditions.

clear

Clears the list of all contacts. If `name` is specified, `clear` removes only the specified contact.

charge, current, power, temperature, voltage

If specified, returns the total charge, or the total current, or the total power, or the average temperature, or the average voltage at the contact.

cut.mesh

By default, when a box contact is created, the mesh is cut at the contact borders to ensure accurate contact dimensions. Specifying `!cut.mesh` switches off mesh cutting, providing better element quality at the contact borders, but possibly sacrificing accuracy of the

contact borders. The contact will only include nodes of the existing mesh within the contact borders.

depth

Depth of the contact in micrometers.

left, right, back, front, top, bottom

These arguments selectively switch on certain outer boundaries where contacts will be placed. Outer boundaries are those of the simulation domain, which includes gas regions.

By default, only `top` and `bottom` are switched on, and the rest are switched off. If any of these arguments are specified, internal interfaces are switched off. In addition, `sidewall` is equivalent to specifying all these arguments.

list

Prints a list of currently defined contacts.

list.existing

Returns a list of all contacts currently existing in the bulk structure. Adding contacts to the mesh is delayed by default. However, additional contacts that do not appear in the returned list may be read directly from a file and, therefore, exist in the structure.

list.region

Prints a list of regions adjacent to the named contact.

<material>

Specifies the material for the contact. Contacts in TDR files are always defined as a set of surface elements. Only elements at the surface of volume regions of the specified material are selected. See [Specifying Materials on page 18](#).

merge

Specifies a list of contacts that must be electrically merged. If `name` is specified, a new contact with the name `name` is created, and contacts in the list are deleted.

name

Name of the contact.

new.name

Used with `name` to change the name of a contact from that specified by `name` to that specified by `new.name`.

A: Commands

contact

precision

Sets the number of digits after the decimal point for the printed value of the total charge, or the total current, or the total power, or the average temperature, or the average voltage at the contact. You also can use the following command for all `contact` commands:

```
pdbSet Compute Matrix.Precision <n>
```

print

Prints the contact information.

region

Name of the volume region or name of the interface region to be used for the contact. For volume regions, only surface elements of that region are selected for the contact. For interface regions, all elements of that region are selected for contacts.

replace

If specified, the material of the region of a contact is replaced by gas for point contacts in the TDR file.

NOTE Replacement of the contact material with gas occurs only when the TDR file is written. However, material replacement is not allowed for the boundary representation geometry that is included in the TDR file. This is needed to restart simulations from a saved TDR file. In addition, the internal structure remains unchanged while the TDR file is written. Therefore, no material replacement occurs for the internal structure.

sidewall

Allows only surface elements on the external boundary of the simulation domain (left, right, front, back) to be selected for a contact. By default, only surface elements at material interfaces and surface elements at the top and bottom of the simulation domain are selected for contacts. Default: false.

<solution>

Specifies the solution name for the contact. It retrieves information, such as current, regarding the specified solution name at the contact.

width

Width of the contact in micrometers.

`x, y, z`

Define the coordinates of a point for a region-type contact. If some coordinates of the point are omitted, the region is selected using the specified coordinates only. Default unit: μm .

`xlo, xhi, ylo, yhi, zlo, zhi`

Define the low and high values in each of the coordinate directions for a box-type contact. If some coordinates are omitted, the current bounds of the simulation domain are used. Default unit: μm .

Description

This command defines new contacts, deletes contacts, and prints contact information. The `contact` command can be called multiple times with the same name if the `add` argument is specified. In this case, the contact will have multiple parts.

NOTE Contacts are not transformed using the `transform` command.

NOTE If a TDR file containing a boundary or mesh is read into Sentaurus Interconnect (during the `init` command), contacts defined in this file are added to the list of contacts.

Examples

List all available contacts:

```
contact list
```

Define a contact named `gate`, which will consist of boundary elements of the region containing the point `(-0.05, 0.0)`. The region material will be replaced by gas:

```
contact name= gate x= -0.05 y= 0.0 replace point
```

Define a box-type contact containing the surface elements of an aluminum region inside the specified box:

```
contact box Aluminum xlo= -0.01 ylo= -0.46 xhi= 0.1 yhi= -0.16 name= source
```

Define the substrate contact at the bottom of the simulation domain, and switch off interior interfaces:

```
contact bottom name= substrate
```

A: Commands

contact

Define a contact named `lfcontact` on the left side (minimum y-coordinate) and the front (maximum z-coordinate) of the simulation domain for that part of the simulation domain inside the box (0,0,0) -> (1,1,1) and not on any interior interfaces:

```
contact left front name= lfcontact xlo= 0 ylo= 0 zlo= 0 xhi= 1 yhi= 1 zhi= 1
```

Merge contacts C1 and C2 so that the same supply value applies to both contacts:

```
contact merge= {C1 C2}
```

See Also

[integrate on page 467](#)

[line on page 481](#)

[struct on page 643](#)

contour

Plots a contour of the selected variable or named data field at the value specified on a 2D plot.

Syntax

```
contour
  [color=<c>] [name=<c>] [print] [value=<n>]
  [x=<n>] [<m>|<cm>|<um>|<nm>]
  [y=<n>] [<m>|<cm>|<um>|<nm>]
  [z=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

color

Specifies the line color of the contour. It can be any color supported by X11 hardware and named in the color database.

name

Name of the data field. It allows plots to be created without using the `select` command. Default: `Z_Plot_Var`.

print

Specifies that the contour values must be printed *not* plotted. The output is compatible with `xgraph`. In addition, a set of Tcl lists is returned.

value

Specifies the value at which the contour line should be plotted. If boron has been selected, a value of `1.0e16` would produce a line of constant boron concentration at that concentration.

x, y, z

Specify the plane on which contouring is performed. In two dimensions, they need not be specified. In three dimensions, two arguments must be specified to indicate the plane of calculation of the contour. Default unit: μm .

Description

The value must be specified in the range of the computed variable. For example, if plotting log boron, the value must be in the range 10 to 20 *not* `1e10` to `1e20`.

A: Commands

contour

The `contour` command assumes that the `plot.2d` command has been specified and that the screen is configured to plot a 2D graphic. If this has not been set, the routine most likely will produce unhelpful results.

Examples

Draw a line at an isoconcentration of 10^{10} :

```
contour value= 1e10
```

See Also

[Compatibility on page 380](#)

[plot.2d on page 549](#)

[slice on page 623](#)

crack

Defines a crack and its characteristic parameters.

Syntax

```
crack
  [angle.tolerance=<n>] [<degree>]
  [clear] [CZM=<c>]
  [CZM.box= {
    xmin= {<n0> [<m> | <cm> | <um> | <nm>] <n1> ... <nn> [...]}
    ymin= {<n0> [<m> | <cm> | <um> | <nm>] <n1> ... <nn> [...]}
    zmin= {<n0> [<m> | <cm> | <um> | <nm>] <n1> ... <nn> [...]}
    xmax= {<n0> [<m> | <cm> | <um> | <nm>] <n1> ... <nn> [...]}
    ymax= {<n0> [<m> | <cm> | <um> | <nm>] <n1> ... <nn> [...]}
    zmax= {<n0> [<m> | <cm> | <um> | <nm>] <n1> ... <nn> [...]}
  }]
  [CZM.polarity= positive | negative]
  [delete] [global] [list]
  [mat.1=<c>] [mat.2=<c>]
  [name=<c>]
  [normal= {<n1> <n2> <n3>}]
  [polygons= <list>] [print] [segments= <list>]
  [tolerance=<n>] [<m> | <cm> | <um> | <nm>]
  [x.min=<n>] [<m> | <cm> | <um> | <nm>] [x.max=<n>] [<m> | <cm> | <um> | <nm>]
  [y.min=<n>] [<m> | <cm> | <um> | <nm>] [y.max=<n>] [<m> | <cm> | <um> | <nm>]
  [z.min=<n>] [<m> | <cm> | <um> | <nm>] [z.max=<n>] [<m> | <cm> | <um> | <nm>]
```

Arguments

angle.tolerance

Sets the tolerance of a surface normal. A crack surface can be defined using `normal`. A surface is considered to be coplanar with the crack surface if the angle between the surface normal directions is within the tolerance.

clear

Clears all the crack definitions.

CZM

Specifies the name of the selected cohesive zone model (CZM), which must be defined using the `CZM` command before the simulation. To model initial cracks, use `CZM=none`.

A: Commands

crack

CZM.box

Specifies the x-, y-, and z-coordinates at the upper-left back corners and lower-right front corners of the cubes enclosing parts of the crack where CZM is active. Default unit: μm .

CZM.polarity

Sets the CZM to be active inside CZM.box when set to `positive` or outside CZM.box when set to `negative`. Default: `positive`.

delete

Deletes the crack definition specified by name.

global

Sets the flag for a massive cohesive zone definition. When `global` is specified, other arguments in the command are ignored, and all other crack definitions are removed. Cohesive surfaces are specified for all surfaces defined by element boundaries. Default: `false`.

list

Lists the names of all crack definitions.

mat.1, mat.2

Specify the material names on the two sides of a crack interface.

name

Sets the name of the crack.

normal

Sets the normal direction vector of the crack surface being defined. This normal direction must have been normalized (the square root of the sum of squares of three components is 1.0).

polygons

Specifies a 2D surface in three dimensions as a list of named polygons. The named polygons must have been defined using `polygon` commands (see [polygon on page 559](#)). The list of polygons must form a continuous 2D surface.

print

Prints information about the specified crack.

segments

Specifies a 1D line in two dimensions as a list of coordinates. Several coordinates can be specified at the same time. The first coordinate defines the beginning of a segment; the second coordinate defines the end of the segment; the third coordinate defines the beginning of the second segment, and so on. The list of coordinates must form a continuous 1D line. Coordinates are given in μm .

tolerance

Controls the snapping distance when processing the geometry features. Default: $1.0 \times 10^{-6} \mu\text{m}$.

x.max, y.max, z.max

Specify the x-, y-, and z-coordinates at the upper-left back corner of the cube enclosing the crack in the unified coordinate system (UCS). Default unit: μm .

x.min, y.min, z.min

Specify the x-, y-, and z-coordinates at the lower-right front corner of the cube enclosing the crack in the UCS. Default unit: μm .

Description

This command defines a crack with a name, location, CZM law, material interface, normal direction to the crack surface, and tolerance.

Examples

Define an initial crack along the interface between silicon and oxide. The crack is bounded by the specified minimum and maximum coordinates in the x-, y-, and z-directions with a snapping distance:

```
crack name= SiliconOxide mat.1= Silicon mat.2= Oxide CZM= none \  
  tolerance= 1.0e-6 x.min= -0.1 x.max= -0.05 y.min= -0.1 y.max= -0.05 \  
  z.min= 0.0 z.max= 0.4
```

Define a V-notch in aluminum for crack analysis. The crack surface and the crack front are bounded by the specified minimum and maximum coordinates in the x-, y-, and z-directions with a snapping distance:

```
crack name= AluminumNotch mat.1= Gas mat.2= Aluminum CZM= none \  
  angle.tolerance= 1.0e-3 tolerance= 1.0e-3 normal= "0.0 -1.0 0.0" \  
  x.min= -0.1 x.max= -0.05 y.min= -0.1 y.max= -0.05 z.min= 0.0 z.max= 0.4
```

A: Commands

crack

Define a bulk crack using segments in two dimensions. The first pair of coordinates is x and y of the beginning of the segment, and the second pair of coordinates is the end of the segment. The crack must be confined to one region and must not intersect another crack:

```
crack name= BulkCrack1 mat.1= Silicon segments= {3.0 1.2 1.0 4.0} \  
angle.tolerance= 1.0e-2
```

Define a bulk crack using polygons in three dimensions, where P1, P2, and P3 are predefined polygons. The crack must be confined to one region and must not intersect another crack. The polygons must form a continuous 2D surface:

```
crack name= BulkCrack2 mat.1= Silicon polygons= {P1 P2 P3} \  
angle.tolerance= 1.0e-2
```

NOTE Bulk cracks require that contacts be defined in the boundary representation (brep) mode (`pdbSet Grid Contacts.In.Brep 1`). If not, the brep mode will be switched on. In addition, the structure in two dimensions must use the brep mode (for example, `pdbSet Grid MGoals use.brep.2d 1`); otherwise, it will be switched on.

See Also

[CZM on page 396](#)

[polygon on page 559](#)

current_ramp

Defines a current profile for use with the `supply` command.

Syntax

```
current_ramp
  (clear | list | name=<c>)
  [c.final=<n>] [<V>]
  [current=<n>] [<A>]
  [current.time=<n>] [<hr>|<min>|<s>]
  [delC=<n> [<A>] | delNC=<n> [<A>]]
  [hold] [last]
  [ramprate=<n>] [<A/s>|<A/min>]
  [reset.init.time]
  [time=<n>] [<hr>|<min>|<s>]
```

Arguments

`c.final`

Final current for a voltage ramp-up or ramp-down. It is used if `ramprate` is not specified. The ramp time is calculated automatically.

`clear`

Clears the global list of current ramps. When defining profiles, the action is to unite the new definition with any prior profiles of the same name.

`current`

Simulation current.

`current.time`

Returns the value of the ramp for the given time.

`delC`

Defines the maximum current step during a current ramp-up if specified.

`delNC`

Defines the maximum current step during a current ramp-down if specified.

`hold`

During this segment, allows the `solve` command time to specify the time of the segment.

A: Commands

current_ramp

last

Defines the final component of the current profile. There will be no more additions to the ramp.

list

Generates a list of current profiles. It returns a Tcl list and can be operated on as such. The default action for commands is to print the return, so if no handling is required, `list` prints a list of names of defined current profiles. If `name` is specified, `current_ramp` only is listed along with details about the ramps.

name

Name of the parameter used to identify the current ramp and to specify it in a supply command.

ramprate

Specifies the current change during the simulation. Default value and unit: 0 A/s.

reset.init.time

Starts each current step with the same initial time step.

time

Simulation time. Default unit: minute.

Description

This command specifies multiple-step current ramps and holds. It can be used to construct a complex current sequence to be simulated with the `solve` command.

Examples

Define the current profile named `cr1` with a current rate of 0.5e-3 V/s:

```
current_ramp name= cr1 current= 0.1 ramprate= 0.5e-3<A/s> time= 1<s>
```

See Also

[solve on page 629](#)

[supply on page 647](#)

CutLine2D

Computes the slice angle when given a cut in wafer coordinates.

Syntax

```
CutLine2D <x1> <y1> <x2> <y2>
```

Arguments

<x1> <y1> <x2> <y2>

Endpoints of the simulation cutline in wafer coordinates.

Description

Given a cut in wafer coordinates defined by the endpoints (<x1>, <y1>) and (<x2>, <y2>), the CutLine2D command computes the slice angle.

Examples

Set the cutline for the simulation from (0,0) to (1,0):

```
init slice.angle= [CutLine2D 0 0 1.0 0]
```

See Also

[Wafer Coordinate System on page 31](#)
[init on page 458](#)

CZM

Defines the parameters for a cohesive zone model (CZM).

Syntax

```
CZM
[clear] [delete] [law=<c>] [list]
[mat.1=<c>] [mat.2=<c>]
[name=<c>]
[normal.critical.length=<n>] [<um>|<cm>|<m>|<nm>]
[normal.stress.max=<n>] [<atm>|<GPa>|<MPa>|<KPa>|<Pa>|<dyn/cm2>]
[normal.toughness=<n>] [<N/m>|<dyn/cm>]
[print]
[shear.critical.length=<n>] [<um>|<cm>|<m>|<nm>]
[shear.stress.max=<n>] [<atm>|<GPa>|<MPa>|<KPa>|<Pa>|<dyn/cm2>]
[shear.toughness=<n>] [<N/m>|<dyn/cm>]
```

Arguments

clear

Clears all CZM definitions.

delete

Deletes the CZM definition specified by name.

law

Specifies a cohesive law: `exponential` (default) or `triangular`.

list

Lists the names of all CZM definitions.

mat.1, mat.2

Specify the material names on the two sides of the interface for the CZM. `mat.1` and `mat.2` can be identical. If either `mat.1` or `mat.2` is omitted, the CZM command defines a CZM at element boundary surfaces within the same material.

name

Name of the cohesive law. A CZM defined by the name `default` will be applied to any element interface that has no CZM defined.

`normal.critical.length`

Sets the critical normal opening distance at the maximum normal material strength for the triangular law. If not specified, the critical normal opening distance is set to half of the maximum normal opening. Default unit: μm .

`normal.stress.max`

Sets the maximum material strength in the normal direction. Default unit: dyn cm^{-2} .

`normal.toughness`

Sets the work of separation in the normal direction. Default unit: dyn cm^{-1} .

`print`

Prints information about the specified CZM.

`shear.critical.length`

Sets the critical shear opening distance at the maximum shear material strength for the triangular law. If not specified, the critical shear opening distance is set to half of the maximum shear opening. Default unit: μm .

`shear.stress.max`

Sets the maximum material strength in the shear direction. Default unit: dyn cm^{-2} .

`shear.toughness`

Sets the work of separation in the shear direction. Default unit: dyn cm^{-1} .

Description

This command defines the parameters (material, toughness, and maximum stresses for both normal and shear directions) for a CZM.

Examples

Define an exponential CZM for cohesive surfaces within material Oxide. The material strength and the work of separation are given in both the normal and shear directions:

```
CZM name= CzmOxide law= exponential mat.1= oxide \  
normal.stress.max= 1.0e3<MPa> shear.stress.max= 1.2e3<MPa> \  
normal.toughness= 5<N/m> shear.toughness= 6<N/m>  
normal.critical.length= 3e-3<um> shear.critical.length= 3e-3<um>
```

define

Defines a Tcl variable.

Syntax

```
define <name> <value>
```

Arguments

<name>

Any user-defined parameter name.

<value>

Any number or string value.

Description

The `define` command is equivalent to the Tcl command `set`, except that variables defined with `set` are *not* saved or re-stored in TDR files. Variables defined using the `define` command are saved or re-stored.

The `define` and `fset` commands are equivalent.

Examples

Define the Tcl variable `LG`, which is stored in and loaded from a TDR file. It can be used in any Tcl expression:

```
define LG 0.02
```

See Also

Tcl documentation for description of `set` syntax

defineproc

Defines a Tcl procedure.

Syntax

```
defineproc <name> { <procedure_arguments> } {  
    <body_of_procedure>  
}
```

Arguments

<name>

Name of the Tcl procedure.

<procedure_arguments>

Lists the arguments of the named Tcl procedure.

<body_of_procedure>

Describes the Tcl procedure.

Description

The `defineproc` command is equivalent to the Tcl command `proc`, except that procedures defined with `proc` are *not* saved or re-stored in TDR files. Procedures defined using `defineproc` are saved or re-stored.

The `defineproc` and `fproc` commands are equivalent.

Examples

Define the Tcl procedure `relerr`, which is stored in and loaded from a TDR file:

```
defineproc relerr { newVal RefVal name my_err } {  
    upvar my_err fl  
    set denom [ expr abs($newVal)+abs($RefVal)+1e-20 ]  
    set deviation [expr 100*abs(($RefVal - $newVal)/$denom)]  
    if { $deviation > 0.5 } {  
        LogFile ILO "Compare: $name= $newVal, ref= $RefVal, relerr= $deviation  
            \n --> failed\n"  
        set fl [ expr $fl+1 ]  
    } else {  
        LogFile ILO "Compare: $name= $newVal, ref= $RefVal, relerr= $deviation  
            ok\n"  
    }  
}
```

A: Commands

DeleteRefinementboxes

```
}  
}
```

See Also

Tcl documentation for description of `proc` syntax

DeleteRefinementboxes

Deletes a set of refinement boxes based on a pattern.

Syntax

```
DeleteRefinementboxes pattern=<c>
```

Arguments

pattern

Specifies the pattern to use.

Description

This command finds all the refinement boxes with names that match the defined pattern and deletes them. The pattern is expanded according to standard Tcl rules.

Examples

Delete all refinement boxes that have names such as `root_1`, `root_2`, and `root_3`:

```
DeleteRefinementboxes pattern= "root*"
```

deposit

Deposits a new layer.

Syntax

```

deposit
  ( [<material>] [anisotropic | crystal | fill | fourier | isotropic]
    [coord=<n>] [<m>|<cm>|<um>|<nm>]
    [thickness=<n>] [<m>|<cm>|<um>|<nm>] )
  [1D] [Adaptive] [angle=<n>]
  [coeffs= {<A0> <A1> <A2> ... <An>}]
  [crystal.rate= {"<100>" =<n> "<110>" =<n> "<111>" =<n> "<311>" =<n>
    "<511>" =<n> "<911>" =<n> "<221>" =<n>}]
  [direction= <list>]
  [doping= <list>]
  ([<fieldname>] | [species=<c>]
    [concentration=<n>] [<m-3>|<cm-3>|<um-3>|<nm-3>])
  [fields.values= <list>]
  [fill.buried] [fill.buried.interface=<material>]
  [fill.buried.match.surrounding]
  [force.full.levelset]
  [mask=<c>]
  [mat.coeffs= {
    <mat1>= {<A0> <A1> <A2> ... <An>}
    <mat2>= {<A0> <A1> <A2> ... <An>}
    ...
    <matn>= {<A0> <A1> <A2> ... <An>} }]
  [material=<c>]
  [polygon= <list>]
  [rate=<n> [<um/min>]]
  [region.name=<c>]
  [remesh] [repair]
  [sde=<c>]
  [selective.materials= <list>]
  [shadowing] [shadowing.nonisotropic]
  [sources= {<beam1> <beam2> ... <beamn>}]
  [steps=<n>] [Strained.Lattice]
  [suppress.remesh]
  [temperature=<n>] [<C>|<K>]
  [time=<n>] [<hr>|<min>|<s>]
  [type= anisotropic | crystal | directional | fill | fourier | isotropic |
    polygon | trapezoidal]

```

A: Commands

deposit

Arguments

1D

Usually, a polygon deposition automatically increases the dimension to two dimensions before performing the operation. Specify 1D to prevent this behavior.

Adaptive

If specified, `Adaptive` switches on adaptive meshing for deposition. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

angle

Specifies the sidewall angle in degrees when using `type=trapezoidal`. An angle of 90° is vertical; an angle $> 90^\circ$ spreads outward with increasing deposited thickness; and an angle $< 90^\circ$ closes inward with increasing deposited thickness. Default unit: degree.

anisotropic, crystal, fill, fourier, isotropic

Specify the type of deposition:

- `anisotropic` defaults to vertically downward deposition.
- `crystal` specifies crystal deposition.
- `fill` is used to fill the structure with the specified material up to the specified coordinate.
- `fourier` specifies Fourier-type deposition. When using Fourier deposition, the coefficients must be specified using `coeffs` or `mat.coeffs`.
- `isotropic` implies the same rate in all directions.

These deposition types must be used with `thickness` or `coord`, *not* with `rate` and `time`.

coeffs

List of single-material coefficients used in Fourier deposition.

coord

Specifies the x-coordinate for `type=fill`. Default unit: μm .

crystal.rate

List of deposition rates defined per crystallographic direction in the format:

```
{ "<100>" =<depo_rate> "<110>" =<depo_rate> "<111>" =<depo_rate> }
```

There is no default value for any undefined rate in a given crystallographic direction.

`direction`

Specifies the direction for directional deposition as a list of x- and y-coordinates of the deposition vector. The x-coordinate must be positive. Positive-y indicates a right-pointing deposition beam, and negative-y indicates a left-pointing beam.

`doping`

List of names of doping profiles that have been previously defined with the `doping` command.

`<fieldname>, species, concentration`

These arguments allow a doped layer to be deposited. `species` specifies the name of the data field to be incorporated (you can add a new user species this way). Instead of specifying `species`, you can specify a field name (for example, boron, arsenic, phosphorus, and indium). The default value and unit for `concentration` is 10^{10} cm^{-3} .

`fields.values`

List of parameters where the parameter name is the name of the field to be introduced in the deposited layer, and the value is the initial value, for example, `fields.values={Boron=1e18}`. A list of fields of any name can be initialized with this argument and, for solution variables or stress components, units are accepted.

`fill.buried`

By default, the material is deposited on the surface exposed to the upper gas region. If the structure has buried gas bubbles, they are untouched. Use `fill.buried` to deposit the deposition material inside those gas bubbles.

`fill.buried.interface`

Specifies the material with which to fill gas bubbles present at a material interface.

NOTE The specified material must already be present in the simulation (for example, silicon).

`fill.buried.match.surrounding`

Similar to `fill.buried`, but instead of depositing the deposition material inside gas bubbles, this option fills gas bubbles that are completely surrounded by a single material with that material, removing the bubbles from the domain entirely. Gas bubbles at material interfaces remain unfilled but can be filled using `fill.buried.interface`.

NOTE This option is available only for 3D deposition.

A: Commands

deposit

`force.full.levelset`

By default, the simplest algorithm is chosen to perform the deposition. However, sometimes the algorithm chosen generates incorrect results if the topology of the structure is complicated. Specifying `force.full.levelset` switches on the general level-set time-stepping algorithm that correctly handles these structures.

`mask`

Name of a mask to be used for the deposition.

NOTE The material is deposited outside of the mask. If deposition inside the mask is required, `negative` must be specified in the `mask` command.

`mat.coeffs`

List of multimaterial coefficients A_0, A_1, \dots, A_n used in Fourier deposition with a different set of coefficients defined for each material.

`<material>`

Allows the specification of the deposited material. See [Specifying Materials on page 18](#).

`material`

Specifies the material to be deposited. Overrides the `<material>` specification.

`polygon`

Specifies a list of x- and y-coordinates for the deposition. This is used only for 2D deposition with `type=polygon`. The list of coordinates must define a single polygon with no self-intersections. The first and last points are connected implicitly to close the polygon. The specified material is deposited inside the polygon. The default unit for the coordinates is μm .

`rate`

Deposition rate. Default unit: $\mu\text{m}/\text{minute}$.

`region.name`

Name of the region created by the `deposit` command. The name must not contain an underscore (`_`) or a period (`.`) because these characters have special meaning. The name must be different than an existing material name (see [Regionwise Parameters and Region Name-Handling on page 24](#)).

`remesh`

Performs a remeshing after the deposition.

repair

In 3D MGOALS mode, small regions are removed automatically by default. Sometimes, this causes small gas bubbles in the structure or other problems. Use `!repair` to switch off the small region removal.

sde

Specifies the arguments and algorithms for 3D Sentaurus Structure Editor. By default, arguments such as `rate`, `thickness`, `time`, and `type` are translated into appropriate Sentaurus Structure Editor commands.

If an algorithm is specified, it overwrites the algorithm used by default for isotropic or anisotropic deposition, for example:

```
sde= {"algorithm" "lop"}  
sde= {"algorithm" "lop" "radius" 0.07}
```

selective.materials

Specifies that deposition will occur only on a list of selected materials. For 2D simulations, MGOALS is used and multiple materials can be selected. For 3D simulations, the Sentaurus Structure Editor interface is called and only one material can be selected. In either case, only one material can be deposited.

shadowing

In two dimensions, switches on the inclusion of shadowing effects if `force.full.levelset` is specified or for Fourier deposition. The visibility of each surface area to each beam is calculated at every level-set time step. In 3D MGOALS mode, this argument enables shadowing effects on both directional and anisotropic deposition. The interface to Sentaurus Structure Editor ignores `shadowing`.

shadowing.nonisotropic

Use instead of `shadowing` to allow the 0th-order Fourier coefficient to deposit in areas where the beam is shadowed.

sources

Defines the source beams for level-set deposition.

steps

Subdivides a deposition into more than one step. If necessary, stress relaxation is calculated at the end of each step. Default: 1.

Strained.Lattice

Specifies strained deposition.

A: Commands

deposit

`suppress.remesh`

Suppresses mesh generation and stress rebalance operations *before* 3D deposition steps. It also suppresses mesh generation and stress rebalance *after* a 3D deposition, except when initializing fields in the newly deposited layer. See [Suppressing Mesh Generation in 3D Simulations on page 237](#).

`temperature`

Deposition temperature used for stress relaxation only. Default value and unit: 26.85°C.

`thickness`

Thickness of the deposited layers. Default unit: μm .

`time`

Deposition time. It must be specified if the `rate` argument is used. Default unit: minute.

`type`

Explicitly specifies the type of deposition to be performed:

- `type=anisotropic` performs deposition in the vertical direction only, which must be used with the `rate` and `time` arguments.
- `type=crystal` performs crystallographic deposition. The `crystal.rate` argument also must be specified.
- `type=directional` performs anisotropic deposition using a specified direction.
- `type=fill` performs a fill of a specified material up to the coordinate specified with the `coord` argument.
- `type=fourier` performs Fourier deposition, which requires the coefficients to be specified with either `coeffs` or `mat.coeffs`.
- `type=isotropic` performs isotropic deposition, which must be used with the `rate` and `time` arguments.
- `type=polygon` performs a polygonal deposition (in two dimensions), which requires the `polygon` argument.
- `type=trapezoidal` performs a trapezoidal deposition, which requires the `angle` and `thickness` arguments, and using `selective.materials` is recommended. This is supported in 3D deposition only.

Description

This command simulates a deposition step.

Examples

Isotropic deposition of a 0.2 μm oxide layer:

```
deposit thickness= 0.2 oxide isotropic
```

Same as above; thickness is defined by rate and time:

```
deposit rate= {0.2} material= {oxide} type= isotropic time= 1
```

Add an intrinsic isotropic stress of 10^9 Pa to the deposited nitride layer before the post-deposition mechanics rebalancing step:

```
deposit thickness= 0.1 nitride \  
  fields.values= {StressELXX= 1e9<Pa> StressELYY= 1e9<Pa> \  
  StressELZZ= 1e9<Pa>}
```

Deposit $\text{Si}_{0.8}\text{Ge}_{0.2}$:

```
deposit thickness= 0.1<um> SiliconGermanium \  
  fields.values= [MoleFractionFields SiGe 0.2]
```

See Also

[doping on page 408](#)

[mask on page 493](#)

[mgoals on page 510](#)

doping

Defines a named piecewise linear doping profile that can be used with the `deposit` command.

Syntax

```
doping
  field=<c> name=<c>
  (depths= <list> | times= <list>)
  stress.values= <list>
  values= <list>
  [clear] [list] [location= vertex | element] [log.grad]
```

Arguments

`clear`

Clears all doping specifications.

`depths`, `times`

You can use either:

- `depths` to specify a numeric list of the depths at which the `values` are applied. Default unit: μm .
- `times` to specify a numeric list of the times at which the `values` are applied. This argument only works for crystallographic deposition. The unit is minutes and cannot be changed.

Only `depths` or `times` can be specified, but not both at the same time.

`field`

Name of the field. For stresses, use the field names:

- `StressELXX`
- `StressELXY`
- `StressELYY`
- `StressELZZ`
- `StressELXZ`
- `StressELYZ`

It is not necessary to specify all stress components. Those not specified are assumed to be zero initially and are updated during the stress rebalance at the end of deposition.

`list`

Returns the names of all doping specifications.

`location`

Location where the field is to be applied. Default: `vertex`.

`log.grad`

Specifies a piecewise logarithmic doping profile.

`name`

Name of the profile to be used in the `deposit` command.

`stress.values`

Numeric list of the values of the stress field. Default unit: dyn/cm^2 .

`values`

Numeric list of the values of the data field. Default unit: cm^{-3} .

Description

This command allows a doping profile specification that can be used inside the `deposit` command to add doping and other fields to the newly deposited layer (on either vertices or elements).

Examples

Create a doping profile definition with the name `init_boron` that consists of a boron profile linearly increasing from 1×10^{10} at the starting surface to 1×10^{20} at $0.1 \mu\text{m}$ and beyond in the deposited layer. This doping profile definition can be used with the `deposit` command to create the specified profile:

```
doping name= init_boron field= Boron values= {1e10 1e20} depths= {0 0.1}
```

Add an intrinsic isotropic stress of 10^9dyn/cm^2 to the deposited layer before the post-deposition mechanics rebalancing step:

```
doping name= film_Sxx field= StressELXX stress.values= {1e9 1e9} depths= {0 1}  
doping name= film_Syy field= StressELYY stress.values= {1e9 1e9} depths= {0 1}  
doping name= film_Szz field= StressELZZ stress.values= {1e9 1e9} depths= {0 1}
```

See Also

[deposit on page 401](#)

element

Extracts the grid for a specified material and returns the grid as a list of coordinates.

Syntax

```
element <material> [region]
```

Arguments

<material>

Name of the material. See [Specifying Materials on page 18](#).

region

Limits output to only one region if specified.

Description

This command can be used to plot the grid. It returns a list of coordinates that define the grid. Each of the outer lists makes up a continuous line through the grid. Each inner list contains coordinate pairs in order for that line.

NOTE This command is not available for 3D simulations.

Examples

Return the grid of oxide material:

```
element oxide
```


Enu2G

Computes the shear modulus from Young's modulus and the Poisson ratio.

Syntax

```
Enu2G <n> <n>
```

Arguments

<n>

The first value is Young's modulus.

The second value is the Poisson ratio.

Description

The same units are assumed for all moduli.

Examples

Compute the shear modulus from Young's modulus ($1.620e12$ dyn/cm²) and the Poisson ratio (0.28):

```
Enu2G 1.620e12 0.28
```

Enu2K

Computes the bulk modulus from Young's modulus and the Poisson ratio.

Syntax

```
Enu2K <n> <n>
```

Arguments

<n>

The first value is Young's modulus.

The second value is the Poisson ratio.

Description

The same units are assumed for all moduli.

Examples

Compute the bulk modulus from Young's modulus ($1.620e12$ dyn/cm²) and the Poisson ratio (0.28):

```
Enu2K 1.620e12 0.28
```

equation

Allows test parsing and resolution of an equation string.

Syntax

```
equation eqn=<c> [nodal]
```

Arguments

eqn

String to be checked.

nodal

If specified, `nodal` returns the nodal part of the string specified by `eqn`.

Description

The equation string is parsed, broken into pieces, and derivatives are taken and printed. This command is useful for debugging problems with the resolver and parsing, as equation strings can be tried before being run.

Examples

Parse and resolve the string `exp(Potential*$Vti)`:

```
equation eqn= "exp(Potential*$Vti) "
```

See Also

[solution on page 626](#)

etch

Removes part or all of an exposed layer.

Syntax

```
etch
  [1D] [Adaptive] [ambient.rate=<n>] [angle=<n>]
  [angles.rates= {
    matA= {angleA0 rateA0 angleA1 rateA1 ... angleAn rateAn}
    matB= {angleB0 rateB0 angleB1 rateB1 ... angleBn rateBn}
    ... }]
  [anisotropic | cmp | isotropic | trapezoidal]
  [bottom.angle=<n>] [bottom.thickness=<n>]
  [coeffs= {<A0> <A1> <A2> ... <An>}]
  [coord=<n>] [<m>|<cm>|<um>|<nm>]
  [crystal.rate= {"<100>" =<n> "<110>" =<n> "<111>" =<n> "<311>" =<n>
    "<511>" =<n> "<911>" =<n> "<221>" =<n>}]
  [direction= <list>]
  [etchstop= {<mat1> <mat2> ...} [etchstop.overetch=<n>]]
  [force.full.levelset]
  [isotropic.overetch=<n>]
  [levelset.upwind]
  [mask=<c>]
  [mat.coeffs= {
    <mat1>= {<A0> <A1> <A2> ... <An>}
    <mat2>= {<A0> <A1> <A2> ... <An>}
    ...
    <matn>= {<A0> <A1> <A2> ... <An>} }]
  [<material>]
  [material= <list>]
  [polygon= <list>]
  [rate= <list>]
  [remesh] [repair] [roundness=<n>]
  [sde=<c>]
  [shadowing] [shadowing.nonisotropic]
  [sources= {<beam1> <beam2> ... <beamn>}]
  [steps=<n>]
  [suppress.remesh]
  [temperature=<n>] [<C>|<K>]
  [thickness=<n>] [<m>|<cm>|<um>|<nm>]
  [time=<n>] [<hr>|<min>|<s>]
  [type= anisotropic | cmp | cmp.flat | crystal | directional | fourier |
    isotropic | polygon | trapezoidal]
  [undercut=<n>]
```

Arguments

`1D`

Usually, a polygon etching automatically increases the dimension to two dimensions before performing the operation. Set `1D` to prevent this behavior.

`Adaptive`

If specified, `Adaptive` switches on adaptive meshing for this etching step. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

`ambient.rate`

Used with `type=trapezoidal` and `force.full.levelset` to approximate the underetch effect. Default unit: $\mu\text{m}/\text{minute}$.

`angle`

Specifies the etching angle for `type=trapezoidal`. Default unit: degree.

`angles.rates`

Specifies a set of angle rate pairs that gives a piecewise linear etch rate versus angle. To be stable, smaller angles must have a higher etch rate. Specify the etching rate to go to zero above some angle produces facets.

`anisotropic, cmp, isotropic, trapezoidal`

Specifies the type of etching:

- `anisotropic`: Anisotropic etching.
- `cmp`: Chemical-mechanical polishing (CMP).
- `isotropic`: Isotropic etching.
- `trapezoidal`: Trapezoidal etching.

`bottom.angle`

Specifies the angle for the bottom of a trapezoidal etching in three dimensions. Default unit: degree.

`bottom.thickness`

Specifies the thickness for the bottom of a trapezoidal etching in three dimensions. Default unit: μm .

A: Commands

etch

coeffs

List of single-material coefficients A_0, A_1, \dots, A_n used in Fourier etching.

coord

The x-coordinate to work with `type=cmp`. Default unit: μm .

crystal.rate

List of etching rates defined per crystallographic direction in the format:

```
{ "<100>" =<etch_rate> "<110>" =<etch_rate> "<111>" =<etch_rate> }
```

You must add a space between the double quotation mark (") after the crystallographic direction and the equal sign. There is no default value for any undefined rate in a given crystallographic direction. Default unit: $\mu\text{m}/\text{minute}$.

direction

Numeric list of x-, y-, and z-values specifying the etching direction for `type=directional`.

NOTE All three values must always be specified. In two dimensions, the z-value must always be zero.

etchstop

Materials, instead of time, can be given as etch-stopping criteria. In this case, etching continues until any of the given `etchstop` materials is exposed. An additional overetch is performed, with a time equal to `etchstop.overetch` (default: 10%) multiplied by the accumulated time required to expose the first `etchstop` material.

In three dimensions, `etchstop` is ignored when Sentaurus Structure Editor is used.

NOTE Materials, in addition to time, can be given as etch-stopping criteria for Fourier etching. If both `time` and `etchstop` are given, Fourier etching stops when either of the two criteria is first met.

etchstop.overetch

When `etchstop` is defined, an additional overetch is performed when the first `etchstop` material becomes exposed to gas. The duration of this overetch step is the already performed etching time multiplied by `etchstop.overetch`. Default: 0.1 (10%).

force.full.levelset

By default, the simplest algorithm is chosen to perform the etching. However, sometimes, the algorithm chosen generates incorrect results if the topology of the structure is

complicated. Specifying `force.full.levelset` switches on the general level-set time-stepping algorithm, which correctly handles these structures.

`isotropic.overetch`

Specifies a required amount of isotropic etching following anisotropic etching. The thickness is specified as a fraction of the anisotropic component. This argument is not implemented in 3D MGOALS mode.

`levelset.upwind`

Used with `force.full.levelset` to choose the Upwind formulation of the full level-set algorithm. The Upwind algorithm is less stable and less robust than the Lax–Friedrichs algorithm, which is the default.

`mask`

Name of the mask to be used for the etching.

`mat.coeffs`

List of multimaterial coefficients A_0, A_1, \dots, A_n used in Fourier etching with a different set of coefficients defined for each material.

`<material>`

Specifies the material to be etched. See [Specifying Materials on page 18](#).

`material`

String list of materials for multimaterial etching.

`polygon`

Numeric list of x- and y-coordinates for `type=polygon`. The list of coordinates $\{x_0 \ y_0 \ x_1 \ y_1 \ x_2 \ y_2 \ \dots \ x_n \ y_n\}$ defines a single polygon that must not be self-intersecting. The first and last points are connected implicitly to close the polygon. The material inside the polygon is etched. The default unit of the coordinates is μm .

`rate`

Numeric list of etching rates. Default unit: $\mu\text{m}/\text{minute}$.

`remesh`

Performs remeshing after the etching.

A: Commands

etch

repair

In 3D MGOALS mode, small regions are removed automatically by default. Sometimes, this causes small gas bubbles in the structure or other problems. Use `!repair` to switch off the small region removal.

roundness

Tuning argument for the curvature of the etch sidewalls in the case of trapezoidal etching when `force.full.levelset` is used. The default value is 1.0. Increased values up to 2.0 or 3.0 increase the curvature of the etch sidewall calculated by the level-set solver.

sde

String used to specify parameters and algorithms for 3D Sentaurus Structure Editor. By default, arguments such as `rate`, `thickness`, `time`, and `type` are translated into appropriate Sentaurus Structure Editor commands. If an algorithm is specified using `sde`, it overwrites the algorithm used by default for isotropic or anisotropic etching, for example:

```
sde= {"algorithm" "lopx"}  
sde= {"algorithm" "lopx" "radius" 0.07}
```

shadowing

In two dimensions, `shadowing` switches on the inclusion of shadowing effects if `force.full.levelset` is specified or for Fourier etching. The visibility of each surface area to each beam is calculated at every level-set time step.

In 3D MGOALS mode, `shadowing` enables shadowing effects on both directional and anisotropic etching.

The Sentaurus Structure Editor interface ignores `shadowing`.

shadowing.nonisotropic

Used instead of `shadowing` to allow the 0th-order Fourier coefficient to etch areas where the beam is shadowed.

sources

Defines the etching source beams for level-set etching.

steps

Subdivides an etching operation into more than one step. If necessary, stress relaxation is calculated at the end of each step. Default: 1.

suppress.remesh

Suppresses mesh generation and stress rebalance operations before and after 3D etching steps (see [Suppressing Mesh Generation in 3D Simulations on page 237](#)).

temperature

Etching temperature. Default value and unit: 26.85°C .

thickness

Thickness that is removed in the etching. Default unit: μm .

time

Refers to the etching time. It must be specified if `rate` is used. Default value and unit: 1.0 minute.

NOTE If both `time` and `etchstop` are given for Fourier etching, the Fourier etching stops when either of these two criteria is met.

type

Specifies the type of etching to be performed:

- `type=anisotropic` performs etching in the vertical direction only and must be used with the `rate` and `time` arguments.
- `type=cmp` performs CMP and is used with the argument `coord`.
- `type=cmp.flat` performs CMP, except the new surface specified by `coord` is flattened following stress relaxation and has zero displacement.
- `type=crystal` performs etching in two dimensions or three dimensions using etching rates dependent on the crystallographic direction defined by the `crystal.rate` argument.
- `type=directional` performs anisotropic etching in other directions and must be used with the `direction`, `rate`, and `time` arguments.
- `type=fourier` performs angle-dependent etching in two dimensions or three dimensions where the rate-versus-angle functions are defined by a cosine series using `coeffs` or `mat.coeffs`.
- `type=isotropic` performs isotropic etching, which must be used with the `rate` and `time` arguments.
- `type=polygon` performs polygonal etching in two dimensions and is used with the `polygon` argument.
- `type=trapezoidal` performs trapezoidal etching like Taurus™ TSUPREM-4™ in two dimensions defined by the `thickness`, `undercut`, and `angle` arguments, and in three dimensions defined by the `thickness`, `angle`, `bottom.thickness`, and `bottom.angle` arguments.

A: Commands

etch

undercut

Distance to etch below the nonetchable material in 2D trapezoidal etching. Default unit: μm .

Description

This command etches a layer exposed to the top gas. Several materials can be etched at the same time. There are different modes to perform etching:

- The MGOALS mode uses either an analytic or a level-set method performed by the MGOALS module.
- A general level-set time-stepping mode can handle more sophisticated etching capabilities such as multimaterial etching, Fourier etching, multiple beam, and shadowing.

Examples

Etch a $0.2\ \mu\text{m}$ silicon layer anisotropically in the direction indicated by `direction`. A mask called `m1` is used during etching:

```
etch time= 2.0 rate= {0.1} material= {silicon} type= directional \  
    direction= {1.0 1.0 0.0} mask= m1
```

Etch silicon for 0.1 minute at a rate of $1\ \mu\text{m}$ per minute, using `source1` as the etching beam, including shadowing effects, and with a Fourier response to the etchant defined by $A_0 = 0.1$ and $A_3 = 1$:

```
beam name= source1 factor= 1.0 incidence= -30  
  
etch material= {silicon} shadowing sources= {source1} type= fourier \  
    coeffs= {0.1 0 0 1} time= 0.1
```

Define multimaterial Fourier etching for 0.2 minutes. The Fourier coefficients for each material are given separately within `mat.coeffs`:

```
beam name= source_beam factor= 1.0 incidence= -30  
  
etch material= {Silicon Nitride Oxide PolySi} sources= {source_beam} \  
    mat.coeffs= { Silicon= {0 0.5} Nitride= {0 1} Oxide= {0 0.75} \  
    PolySilicon= {0.5} } type= fourier time= 0.2
```

See Also

[deposit on page 401](#)

[mask on page 493](#)

[mgoals on page 510](#)

exit

Terminates the execution of Sentaurus Interconnect.

Syntax

```
exit
```

Description

This command can be used in interactive mode as well as in command files.

See Also

[fbreak on page 424](#)

[fcontinue on page 424](#)

extract

Extracts historical data during a solve step.

Syntax

```
extract
  [clear]
  [command= {<c> <c> ...}]
  [name=<c>] [print]
  [syntax.check.value=<c>]
```

Arguments

clear

Clears the stored historical data.

command

List of commands for data interpolation.

name

Name of data extraction.

print

Returns the extracted data values as a Tcl list with all interpolated variables.

syntax.check.value

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

Description

Specifies the commands for data extraction during a solve step.

The extracted historical data can be returned as a Tcl list. Typically, the extracted data is in internal units. Internal units are the centimeter-gram-second (CGS) system of units [s, cm, g, dyn/cm², poise, cm²/s]. For example, the unit for stress is dyn/cm².

Examples

Extract and store the value of `Potential` at the position `0.04 μm` in the silicon for each solve step:

```
extract name= etest command= {select z= Potential interpolate Silicon x= 0.04}
```

Return the extracted data values with the extraction name `etest` in a Tcl list (for example, if two values are extracted in `etest`, the Tcl list takes the format `<time1> <value1_1> <value2_1> <time2> <value1_2> <value2_2> ...`):

```
extract print name= etest
```

Clear all stored historical data:

```
extract clear
```

See Also

[Extracting Values During solve Step: extract on page 324](#)
[interpolate on page 473](#)

fbreak

Starts interactive mode.

Syntax

```
fbreak
```

Description

This command interrupts the execution of command files and starts the interactive mode.

See Also

[exit on page 421](#)

[fcontinue](#)

fcontinue

Resumes execution of command files.

Syntax

```
fcontinue
```

Description

This interactive mode command resumes the execution of command files in batch mode.

See Also

[exit on page 421](#)

[fbreak](#)

fexec

Executes system commands.

Syntax

```
fexec
```

Description

Executes system calls through the Tcl command `exec` (with exactly the same syntax). Using `fexec`, the system calls are not executed during syntax-checking as they would be if the plain `exec` command were used. If there is an error in the execution, this command prints an `Error in system call` message, in addition to the error output of `exec`.

Examples

List the contents of the current directory:

```
fexec ls
```

See Also

Tcl documentation for description of `exec` syntax

fproc

The `defineproc` and `fproc` commands are equivalent.

See [defineproc on page 399](#).

fset

The `define` and `fset` commands are equivalent.

See [define on page 398](#).

generic_step

Define a process step in a generalized way.

Syntax

```
generic_step
  [<material>]
  [<par1>=<c>] [<par2>=<c>] ... [<parn>=<c>]
  [sde_proc=<c>]
  [sxx=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [syy=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [szz=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [temperature=<n>] [<C> | <K>]
```

Arguments

<material>

Specifies the material to which the command applies. See [Specifying Materials on page 18](#).

<par1>, <par2>, ... <parn>

Parameters to be passed to user-defined Sentaurus Structure Editor procedures. The names of these parameters must exactly match the names used in the header of the user-defined procedure, although the order is insignificant.

sde_proc

Specifies the name of the Sentaurus Structure Editor procedure for this step. The procedure must have been defined in the `sde_init` file, which is assumed to be located in the current working directory.

sxx, syy, szz

Specify the intrinsic stresses in the regions generated in this step. Default value and unit: 0.0 dyn/cm^2 .

temperature

Specifies the temperature of this process step. Default value and unit: 25°C .

Description

This command defines a process step that employs user-defined Sentaurus Structure Editor procedures for structure-geometry generation, and defines the temperature and intrinsic stresses for this step.

NOTE Dimension-dependent intrinsic stresses can only be defined using the `stressdata` command.

Examples

Define a process step at temperature of 200°C, define which structure geometry is generated by the user-defined Sentaurus Structure Editor procedure named `mydepo1`, and pass values of oxide, 0.1, Copper, `depo4` to the parameters `mat`, `thickness`, `mat2`, and `region4` of the procedure `mydepo1`, respectively:

```
generic_step temperature= 200.0 sde_proc= mydepo1 mat= Oxide \  
    thickness= 0.1 mat2= Copper region4= depo4
```

See Also

[stressdata on page 634](#)

graphics

Controls the user interface of Sentaurus Visual (both the plot and field selection) from the command file.

Syntax

```
graphics
  [add.active] [add.dopants] [add.total] [cmd=<c>]
  [display.all= <list> | all]
  [display.border= <list> | all]
  [display.bulk= <list> | all]
  [display.field= <list> | all]
  [display.mesh= <list> | all]
  [field.1d.max=<n>] [field.1d.min=<n>]
  [field.max= {<c>=<n> ...}] [field.min= {<c>=<n> ...}]
  [field.scale= asinh | linear | log]
  [fields= <list>]
  [keep.aspect.ratio] [off | on]
  [svisual.cmd=<c>] [update]
  [xmax=<n>] [xmin=<n>] [ymax=<n>] [ymin=<n>]
```

Arguments

add.active, add.dopants, add.total

These options are used to select the types of fields that will be displayed in one dimension and are available to be displayed in two and three dimensions. To switch off the active dopant concentrations and NetActive, use !add.active. To switch off the dopant fields, use !add.dopants. To switch off the total concentration fields, use !add.total. Default: true.

cmd

Whenever a graphics update is typically executed, a script defined by the value of cmd is executed in addition to the graphics updating.

display.all, display.border, display.bulk, display.field, display.mesh

These arguments switch on and off respective parts of a 2D or 3D structure. They all take a list of region names or material names, or all (meaning all regions or materials) as values. For each material or region in the list, that particular part of the structure will be visible. Using an exclamation mark in front of a material or region name will hide that part of the structure. In these lists, if all appears, it is processed first for convenience.

For example, to switch on the mesh in all regions of material `Silicon` and to switch off the mesh in the region named `Oxide_1`, set:

```
display.mesh= {Silicon !Oxide_1}
```

This applies similarly to `display.border`, `display.bulk`, and `display.field`. Use `display.all` to switch on or off all bulk, field, mesh, and borders simultaneously.

For example, to switch off all regions except for the material `Silicon`, use the command:

```
graphics display.bulk= {!all Silicon}
```

`field.1d.max`, `field.1d.min`

Set the data range for all fields in xy plots. These arguments specify the maximum and minimum of the y-axis of xy plots, and apply only when Sentaurus Visual displays an xy plot of field values versus depth.

For example, `field.1d.min=1e10` sets the minimum value for an xy plot.

`field.max`, `field.min`

These arguments apply only to 2D and 3D simulations when Sentaurus Visual represents field values using color contours on the structure. These arguments adjust the field values corresponding to the maximum and minimum color contours.

For example, `field.min= {Boron=1e10 Interstitial=1e7}` sets the minimum of the color range to 10^{10} cm^{-3} when boron is displayed and to 10^7 cm^{-3} when interstitial is displayed.

`field.scale`

Changes the scaling of the contours (in 2D or 3D simulations) or the y-axis of the xy plot (1D simulation). Valid values are `asinh`, `linear`, or `log`.

`fields`

Specifies which fields to display. For 1D simulations, all fields in this list are displayed on an xy plot. For 2D and 3D simulations, the first field in the list is displayed, and the remaining fields are sent to Sentaurus Visual and can be displayed using the **Data** menu (**Scalars** tab).

In addition to this list, the options `add.active`, `add.dopants`, and `add.total` control the default availability of the fields. Since these options are true by default, the active, total, and dopant fields are sent in addition to fields specified by the `fields` argument.

`keep.aspect.ratio`

Applies to 2D simulations only. The plot range can be adjusted using `xmin`, `xmax`, `ymin`, and `ymax`. By default, the aspect ratio between the x-axis and y-axis is maintained, so that

A: Commands

graphics

the specified plot range of those arguments is not necessarily used. To remove this restriction, specify `!keep.aspect.ratio`.

`off`, `on`

By default, graphics updating is automatically switched on when in the Sentaurus Visual interface mode (command-line option `--svi`). These options can be used as follows:

- Set `off` to switch off automatic updating of graphics.
- Set `on` to switch on automatic updating of graphics using the command given by `cmd`.

`svisual.cmd`

Specifies a Tcl command file to be sent directly to Sentaurus Visual. Refer to the *Sentaurus™ Visual User Guide* for commands, parameters, and syntax.

`update`

Forces a graphics update. Usually this is not necessary, as the graphics are updated automatically when the structure or data changes.

`xmax`, `xmin`, `ymax`, `ymin`

These arguments set the range of the plot and can be used independently if required. The `xmin` and `xmax` arguments apply to 1D and 2D simulations. The `ymin` and `ymax` arguments apply only to 2D simulations.

By default, the aspect ratio between the x-axis and the y-axis is maintained, so the specified plot range is not necessarily used. Specify `!keep.aspect.ratio` to remove this restriction. Default unit: μm .

Description

The `graphics` command controls plot settings in Sentaurus Visual and selects which fields are available for visualization. Plot settings also can be controlled directly using the user interface of Sentaurus Visual. However, if for example, the same command file is to be run and rerun, it can be very convenient to add `graphics` commands to control the plot settings instead of making adjustments repeatedly in the user interface.

The `graphics` command also can be used to display or make available nonstandard fields. By default, standard fields will be sent to Sentaurus Visual. The `fields` argument is used to extend the available list (that is, to include fields not usually stored in the TDR file).

Examples

In one dimension, add `Vacancy` to the list of fields that are displayed (or, in two dimensions and three dimensions, display `Vacancy`):

```
graphics fields= {Vacancy}
```

Set the display range in two dimensions without maintaining the xy aspect ratio (the values default to the outer simulation boundaries, so only specify those boundaries to zoom in from the outer boundary):

```
graphics xmin= -0.05<um> xmax= 0.1<um> ymax= 1.0<um> !keep.aspect.ratio
```

Specify the default minimum values for various species:

```
graphics field.min= {Vacancy=1e7 VacancyGbc=1e3 VTotal=1e7}
```

Send a special command directly to Sentauros Visual (in this case, to adjust the plot view):

```
graphics svisual.cmd= "move_plot -position {0.09 -0.125}"
```

See Also

Sentauros™ Visual User Guide

grid

Sets meshing parameters, performs grid operations, and computes statistics about the mesh.

Syntax

```
grid
  [Gas]
  [get.bbox | get.bbox.cm | get.bbox.um | get.bulk.nodes | get.bulk.regions |
  get.dimension | get.elements | get.interface.nodes |
  get.interface.regions | get.max.angle | get.max.connectivity |
  get.max.edge | get.max.volume | get.max.volume.ratio |
  get.max.volume.ratio.location | get.min.angle | get.min.edge |
  get.min.edge.vertices | get.min.volume | get.min.volume.location |
  get.nodes | get.obtuse | get.total.nodes | get.total.volume | get.vertices]
  [get.brep.faces | get.brep.min.angle | get.brep.min.angle.location |
  get.brep.min.edge | get.brep.min.edge.location | get.brep.stats |
  get.brep.vertices]
  [get.interface.area] [get.mesh.stats]
  [interpolate]
  [<material>]
  [merge]
  [remesh [Adaptive] | (2D | 3D | FullD) | get.refine.check]
  [rename [print.names]]
  [save.min.edge] [save.volume]
  [set.Delaunay.type= boxmethod | conformal | constrained]
  [set.max.connectivity=<n>]
  [set.max.neighbor.ratio=<n>]
  [set.max.points=<n>]
  [set.min.angle=<n>]
  [set.min.edge=<n> [<m> | <cm> | <um> | <nm>]]
  [set.min.grid=<n> [<m> | <cm> | <um> | <nm>]]
  [set.min.normal.size=<n> [<m> | <cm> | <um> | <nm>]]
  [set.normal.growth.ratio.2d=<n>]
  [set.normal.growth.ratio.3d=<n>]
  [set.syntax.check.value=<c>]
  [smooth.brep [
    set.delpsc.accuracy=<n> [<m> | <cm> | <um> | <nm>]
    set.delpsc.resolution=<n> [<m> | <cm> | <um> | <nm>]
    set.repair.resolution=<n> [<m> | <cm> | <um> | <nm>]]]
```

Arguments: Regridding, Renaming, Refinement

2D, 3D, FullD

Extrudes grid to higher dimension. The line commands must be issued before extruding to a higher dimension. For two dimensions, at least two y-lines must have been specified.

For three dimensions, two y-lines and two z-lines must have been specified. `FullD` extrudes to the highest possible dimension.

Adaptive

If specified, `Adaptive` switches on adaptive meshing if `remesh` is specified. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

`get.refine.check`

Returns 1 if remeshing is needed, based on refinement criteria. Otherwise, it returns 0.

`interpolate`

Performs interpolation if `remesh` is specified.

`merge`

Merges adjacent regions of the same material into one region.

NOTE Do not use in combination with other arguments.

`print.names`

Prints the region names if `rename` is specified.

`remesh`

If specified, the mesh is re-created using the active mesh generator.

`rename`

Renames all regions of the structure according to the material they contain and the smallest y-coordinate point of the region, that is, from the bottom of the structure upwards. Multiple regions of the same material with the smallest y-coordinate within the given coordinate interval will increase the associated index towards the positive x-axis, from left to right (see `print.names`).

`save.min.edge`

Must be used by itself. When specified, Sentaurus Interconnect computes the smallest edge length in each direction and saves it in three fields: `MinXEdgeLength`, `MinYEdgeLength` (for 2D or 3D structures), and `MinZEdgeLength` (for 3D structures). In addition, this argument stores the element volumes in the field `ElementVolume`. When `set.min.edge` is specified, the average edge length in each direction is returned and can be used to set a Tcl variable, for example:

```
set aveEdgeLength [grid save.min.edge]
```

A: Commands

grid

`save.volume`

Sets element volumes as element values over the mesh. This field is not updated automatically.

`set.min.grid`

Specifies the minimum-allowed grid spacing. Default unit: μm .

`set.syntax.check.value`

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

Arguments: Setting Mesh Parameters

NOTE These arguments set the corresponding meshing parameters under `pdbSet Grid SnMesh`. See [Table 17 on page 188](#) for more details.

`set.Delaunay.type`

Sets the meshing PDB parameter `DelaunayType`. Specifies the type of mesh generated by Sentaurus Mesh. Available types are `boxmethod`, `conformal`, or `constrained`. Default: `constrained`.

`set.max.connectivity`

Sets the meshing PDB parameter `MaxConnectivity`. Specifies the maximum number of elements connected to a point in the final mesh. Default: `1e37`.

`set.max.neighbor.ratio`

Sets the meshing PDB parameter `MaxNeighborRatio`. Binary-tree smoothing is performed after refinements have been added to the binary tree. This prevents sudden changes in the element size that can be especially detrimental to mechanics results. The ratio of neighboring collinear edges in the binary tree can be adjusted with this argument. Default: `3.0`.

`set.max.points`

Sets the meshing PDB parameter `MaxPoints`. Specifies the maximum number of points allowed by the delaunization module of Sentaurus Mesh. Default: `500000`.

`set.min.angle`

Sets the meshing PDB parameter `MinAngle`. Specifies the minimum angle allowed in the elements of the mesh (2D only). Default value and unit: `0°`.

`set.min.edge`

Sets the meshing PDB parameter `min.edge`. Specifies the minimum edge length. Default value and unit: $2.0e-6 \mu\text{m}$.

`set.min.normal.size`

Sets the meshing PDB parameter `min.normal.size`. Specifies the smallest normal (to the interface) mesh element size on either side of an interface. Default value and unit: $8.0e-4 \mu\text{m}$.

`set.normal.growth.ratio.2d`

Sets the meshing PDB parameter `normal.growth.ratio.2d`. This factor is used to increase the size of the elements in the direction normal from the interface. The size of each layer equals the size of the previous layer multiplied by this factor (2D only). Default: 2.0.

`set.normal.growth.ratio.3d`

Sets the meshing PDB parameter `normal.growth.ratio.3d`. This factor is used to increase the size of the elements in the direction normal from the interface. The size of each layer equals the size of the previous layer multiplied by this factor (3D only). Default: 3.0.

Arguments: Grid Statistics

`Gas`

By default, quality (except volume) and `bbox` measures include the gas mesh. To exclude gas in the quality or `bbox` measure, use `!Gas`.

`<material>`

If specified, limits the measured grid statistics to the specified material.

Arguments: Reporting Grid Statistics

`get.bbox`

Prints the bounding box of the structure (coordinates in μm).

`get.bbox.cm`

Prints the bounding box of the structure (coordinates in cm).

`get.bbox.um`

Prints the bounding box of the structure (coordinates in μm).

A: Commands

grid

`get.bulk.nodes`

Returns the number of bulk nodes.

`get.bulk.regions`

Returns the number of bulk regions.

`get.dimension`

Returns the current simulation dimension.

`get.elements`

Returns the number of elements.

`get.interface.area`

Computes the area of interfaces. It can be limited to one particular interface by specifying a material interface such as "Silicon /Oxide" (μm in two dimensions, μm^2 in three dimensions).

`get.interface.nodes`

Returns the number of interface nodes.

`get.interface.regions`

Returns the number of interface regions.

`get.max.angle`

Returns the maximum angle in degrees between edges (two dimensions) or faces (three dimensions).

`get.max.connectivity`

Returns the maximum number of edges connected to a single node.

`get.max.edge`

Returns the maximum edge length in μm .

`get.max.volume`

Returns the maximum element volume (μm^2 in two dimensions, μm^3 in three dimensions).

`get.max.volume.ratio`

Returns the maximum ratio of volumes of two elements that share the same node (three dimensions only).

`get.max.volume.ratio.location`

Returns the location where the maximum volume ratio occurs (coordinates in μm).

`get.mesh.stats`

Indicates that all bulk mesh statistics must be printed. Here, all mesh statistics are listed as command arguments if they can be individually queried. They also are printed with `get.mesh.stats` or are listed below if they are available only by using `get.mesh.stats`:

- `bbox`: Bounding box (minimum and maximum extents) of the mesh.
- `bulk.nodes`: Number of nodes in mesh excluding those in the interface meshes (this gives two nodes for each vertex on an interface and one node for each bulk vertex).
- `bulk.regions`: Number of regions in the mesh.
- `dimension`: Simulation dimension.
- `elements`: Number of elements in the mesh.
- `interface.nodes`: Number of interface nodes in mesh. This will be the same as the number of interface vertices.
- `interface.regions`: Number of interface regions in the mesh.
- `max.angle`: Maximum of all angles of all elements in the mesh.
- `max.connectivity`: Maximum number of edges sharing one vertex in the mesh.
- `max.edge`: Length of maximum edge in the mesh (in μm).
- `max.volume`: Maximum element volume (in μm^D where D is the dimension).
- `max.volume.ratio`: Maximum ratio of volumes (larger volume to smaller volume) of neighboring elements.
- `max.volume.ratio.location`: Location where maximum ratio of volumes occurs.
- `min.angle`: Minimum of all angles of all elements in the mesh.
- `min.edge`: Length of minimum edge in the mesh (in μm).
- `min.edge.vertices`: Endpoints of minimum edge.
- `min.volume`: Minimum element volume (in μm^D where D is the dimension).
- `min.volume.location`: Location of the center of the element with the minimum volume.

A: Commands

grid

- `nodes`: Total number of nodes in mesh. At interfaces, there are three nodes for each vertex. In the bulk, there is one node for each vertex.
- `obtuse`: Percentage of triangles or tetrahedra that have obtuse angles.
- `total.nodes`: Same as `nodes`, that is, the total number of nodes in mesh.
- `total.volume`: Volume of mesh (in μm^D where D is the dimension).
- `vertices`: Number of vertices in mesh.

`get.min.angle`

Returns the minimum angle in degrees between edges (two dimensions) or faces (three dimensions).

`get.min.edge`

Returns the minimum edge length in μm .

`get.min.edge.vertices`

Returns the vertices of the minimum edge.

`get.min.volume`

Returns the element with the smallest area in two dimensions (in cm^2) or the smallest volume in three dimensions (in μm^3).

`get.min.volume.location`

Prints the location of the smallest element (coordinates in μm).

`get.nodes`

Same as `get.bulk.nodes`.

`get.obtuse`

Returns the percentage of triangles or tetrahedra that have obtuse angles.

`get.total.nodes`

Returns the total number of nodes in the mesh. Note the difference between points and nodes: There is a node for each region sharing an interface point in addition to each point not on an interface.

`get.total.volume`

Returns the total volume of the bounding box (cm in one dimension, cm² in two dimensions, and cm³ in three dimensions). The unit of the angles reported is degree. If none of these is chosen, all values are reported.

`get.vertices`

Returns the total number of vertices in the mesh.

Arguments: Boundary Representation (Brep) Statistics Reporting

`get.brep.faces`

Returns the number of faces in the brep.

`get.brep.min.angle`

Returns the brep minimum angle in degrees.

`get.brep.min.angle.location`

Returns the coordinates of the minimum angle (coordinates in μm).

`get.brep.min.edge`

Returns the minimum edge length (in μm) (three dimensions only).

`get.brep.min.edge.location`

Returns the coordinates of the minimum edge length (three dimensions only) (coordinates in μm).

`get.brep.stats`

Indicates that all brep statistics must be printed. Here, all brep statistics are listed as command arguments if they can be individually queried. They also are printed with `get.brep.stats` or are listed below if they are available only by using `get.brep.stats`:

- `brep.are.all.boundaries.on.bbox`: Returns true if all outer boundaries are on a flat bounding box; otherwise, false.
- `brep.bbox`: Returns the bounding box of the brep.
- `brep.conformal`: Returns true if the brep is conformal; otherwise, false.
- `brep.dimension`: Returns the dimension of the brep.
- `brep.max.angle`: Returns the brep maximum angle in degrees.

A: Commands

grid

- `brep.max.angle.from.flat`: Returns 180° -brep maximum angle (three dimensions only).
- `brep.max.angle.location`: Returns the coordinates (in μm) of the maximum angle (three dimensions only).
- `brep.min.dihedral.angle`: Returns the minimum dihedral angle in degrees in the brep (three dimensions only).
- `brep.min.dihedral.angle.location`: Returns the coordinates (in μm) of the minimum dihedral angle (three dimensions only).
- `brep.min.dihedral.angle.material`: Returns the material where the minimum dihedral angle is located (three dimensions only).
- `brep.regions`: Returns the number of regions in the brep.
- `brep.total.area`: Returns the total brep interface area in three dimensions or the bulk area in two dimensions (in μm^2).
- `brep.total.volume`: Returns the brep volume (three dimensions only) (in μm^3).

`get.brep.vertices`

Returns the number of vertices in the brep.

Arguments: Smoothing Boundary Representation (Brep)

`smooth.brep`

Removes noise and sharp features by smoothing the brep. First, a multimaterial level-set (MLS) algorithm removes any noise that may be present in the brep. Second, surface remeshing creates good-quality triangles on uneven surfaces using the Delaunay refinement for piecewise smooth complex (DelPSC) algorithm. Using multithreading with the MLS algorithm is controlled by `math numThreadsMGoals`, and using multithreading with the DelPSC algorithm is controlled by `math numThreadsSnMesh`.

`set.delpsc.accuracy`

Controls the deviation between the new surface from the DelPSC algorithm and the surface from the MLS algorithm. Default unit: μm .

The DelPSC surface can deviate from the MLS surface by, at most, the value of `set.delpsc.accuracy`. The DelPSC vertices lie exactly on the MLS surface, but DelPSC triangles cannot lie exactly on the MLS surface *unless* the MLS surface is flat. In general, the smaller the value of `set.delpsc.accuracy` is, the smoother the DelPSC surface becomes, and the more accurate the DelPSC surface represents the MLS surface. It uses the default value from:

```
pdbGet Grid Apply.Brep.DelPSC.Accuracy
```

`set.delpsc.resolution`

Controls the size of small triangles in the DelPSC algorithm. Default unit: μm . It uses the default value from:

```
pdbGet Grid Apply.Brep.DelPSC.Resolution
```

`set.repair.resolution`

Specifies the level-set cell size to remove noise and sharp features by the MLS algorithm. Default unit: μm .

It must be, at most, one-third of the thickness of the thinnest region. Otherwise, the thin region may be considered noise and may disappear. It uses the default value from:

```
pdbGet Grid Repair.Geometry.Resolution
```

The amount of geometry smoothing performed by the MLS algorithm depends on both the curvatures in the input and the level-set cell size. A noisy surface has a high curvature, so it is smoothed to a large extent to remove noise. On the other hand, a planar surface has zero curvature and is well preserved. Unfortunately, a sharp corner has a theoretically infinite curvature, so it becomes a rounded corner. The specified level-set cell size is the threshold to distinguish between the noise to be removed and the features to be preserved.

Description

This command allows you to:

- Remesh.
- Setting meshing parameters.
- Merge regions.
- Extrude.
- Rename regions.
- Measure and report on various mesh statistics.
- Measure and report on various boundary representation (referred to as *brep*) statistics.

To retrieve mesh or *brep* statistics, you can either:

- In a single pass, all statistics can be computed and returned in a Tcl array that can be accessed by the name of the measured value, for example:

```
array set returnArray [grid get.mesh.stats]
set numNodes $returnArray(get.total.nodes)
set numElements $returnArray(get.elements)
```

- To retrieve only the required parameter, use:

```
set numNodes [grid get.nodes]
```

A: Commands

grid

NOTE If you are interested in several mesh statistics, it is more efficient to retrieve all statistics in one pass and read them from a Tcl array.

To limit grid or brep statistics to a particular material or interface, specify the material on the command line. For example:

- To limit brep statistics to silicon, use:

```
grid get.brep.stats Silicon
```

- To limit brep statistics to the oxide–silicon interface, use:

```
grid get.brep.stats Silicon /Oxide
```

Several mesh statistics parameters compute a measure of element quality including:

- `avg.element.quality` (average element quality)
- `best.element.quality`
- `worst.element.quality`
- `lt3.element.quality` (percentage of elements whose quality is less than 0.3)
- `gt6.element.quality` (percentage of elements whose quality is greater than 0.6)

These arguments are computed and returned as part of `get.mesh.stats`, but they are not separately available.

NOTE For the purpose of this command, quality is defined as:

- Triangles: $4.0 * \text{sqrt}(3.0) * \text{area} (\text{sum of side lengths})^2$.
- Tetrahedra: The ratio of the radius of the inscribed sphere to the radius of the circumsphere.

Examples

Recreate the mesh using currently specified refinements:

```
grid remesh
```

Report the percentage of obtuse elements in silicon:

```
grid get.obtuse silicon
```

Compute all mesh statistics, and then read the number of nodes and the number of elements from the statistics array into `numNodes` and `numElements`, respectively:

```
array set returnArray [grid get.mesh.stats]
set numNodes $returnArray(get.total.nodes)
set numElements $returnArray(get.elements)
```


Remove noise and sharp features from the brep:

```
grid smooth.brep set.delpsc.resolution= 0.02 set.delpsc.accuracy= 0.0002 \  
set.repair.resolution= 0.003
```

See Also

[line on page 481](#)

A: Commands

help

help

Prints a list of all commands available in Sentaurus Interconnect.

Syntax

```
help
```

Description

This command can be used in interactive mode as well as in command files.

icwb

IC WorkBench (ICWB)–related functions.

Syntax

```
icwb bbox (xmin | xmax | ymin | ymax | left | right | front | back)
icwb create.all.masks
icwb dimension
icwb domain=<c>
icwb domain= [list "<domain_name1>" "<domain_name2>" ... "<domain_namen>"]
icwb filename=<c> [scale=<n>] [!recenter]
icwb gds.file=<c> cell=<c> layer.names= {<list>} layer.numbers= {<list>}
  sim2d | sim3d= {<n>} [domain.name=<c>] [stretches= {<c>= {<n>}}]
  [scale=<n>] [!recenter]
icwb layer.name=<c> list polygon.names
icwb list domains
icwb list (layerIDs | layerNames)
icwb list polygon.bounding.bboxes layer.name=<c>
icwb list polygon.inside.points layer.name=<c>
icwb list polygon.tessellations layer.name=<c>
icwb polygon.name=<c> list.segments
icwb slice.angle.offset
icwb stretch name=<c> value=<n>
```

Description

The keyword `icwb` introduces commands used to operate with ICWB TCAD layout files. The different uses of the keyword `icwb` are given here, along with their syntax and corresponding descriptions:

```
icwb bbox (xmin | xmax | ymin | ymax | left | right | front | back)
```

Returns the corresponding coordinate, which can be one of the following in either ICWB coordinates (`xmin`, `xmax`, `ymin`, `ymax`) or Sentaurus Interconnect coordinates (`left`, `right`, `front`, `back`).

A: Commands

icwb

`icwb create.all.masks`

Faster version of the `icwb.create.all.masks` command; intended to create large masks from complex layouts.

`icwb dimension`

Returns the dimension of the current domain. For the following domain types, the corresponding value for `dimension` is returned:

- Point: 1
- Gauge: 2
- Highlight: 3

`icwb domain=<c>`

`icwb domain= [list "<domain_name1>" "<domain_name2>" ... "<domain_namen>"]`

Defines the current domain. Setting the name of the current domain is a prerequisite for other ICWB commands that implicitly depend on the current domain being defined.

The second variation allows for the concatenation of multiple *gauge* domains, reorientated into one linear simulation domain.

`icwb filename=<c> [scale=<n>] [!recenter]`

Reads an ICWB TCAD layout file with the extension `.mac`. Coordinates in the ICWB file are multiplied by the optional `scale` argument as the file is read. Translation of the ICWB file coordinates to the (0,0) origin is switched off when the `!recenter` option is specified, which is supported in 3D simulations. The ICWB TCAD layout file must be read as a prerequisite to other ICWB commands that act on the domains and masks defined in that file.

`icwb gds.file=<c> cell=<c> layer.names= {<list>} layer.numbers= {<list>}
sim2d | sim3d= {<n>} [domain.name=<c>] [stretches= {<c>= {<n>}}]
[scale=<n>] [!recenter]`

Reads a GDSII layout file:

The `gds.file` argument specifies the input GDSII file name, and `cell` specifies the cell name. You can use `?` for the cell name, in which case, the first root cell will be used. The `layer.numbers` is a list of selected layers from the cell, and `layer.names` is a list of names for those layers.

The `domain.name` argument defines the name of the simulation domain. If no name is specified, `SIM3D` is used for a 3D domain, and `SIM2D` is used for a 2D domain. The domain will be set to be the current domain automatically, so you do not need to call `icwb domain=<c>` before using other ICWB commands. However, you can call `icwb domain=<c>` to set another preferred current domain.

`sim2d` | `sim3d` indicates whether it is a 2D or 3D simulation domain. The simulation domain is defined by two points, with each point defined by x- and y-coordinates in the GDSII coordinate system. For two dimensions, the two points are endpoints of a segment. The segment must be horizontal or vertical in the GDSII coordinate system. For three dimensions, the two points are the two opposite corners of the simulation domain. The first point is the left-back corner and the second point is the right-front corner in the UCS.

The `stretches` argument is a list of stretches, with each stretch having a name and being defined by a segment with two points. For a 3D domain, the segment must cross the bounding box of the domain. For a 2D domain, the segment must intersect with the 2D domain.

The `scale` argument is the same as that in the `icwb filename=<c> [scale=<n>]` command. Coordinates in the GDSII file are multiplied by `scale` as the file is read in to Sentaurus Interconnect.

Translation of the ICWB file coordinates to the (0,0) origin is switched off when the `!recenter` option is specified, which is supported in 3D simulations.

This command can be called multiple times to set multiple simulation domains. The GDSII layout file or the ICWB TCAD layout file must be read as a prerequisite to other ICWB commands that act on the domains and masks defined in that file.

```
icwb layer.name=<c> list polygon.names
```

Lists polygon names given a layer name.

```
icwb list domains
```

Queries the names of the current domains.

```
icwb list (layerIDs | layerNames)
```

Returns a list of layer IDs or layer names of the ICWB TCAD layout file.

```
icwb list polygon.bounding.bboxes layer.name=<c>
```

Returns the bounding box rectangle for each polygon in the layer.

```
icwb list polygon.inside.points layer.name=<c>
```

Returns a point from inside each polygon in the layer.

```
icwb list polygon.tessellations layer.name=<c>
```

Breaks each polygon in the layer into a set of rectangles, and then returns these rectangles.

```
icwb polygon.name=<c> list.segments
```

Lists polygon segments given a polygon name.

A: Commands

icwb

```
icwb slice.angle.offset
```

Returns the relative angle of the active simulation domain.

```
icwb stretch name=<c> value=<n>
```

Applies the given stretch by the given amount to the current domains. The order of applied stretches is important since the location of other stretches can change given the application of one stretch.

icwb.composite

Loads a GDSII file for a 2D TCAD simulation based on composite simulation domains.

Syntax

```
icwb.composite
  cell=<c>
  composite.domain.name= {<list>}
  gds.file=<c>
  layer.names= {<list>}
  layer.numbers= {<list>}
  sim2d.list= {{<n>} {<n>}}
  [scale=<n>]
  [stretches= {<c>= {<n>}}]
```

Arguments

`cell`

Specifies the cell name.

`composite.domain.name`

Defines the name of the composite simulation domain. The name must consist of one alphanumeric name for each 2D simulation domain. The names are combined with a plus sign. No whitespace is allowed.

`gds.file`

Specifies the name of the input GDSII file.

`layer.names`

List of the names for the layers specified using `layer.numbers`.

`layer.numbers`

List of the selected layers from the GDSII file specified by `gds.file`.

`scale`

Coordinates in the GDSII file are multiplied by the optional `scale` argument as the file is read.

A: Commands

icwb.composite

sim2d.list

Defines a sequence of 2D simulation domains that are joined to form a composite 2D simulation domain.

stretches

List of stretches, with each stretch having a name and being defined by a segment with two points. For a 2D domain, the segment must intersect with the 2D domain.

Description

This command is similar to the `icwb gds.file=<c>` command, but it applies to 2D composite simulation domains.

Examples

The following command loads the `BCD.gds` GDSII file and activates a 2D composite simulation domain consisting of two regular simulation domains called `BJT1` and `BJT2`:

```
set LAYERS "1:0 2:0 3:0 4:0 5:0 6:0 7:0 9:0"

set NAMES "NWELL NACTIVE PACTIVE NPLUS POLY PBASE EMIT CONT"

icwb.composite \
  gds.file= "BCD.gds" cell= "Inverter" scale= 1e-3 \
  layer.numbers= $LAYERS layer.names= $NAMES \
  composite.domain.name= "BJT1+BJT2" \
  sim2d.list= {{6200.0 8150.0 8400.0 8150.0} {8250.0 8800.0 8250.0 9900.0}}
```

icwb.contact.mask

Creates contacts for subsequent device simulations that are tied to a layer in the ICWB TCAD layout file.

Syntax

```
icwb.contact.mask
  (label.name=<c> [distance=<n>] | layer.name=<c>)
  (box <material> [adjacent.material=<c>] [boxheight=<n>] [xhi=<n>] [xlo=<n>]
  |
  point <material> [replace] [x=<n>])
  [name=<c>] [<other_arguments>]
```

Arguments

box, <material>, adjacent.material, boxheight, xhi, xlo

When you specify the `box` option, the part of the interface between the <material> and the `adjacent.material` that intersect the box given by the lateral extent of the layout polygon and the vertical extent given by `xhi` and `xlo` is turned into a contact.

If `xlo` is not specified, the vertical location of the interface is determined automatically, and the vertical extent of the box is controlled by `boxheight`. The default for `boxheight` is 0.1 μm .

label.name, distance

Name of a text label in the ICWB TCAD layout file.

Only the text label within the given distance of the active 2D simulation domain is recognized. The default for `distance` is 0.03 μm . Simulation domain units are used (not layout units). The `distance` argument has no effect for 3D simulation domains.

layer.name

Name of a layer in the ICWB TCAD layout file.

name

Name of the contact. If `name` is not specified, the default value is either the label name or the layer name, whichever is specified.

point, <material>, replace, x

A point-type contact contains all the boundary elements of one or more regions. The regions are identified by the specified material and the spatial coordinate of a set of one or more points.

A: Commands

icwb.contact.mask

If `layer.name` is specified, the lateral coordinates of these points are taken as a point inside each lump of the layer (if there is more than one lump).

If `label.name` is specified, the lateral coordinates are taken as the anchor points of all text labels with the given label name.

The height coordinate can be either specified explicitly with the `x` argument or computed automatically. If the `replace` option is specified, the original material of the region of the contact is replaced by gas in the TDR file.

<other_arguments>

Any other arguments supported by the `contact` command.

Description

This command serves as an interface between the ICWB TCAD layout and the `contact` command by automatically obtaining the lateral placement of the contact from the specified ICWB layer or text label location, taking the vertical placement from either the `x` argument (point type) or the `xhi` and `xlo` arguments (box type) (or, if not specified, computing it internally), and passing all other options directly to the `contact` command.

The `icwb.contact.mask` command supports both box-type and point-type contacts:

- A *box-type* contact consists of elements at the surface of one region or material inside the box. The lateral extent of the box is determined automatically from the layer segment (two dimensions) or the layer polygons (three dimensions), while the vertical extent is taken from the `xlo` and `xhi` arguments of the `contact` command. If the `xlo` argument is not explicitly given, the vertical extent is determined automatically, based on the topmost interface with the material mentioned in the command call. The vertical extent of the box is controlled by `boxheight`. If a text label name is given, the layer associated with that text label is used.
- A *point-type* contact contains all the boundary elements of one region. The lateral position of the point is determined automatically as a point inside the layer segment (two dimensions) or the polygon (three dimensions), while the vertical position is taken from the `x` argument of the `contact` command. If the `x` argument is not explicitly given, the vertical position is determined automatically, based on the interfaces with the material given in the command call. If a text label name is given, the anchor point of the text label is used.

For details on how to define contacts, see [contact on page 381](#).

NOTE It may be advantageous to create auxiliary layers or text labels in IC WorkBench Edit/View Plus for the placement of contacts.

NOTE For 3D simulations, sometimes the placement of contacts in Sentaurus Interconnect causes meshing problems. Consider using the `!cut.mesh` option of the `contact` command (see [contact on page 381](#)).

Examples

The following commands create contacts for subsequent device simulations tied to the layers named emitter and pdrain, and the text label drain, in the example ICWB TCAD layout file:

```
icwb.contact.mask layer.name= emitter box polysilicon \  
    adjacent.material= oxide xlo= -2.05 xhi= -1.95  
  
icwb.contact.mask layer.name= pdrain name= drain point aluminum replace \  
    x= -2.0  
  
icwb.contact.mask label.name= "drain" point Aluminum replace
```

A: Commands

icwb.create.all.masks

icwb.create.all.masks

Creates positive and negative mask versions for all layers found in the currently active ICWB simulation domain.

Syntax

```
icwb.create.all.masks
[save.to.tdr= 0 | 1]
[shift= {<dy> <dz>}]
[stretchypos= {<y0> <dy>}] [stretchyneg= {<y0> <dy>}]
[stretchzpos= {<z0> <dz>}] [stretchzneg= {<z0> <dz>}]
[reflecty= <y0>] [reflectz= <z0>]
```

Arguments

See [icwb.create.mask on page 455](#).

Description

The names of the masks are given by the layer names and the suffix `_p` for the positive and `_n` for the negative version of the mask.

For example, if the TCAD layout contains a layer with the name `TRENCH`, the corresponding mask names are `TRENCH_p` and `TRENCH_n`.

This command makes two calls to the `icwb.create.mask` command for each layer it finds in the layout file. Therefore, this command supports the `reflect*`, `save.to.tdr`, `shift`, and `stretch*` arguments of the `icwb.create.mask` command.

NOTE Use the command-line option `-n` to suppress automatic syntax-checking in Sentaurus Interconnect when using this feature. The syntax-checker cannot determine which masks are available. Therefore, it may incorrectly flag the use of an unknown mask.

icwb.create.mask

Creates a mask for subsequent use in `etch`, `deposit`, or `photo` commands from one or more ICWB layers.

Syntax

```
icwb.create.mask
  layer.name= <c> | <list>
  [name=<c>] [polarity= positive | negative]
  [save.to.tdr= 0 | 1]
  [shift= {<dy> <dz>}]
  [stretchypos= {<y0> <dy>}] [stretchyneg= {<y0> <dy>}]
  [stretchzpos= {<z0> <dz>}] [stretchzneg= {<z0> <dz>}]
  [reflecty= <y0>] [reflectz= <z0>]
```

Arguments

`layer.name`

Name of one or more layers in the ICWB TCAD layout file. If more than one layer name is given, the resulting mask is the union of the polygons (3D) or segments (2D) from all the layers listed. (Use the `icwb list layerNames` command to obtain a list of all layer names.)

`name`

Name of the mask. The mask name defaults to the layer name. If the layer name list contains more than one entry, the first layer name is used.

`polarity`

Sets the polarity. Select `negative` to invert a mask. The polarity is assumed to be `positive` by default, that is, points inside the mask are considered masked.

`reflecty`

Reflects the layer around the y-axis defined by the given coordinate before creating the mask.

`reflectz`

Reflects the layer around the z-axis defined by the given coordinate before creating the mask.

A: Commands

icwb.create.mask

save.to.tdr

If `save.to.tdr=0`, the created masks are not saved in a TDR file. If `save.to.tdr=1`, the mask is saved in a TDR file if – *and only if* – the total number points in the mask is less than or equal to the limit set in the PDB by the `GDS Max.Points` parameter. The default value of `save.to.tdr` is 200. If needed, you can increase this limit with:

```
pdbSet GDS Max.Points <n>
```

In addition, you can suppress the saving of *any* mask created from GDSII layers by setting:

```
pdbSetBoolean GDS Save.To.TDR 0
```

shift

Shifts the layers by the specified amount before creating the mask.

stretchyneg, stretchypos, stretchzneg, stretchzpos

Stretches the layer before creating the mask. The last four characters of the arguments determine whether the stretch is applied along the y- or z-direction and whether the layer is stretched to the positive or negative side of the stretch position.

NOTE More than one `shift`, `stretch*`, and `reflect*` argument can be used in the `icwb.create.mask` command. As these operations may not commute, you must note the order in which these operations are applied if more than one is used. First, `shift` is applied, and then `stretchypos`, `stretchyneg`, `stretchzpos`, `stretchzneg`, `reflecty`, and finally `reflectz` are applied.

Description

This command serves as an interface between the ICWB TCAD layout and the `mask` and `polygon` commands, and provides a convenient way to generate 1D, 2D, and 3D masks consisting of the points, segments, or polygons from one or more ICWB layers based on a dimension-independent syntax. The command automatically determines the dimension of the currently active ICWB simulation domain.

Examples

The following commands create masks from the corresponding layers named by `layer.name`:

```
icwb.create.mask layer.name= NWELL polarity= negative
```

```
icwb.create.mask layer.name= NWELL name= NOTNWELL
```

```
icwb.create.mask layer.name= "NPDIFF PPDIFF NPLUG PBASE" name=STI info=1
```

```
icwb.create.mask layer.name= 1:0 stretchyneg= {1.2 -0.25}
```

See Also

[deposit on page 401](#)
[etch on page 414](#)
[mask on page 493](#)
[photo on page 544](#)
[polygon on page 559](#)

init

Sets up the mesh and background doping levels.

Syntax

```

init
  [Adaptive]
  [concentration=<n>] [<m-3>|<cm-3>|<um-3>|<nm-3>]
  [DelayFullD] [done]
  [field=<c>]
  [fields.values= <list>]
  [load.commands] [<material>]
  [mixed.mesh]
  [multiple.cell.param= {
    interface0= { {rangeY0 rangeZ0 rangeR0} [{rangedy1 rangedz1 rangeR1}
      {rangedy2 rangedz2 rangeR2}] ...}
    [interface1= { {rangeY0 rangeZ0 rangeR0} {rangedy1 rangedz1 rangeR1}
      {rangedy2 rangedz2 rangeR2}] ...}]
    [interface2= { {rangeY0 rangeZ0 rangeR0} {rangedy1 rangedz1 rangeR1}
      {rangedy2 rangedz2 rangeR2}] ...}]
    number.pillars= {Ny Nz}
    variation.type= Gaussian | Linear | Uniform
  }]
  [notch.direction= <list>]
  [pdb] [pdb.only]
  [sat=<c>] [sigmac=<c>]
  [slice.angle=<n>] [<degree>]
  [tdr=<c>] [top]
  [unit.cell=pillar]
  [unit.cell.param= {
    setup= {
      {<substrate_mat> <surround_mat>}
      {Xmin Xmax Ymin Ymax Zmin Zmax}
      {yz.spacing=<n>}
      [{Xmin s0 x1 s1 ... Xmax s}]
    }
    thicknesses= {h0 h1 h2 ...}
    pillar.materials= {mat1 mat2 mat3 ...}
    interface0= { {Y0 Z0 R0} [{dy1 dz1 R1} {dy2 dz2 R2}] ...}
    [interface1= { {Y0 Z0 R0} {dy1 dz1 R1} {dy2 dz2 R2}] ...}]
    [interface2= { {Y0 Z0 R0} {dy1 dz1 R1} {dy2 dz2 R2}] ...}]
  }]
  [wafer.orient= <list>]

```


Arguments

Adaptive

When loading a TDR file containing geometry but no mesh, a mesh is generated automatically. This argument determines whether adaptive meshing is used. The default is the return value of `pdbGet Grid Adaptive`.

concentration

Concentration of the incorporated data field. The only available unit is cm^{-3} , but any nodal quantity (with any internal unit) can be initialized with this argument if no unit is specified. Default value and unit: 0.0cm^{-3} .

DelayFullD

By default, Sentaurus Interconnect generates a minimum-dimensional structure, which will be extruded to higher dimensions when Sentaurus Interconnect encounters a *mask*. To generate a full-dimensional structure, use `!DelayFullD`.

done

Returns 1 if the initialization is performed; otherwise, returns 0.

field

Name of the data field to be initialized everywhere in the structure.

fields.values

Specifies a list of parameters where the parameter name is the name of the field to be introduced in the bulk regions of the new structure, and the value is the initial value, for example, `fields.values={Boron=1e18}`. A list of fields of any name can be initialized with this argument and, for solution variables or stress components, units are accepted.

load.commands

Loads the commands in the TDR format file. Default: true.

<material>

Specifies a material for doping. It must be used with the `field` argument.

mixed.mesh

Sets the simulation structure to mixed-mesh mode, that is, it allows the presence of rectangles, bricks, prisms, and pyramids without converting them to triangles or tetrahedra.

A: Commands

init

NOTE This mode is only available for mechanics.

`multiple.cell.param`

Specifies fields to define a structure having multiple pillar cells. Each pillar cell is defined as in `unit.cell.param`, with process variation specified by this parameter (see [Multiple-Pillar Structure With Hexahedral Mesh on page 249](#)).

The `number.pillars` field specifies the number of pillar cells along the y- and z-directions, respectively. By default, both numbers are 1.

The `variation.type` field specifies the variation type for all parameters that are allowed to be varied. These include the radii of all circles, the centers of inner circles, and the center shifts for all outer circles. Default: `Uniform`.

The `interface0`, `interface1`, and `interface2` fields define the variation range for each parameter on the interface. If all the variation ranges are zero, all pillar cells are identical. For a pillar structure, there are at least two interfaces. Identical to `unit.cell.param`, when the number of interfaces is less than the number of thicknesses, this command will assume the next interface has the same variation ranges as the previous one. For example, if only `interface0` is defined, it assumes `interface1` (and `interface2` if there are three interfaces) has the same variation ranges. When no interface is defined, all pillar cells will be identical.

NOTE The variation of each parameter must not violate any limitations that are set for one-pillar cells. Circles inside each cell must be well separated. The simulation will exit with an error message if any overlap exists between circles.

`notch.direction`

NOTE This was originally the `flat.orient` argument.

Specifies a direction from the wafer center to the notch using a list of three Miller indices, or specifies an orientation that is normal to the wafer flat (imagine that a wafer flat is a plane parallel to `wafer.orient`). Default: `{1 1 0}`.

`pdb`

Loads `pdb` parameters along with geometry and data in the TDR file. Default: `true`.

`pdb.only`

Loads only `pdb` parameters without geometry and data in the TDR file. Default: `false`.

`sat`

Specifies to read the structure file in the Sentaurus Structure Editor format.

`sigmac`

Specifies to read the structure file in Sigma-C 3D format.

`slice.angle`

Angle of the simulation domain with respect to the wafer coordinate system. Default value and unit: -90° .

The `slice.angle` can be specified using a `CutLine2D` command:

```
init slice.angle= [CutLine2D 1.65 0.15 1.95 0.6]
```

`tdr`

Specifies the TDR format file to read.

In general, a TDR file can contain a variety of information depending on which tool was used to generate the file. By default, Sentaurus Interconnect writes files with sufficient information to restart a simulation. This includes current parameter settings, stored commands (such as `polygon`, `mask`, `contact`), bulk mesh and data, and, in three dimensions, a boundary (see [Saving a Structure for Restarting Simulations on page 39](#)). If such a file is specified, all this data is read and used to restart the simulation. It is also possible to read TDR files that include only bulk mesh and data, or only a boundary. If only a boundary is available, Sentaurus Interconnect creates a mesh using current refinement criteria. However, TDR files generated by Sentaurus Device do not contain information to restart a simulation and may not be read correctly.

For information about the TDR format, refer to the *Sentaurus™ Data Explorer User Guide*.

`top`

Specifies that the gas is found at the smallest x-value (at the top of the structure). If `!top` is specified, the gas is added at the highest x-value (at the bottom). Default: `true`.

`unit.cell`

When `unit.cell=pillar`, this argument specifies that a pillar structure with a hexahedral mesh will be generated (see [Pillar Structure With Hexahedral Mesh on page 246](#)).

`unit.cell.param`

Specifies a list of arguments that define the geometry of a pillar structure (see [Pillar Structure With Hexahedral Mesh on page 246](#)):

- `setup` defines:
The substrate material and the surrounding material:
{<substrate_mat> <surround_mat>}

A: Commands

init

The structure bounding box: {Xmin Xmax Ymin Ymax Zmin Zmax}.

The grid spacing in the yz plane: {yz.spacing=<n>}.

The grid spacing along the x-direction: {Xmin s0 x1 s1 x2 s2 ... Xmax s}.

The specification for the grid spacing along the x-direction is optional. If it is used, Xmin, Xmax and all the interface locations must be included. Xmin, x1, x2 ... Xmax are locations, and s0, s1, s2 ... s are spacings. Values must be paired, and each location is followed by a spacing. It works like the `line` command.

If the grid spacing along the x-direction is not defined, `yz.spacing` is used to determine the grid size along the x-direction. In the yz plane, `yz.spacing` determines the grid size around the center of the inside circle on the top interface as well as the parts that are meshed independently, for example, the four corners of the top interface. Grid sizes in some parts depend on the meshing of other parts, for example, the grid size of eccentric circles depends on the grid size of the inner circle, and the grid size of other interfaces depends on the meshing of the top interface. The default value of `yz.spacing` is 0.02 μm .

- `thicknesses` defines the thickness of each layer, starting with the thickness of the substrate. At least two thicknesses are required and, at most, three thicknesses can be specified.
- `pillar.materials` defines the material for each pillar region, starting with the innermost pillar. Up to five concentric or eccentric pillars are allowed.
- `interface0` defines the first interface between the pillars and the substrate. The values {Y0 Z0 R} are the center coordinates and radius that define the inside circle. The values {dy1 dz1 R1}, {dy2 dz2 R2} are optional and define the concentric or eccentric circles around the inside circle. They are the center shifts from the inside circle and its radius. For concentric circles, set `dy1=dz1=dy2=dz2= ... = 0`. You can define up to five circles. At least one interface is required.
- `interface1` and `interface2` are optional. The number of interfaces must be as same as the number of thicknesses, or one less than the number of thicknesses. When the number of interfaces is one less than the number of thicknesses, the command assumes that the next interface is the same as the previous one. For example, if only `interface0` is defined, `interface1` is assumed to be the same as `interface0`. The circles on interfaces define the tops or bottoms of pillars.

`wafer.orient`

Wafer orientation specified as three Miller indices. Default: `wafer.orient= {0 0 1}`.

Description

Sets up the mesh from either a rectangular specification or a file. The command also allows initialization of the background doping concentration and type.

Examples

Read in a structure previously saved using `struct tdr=temp`, which produces the file `tmp_sis.tdr`:

```
init tdr= temp
```

Create a structure with the current mesh parameters and a uniform boron doping of 1×10^{15} :

```
init field= Boron concentration= 1e15
```

Initialize a structure and set the Germanium and Silicon fields so that the mole fraction is 0.2, that is, $\text{Si}_{0.8}\text{Ge}_{0.2}$:

```
init fields.values= [MoleFractionFields SiliconGermanium 0.2]
```

Initialize a structure with a (110) wafer and an *miscut towards* the $\langle 1\bar{1}0 \rangle$ direction:

```
init wafer.orient= {1 1 0} notch.direction= {1 -1 0} miscut.tilt= 8 \  
miscut.toward= {1 -1 0}
```

See Also

[CutLine2D on page 395](#)

[line on page 481](#)

[region on page 601](#)

[struct on page 643](#)

insert

Inserts segments into 1D structures, polygons into 2D structures, and polyhedra into 3D structures.

Syntax

```
insert
  polygon=<c> | polyhedron=<c> | segments= {<n1> <n2> ...}
  [Adaptive]
  [fields.values= <list>]
  [new.material=<c>] [new.region=<c>]
  [repair]
  [replace.materials= {<mat1> ... <matn>}]
  [replace.regions= {<reg1> ... <regn>}]
```

Arguments

Adaptive

If specified, `Adaptive` switches on adaptive meshing if remeshing. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

fields.values

Specifies a list of parameters where the parameter name is the name of the field to be introduced in the inserted geometry and the value is the initial value, for example, `fields.values= {Boron=1e18}`. A list of fields of any name can be initialized with this argument and, for solution variables or stress components, units are accepted.

new.material

Sets the material for the inserted segment, polyhedron, or polygon. This argument is mandatory for all but TDR polyhedra or polygons.

new.region

Sets the name of the inserted region. It works with one segment, polygon, or polyhedron only, that is, it does not work if the segment, polygon, or polyhedron contains more than one region.

NOTE This argument is not allowed for TDR polyhedra.

polygon

Specifies the polygon to insert. Only polygons created with the `xy` argument of the `polygon` command are allowed. The `polygon` argument fails if the simulation is three dimensional, or if it is one dimensional and cannot be extruded to two dimensions because there are no y-lines available.

polyhedron

Specifies the polyhedron to insert. It needs a 3D simulation or a simulation that can be extruded to three dimensions.

repair

If specified, this option enables an algorithm that tries to repair sharp features or geometric artifacts produced by polyhedron insertion. Repair operations can be disabled either globally by setting `mgoals !repair.3d` or locally for each `insert` command by specifying `!repair`. Default: true.

replace.materials

Specifies a list that indicates the materials to be replaced by the polyhedron. In addition to explicit materials, the argument `bulk.materials` is allowed. If `bulk.materials` is used, it means that all materials in the structure, except gas, will be replaced.

replace.regions

Specifies a list of regions to be replaced by the polyhedron.

segments

Segments are defined as a list of an even number of coordinates (in μm). If more than two coordinates are specified, unique region names are generated for each segment or region.

Description

Segments are defined using the `segments` argument, but polygons and polyhedra must be defined using the `polygon` and `polyhedron` commands, respectively. You must specify one of the `segments`, `polyhedron`, or `polygon` arguments.

When specified, `replace.materials` and `replace.regions` provide a list of materials and regions to be replaced. If neither is specified, all materials will be replaced. If both are specified, the union of them will be replaced. The `new.material` argument changes the polyhedron material temporarily.

The `insert` command can be used to perform polyhedron or polygon etching and deposition as well as the more general polyhedron or polygon insert functionality. Polyhedron or polygon etching is performed by specifying `new.material=gas` or by creating a gas polyhedron. You

A: Commands

insert

also can do the same in one dimension with `segments`, but this is the same as `CMP` or `fill`. Polyhedron or polygon deposition is performed by specifying `replace.materials=gas` as well as having one or more bulk materials in the polyhedron or polygon, or defining them temporarily with `new.material`.

The `insert` command operates only in the 3D MGOALS mode for polyhedra. If the SDE mode is switched on, calling this command will set `sde off`.

NOTE It is recommended that no faces of the polyhedron overlap material interfaces or the outermost simulation boundary. This may cause failures due to round-off errors or may create poor mesh elements such as slivers.

Examples

Etch the structure using a polyhedron called `prism`:

```
insert polyhedron= prism new.material= Gas
```

Replace all materials in the structure with a polyhedron called `smallCube`, and fill the polyhedron with oxide:

```
insert polyhedron= smallCube new.material= Oxide
```

Replace the nitride, and only the nitride, in the simulation with oxide inside the polyhedron called `smallCube`:

```
insert polyhedron= smallCube replace.materials= {Nitride} new.material= Oxide
```

Replace all materials in the structure with a polygon called `Channel`, and fill the polygon with material `Silicon2`, and the region is named `ChannelRegion`. This polygon can be inserted without merging with neighboring silicon regions:

```
mater add name= Silicon2 new.like= Silicon alt.matename= Silicon  
insert polygon= Channel new.material= Silicon2 new.region= ChannelRegion
```

See Also

[Inserting Polyhedra on page 274](#)

[polygon on page 559](#)

[polyhedron on page 563](#)

integrate

Returns the volume integration of the named quantity.

Syntax

```
integrate
  [absolute] [average] [element] [interfaces]
  [<material>]
  [max= {<n> <n> <n>}]
  [min= {<n> <n> <n>}]
  [mode= mesh | boundary | jagged]
  [name=<c>] [region=<c>]
  [skipgas] [syntax.check.value=<c>]
```

Arguments

`absolute`

Specifies that integration is performed with the absolute values of the named quantity.

`average`

Specifies that the average value of the named quantity is computed and added to the returned Tcl list.

`element`

Specifies that integration is performed elementwise. In this case, it is expected that the quantity specified by `name` is an elemental quantity.

`interfaces`

Specifies that integration is performed on interface meshes.

`<material>`

Used to limit integration to regions of the specified material. See [Specifying Materials on page 18](#).

`max`

List of numbers defining the x-, y-, and z-coordinates of the lower-right front corner of the cutting box. For 1D, 2D, and 3D structures, a list of one, two, or three numbers is required, respectively. The possible maximum number is used for missing numbers.

A: Commands

integrate

min

List of numbers defining the x-, y-, and z-coordinates of the upper-left back corner of the cutting box. For 1D, 2D, and 3D structures, a list of one, two, or three numbers is required, respectively. The possible minimum number is used for missing numbers.

mode

Specifies the integration mode within a given box:

- The mesh mode (default) uses a mesh-cutting algorithm.
- The boundary mode for three dimensions cuts the boundary to a cuboid and remeshes the cuboid using the given mesh refinements. The boundary mode is for 3D simulations only.
- The jagged mode includes all nodes contained entirely within the given box for integration.

name

Quantity to be integrated. Default: `Z_Plot_Var`.

region

Limits integration to only the region specified.

skipgas

Specifies that integration is omitted on invisible meshes.

syntax.check.value

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

Description

This command integrates the field specified with the `name` argument (by default, the last unnamed `select` command field) over the entire structure or within a given box specified by `min` and `max`. If `material` is specified, the integration is limited to regions of the given material. If a `region` is specified, the integration is limited to only the named region.

The command by default expects the quantity to be nodal and the integration is performed nodewise, but if `element` is given, an elemental quantity is expected and the integration proceeds elementwise.

A Tcl list is returned where the first value is the integrated value.

The second value is the volume of the computed regions:

- In `<value_unit>*cm` for one dimension
- In `<value_unit>*cm2` for two dimensions
- In `<value_unit>*cm3` for three dimensions

where `<value_unit>` is the unit of the named quantity.

The third value is the dose (the integrated value divided by the simulated area in cm^{-2} in all dimensions). The fourth and fifth values are the minimum and maximum of the named quantity, respectively. If `average` is specified, the averaged result for the named quantity is appended to the returned Tcl list.

Examples

Return the integral boron in all silicon regions in the structure, the volume of all silicon regions, and the combined boron dose in all silicon regions:

```
integrate silicon name= Boron
```

Return the integrated pressure elementwise in the entire structure (excluding gas) and the volume of the structure (excluding gas):

```
select z= "1.0/3*(StressEL_xx+StressEL_yy+StressEL_zz) "  
integrate element
```

Return a list of integral, volume, dose, minimum, maximum, and average values of boron within the box defined by the upper-left corner (0.0, 0.0) and the lower-right corner (10.0, 0.2):

```
integrate name= Boron average min= {0. 0.} max= {10. 0.2}
```

Return the integrated term `BActive`, the volume, and the combined dose in all silicon regions in the structure. The term `BActive` is first converted to a temporary data field before integration:

```
select z= BActive  
integrate silicon
```

Return a list of integral, volume, dose, minimum, and maximum of boron within the cuboid defined by the upper-left back corner (0.0, 0.0, 0.0) and the lower-right front corner (0.4, 0.4, 0.4):

```
integrate name= Boron mode= jagged min= {0. 0. 0.} max= {0.4 0.4 0.4}
```

interface

Returns the location or the value of the selected data field at a material interface.

Syntax

```
interface
  [All] [data] [<material>] [name=<c>]
  [p1= {<n> <n> <n>} p2= {<n> <n> <n>}]
  [precision=<n>]
  [region=<c>]
  [side=<c>]
  [syntax.check.value=<c>]
  [thickness]
  [x=<n>] [<m>|<cm>|<um>|<nm>]
  [y=<n>] [<m>|<cm>|<um>|<nm>]
  [z=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

All

If specified, all interface locations are returned. Otherwise, only the first value is returned.

data

If specified, the value of the selected data field at the interface is returned.

<material>

Usually works with an interface description and returns the location or value of the selected quantity at the interface. See [Specifying Materials on page 18](#). When the `thickness` option is specified, a material name can be used.

NOTE If the `thickness` option is not specified *and* an interface is not specified, an error occurs. If the specified interface does not exist in the current structure, an error is reported.

name

Name of the data field to be returned when `data` is specified. Default: `z_plot_var`.

p1, p2

Specify the two endpoints of a cutline. Both `p1` and `p2` must be specified together as a list of numeric values. Only the first `<dim>` numbers from each list is read, where `<dim>` is the spatial dimension of the simulation. Specifying the endpoints with `p1` and `p2` allows for

nonaxis-aligned cuts. Endpoints also can be used to limit axis-aligned cuts instead of cutting through the entire structure.

precision

Controls the number of precision digits of floating values (in scientific notation).
Default: 6.

region

When the `thickness` option is specified, you can specify the name of a region from which to extract its thickness.

side

Takes its value from one of the two bulk materials consisting of the interface or the 'interface' (literally) itself. If `side` is not specified, the 'interface' itself is assumed. If `side` is specified as one of the bulk materials, the value of the selected data field for the bulk material is returned. This argument is effective only if `data` is specified.

`syntax.check.value`

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

thickness

Specify this option to extract the thickness of a region. The thickness of the first region along the cutline will be returned. The unit of the returned thickness is micrometer. When you specify this option, you can specify a material or region, but not an interface. If a material is specified, the thickness of the first material region will be returned. If a region is specified, the thickness of the region will be returned.

`x`, `y`, `z`

Provide the description of a line to look for the interface. These arguments are unnecessary in 1D simulations. In 2D simulations, one of `x` and `y` must be specified. In 3D simulations, two of these must be specified. Default value and unit: 0.0 μm .

Description

This command returns the position of an interface or returns the value of the selected data field if `data` is specified. Therefore, the command can be used to prepare plots of material thickness, silicon consumption, or material growth. It also is used to provide an argument to the `interpolate` command, which returns a list if there is more than one interface. The list-processing commands of Tcl, particularly `lindex`, are very helpful.

A: Commands

interface

Examples

Return the silicon–oxide interface at a lateral position of 1.0 μm :

```
interface y= 1.0 silicon /oxide
```

Return the top position of the oxide:

```
interface oxide /gas
```

Return any oxide–nitride interfaces between (1.0, 1.0) and (1.1, 1.1). This specification is valid for one or two dimensions, but not three dimensions. In one dimension, it returns the interfaces between 1.0 and 1.1:

```
interface p1= {1.0 1.0} p2= {1.1 1.1} Nitride /Oxide
```

See Also

[interpolate on page 473](#)

[plot.xy on page 553](#)

[point.xy on page 557](#)

interpolate

Returns the requested position or value at a specified location.

Syntax

```
interpolate
  <material> [name=<c>]
  [syntax.check.value=<c>]
  [value=<n>]
  [x=<n>] [<m> | <cm> | <um> | <nm>]
  [y=<n>] [<m> | <cm> | <um> | <nm>]
  [z=<n>] [<m> | <cm> | <um> | <nm>]
```

Arguments

<material>

Limits the search to a single material. See [Specifying Materials on page 18](#).

name

Name of a data field. This allows printing without using the `select` or `tclsel` commands. Default: `z_plot_var`.

syntax.check.value

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

value, x, y, z

The combination of these arguments determines how the command operates:

- In 1D simulations, you must supply either `x` or `value`. If `x` is supplied, Sentaurus Interconnect returns the value at `x`. If `value` is supplied, Sentaurus Interconnect returns the locations at which the selected profile crosses `value`.
- In 2D simulations, two of the four arguments must be given (not `z`).
- In 3D simulations, three of the four arguments must be given.

For example, in two dimensions, if `x` and `value` are given, the locations along `x` where `value` is crossed are returned. If `x` and `y` are given, the value at the location `(x,y)` is returned. The default unit of `x`, `y`, and `z` is μm .

A: Commands

interpolate

Description

This powerful command analyzes simulation results. It returns the interpolated value of one coordinate given the other two coordinates in three dimensions defined by x- and y-variables, and the data field. It also works for 1D simulations by returning values as a function of one coordinate.

This command returns a Tcl list of values if more than one is found. For example, there may be several junctions found along a given line. All of these are returned and can be processed by standard Tcl list operations. In most cases, this command returns a single value. The returned value is in internal units. Internal units are CGS; for example, the unit for stress is dyn/cm^2 .

In addition, this command can return the data value at a specified position in the structure or return the position at which a specified data value occurs.

Examples

Return the value of the data field at the position (1.0 μm , 1.0 μm) in the oxide:

```
interpolate oxide x= 1.0 y= 1.0
```

Return a list of zero crossings in silicon of the data field along the vertical line $y = 0.0 \mu\text{m}$:

```
interpolate y= 0.0 silicon value= 0.0
```

Return a list of zero crossings in silicon along a horizontal line at a depth of 2.0 μm :

```
interpolate silicon x= 2.0 value= 0.0
```

Return the value of the data field at 1.0 μm :

```
interpolate x= 1.0 silicon
```

Return the yy component of the element stress field at 1.0 μm :

```
interpolate x= 1.0 oxide name= StressEL_yy
```

Return a list of zero crossings in silicon material:

```
interpolate silicon value= 0.0
```

See Also

[interface on page 470](#)

[plot.xy on page 553](#)

[point.xy on page 557](#)

j_integral

Calculates the J-integral of a crack and returns its value.

Syntax

```
j_integral
  crack=<c>
  x=<n> [<m> | <cm> | <um> | <nm>]
  y=<n> [<m> | <cm> | <um> | <nm>]
  z=<n> [<m> | <cm> | <um> | <nm>]
  [CtIntegral | JvIntegral] [number=<n>]
```

Arguments

crack

Name of the crack. The crack must have been defined and solved.

CtIntegral, JvIntegral

Specifies the type of contour integral to compute, either:

- CtIntegral is used to compute the C(t)-integral for viscoplastic materials.
- JvIntegral is used to compute the J_v-integral for viscoelastic materials.

These contour integral types must be used only for cracks in viscoplastic or viscoelastic materials.

number

Specifies the number of contours around the crack tip where the J-integral is evaluated. By default, the J-integral is evaluated over the first contour. When two or more contours are specified, the J-integral is evaluated for every contour starting from the first contour.

x, y, z

Specify the x-, y-, and z-coordinates of the point where the J-integral is evaluated. The point must be located at the crack tip. If the point is not located at the crack tip, a point that is on the crack tip and is closest to the requested point is used to perform the calculation. Default unit: μm .

A: Commands

j_integral

Description

The return value can be used in a Tcl expression. The command must be used after a simulation, or after loading a TDR file that contains crack simulation results. For 3D cracks, an average value of the J-integral over the crack front is returned in addition to the value at the specified point.

Examples

Calculate the J-integral of crack `crk01` at point (4.0, 5.0, 0.0) over the first three contours, and assign the results to a Tcl variable `the_j`:

```
set the_j [j_integral crack= crk01 x= 4.0 y= 5.0 z= 0.0 number= 3]
```

KG2E

Computes Young's modulus from the bulk modulus and the shear modulus.

Syntax

```
KG2E <n> <n>
```

Arguments

<n>

The first value is the bulk modulus.

The second value is the shear modulus.

Description

The same units are assumed for all moduli.

Examples

Compute Young's modulus from the bulk modulus $1.2272e12$ dyn/cm² and the shear modulus $6.328e11$ dyn/cm²:

```
KG2E 1.2272e12 6.328e11
```

KG2nu

Computes the Poisson ratio from the bulk modulus and the shear modulus.

Syntax

```
KG2nu <n> <n>
```

Arguments

<n>

The first value is the bulk modulus.

The second value is the shear modulus.

Description

The same units are assumed for all moduli.

Examples

Compute the Poisson ratio from the bulk modulus $1.2272e12$ dyn/cm² and the shear modulus $6.328e11$ dyn/cm²:

```
KG2nu 1.2272e12 6.328e11
```

layers

Prints material interfaces and integrated data field values.

Syntax

```
layers
  [<material>] [merge] [name]
  [precision=<n>] [print.logfile]
  [region.names] [syntax.check.value=<c>]
  [x=<n>] [<m>|<cm>|<um>|<nm>]
  [y=<n>] [<m>|<cm>|<um>|<nm>]
  [z=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

<material>

Used to limit the reporting of layers to regions of the specified material. See [Specifying Materials on page 18](#).

merge

Specifies that the adjacent regions with the same material should be merged. Default: false.

name

Name of a data field. This allows printing without using the `select` or `tclsel` commands. Default: `Z_Plot_Var`.

precision

Controls the number of precision digits of floating values (in scientific notation). Default: 12.

print.logfile

Allows output to be written to the log file.

region.names

Specifies that region names must be printed in addition to the material names for each region in the structure.

A: Commands

layers

`syntax.check.value`

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

`x, y, z`

Specify the constant values of a line along which sectioning will be performed. In one dimension, these arguments are not necessary. In two dimensions, only one of `x` or `y` can be specified for a given device. Specifying `x` produces a horizontal slice through the device, and `y` specifies a vertical slice. An easy way to remember this is that the cross section is taken at the constant value specified. For a 3D simulation, two of these three arguments must be specified. Default unit: μm .

Description

This command prints the material interfaces and integrates the selected data field in each region. It is most useful for examining doping because it gives the integrated doping in each layer. This command can be simulated with the `integrate` and `interface` commands, and it returns a Tcl list of each material.

Examples

In a 1D simulation, list all material interfaces:

```
layers
```

In a 2D simulation, list all material interfaces at a lateral position of $0.0\ \mu\text{m}$ and integrate the data field named `Boron`:

```
layers y= 0.0 name= Boron
```

See Also

[integrate on page 467](#)

[interface on page 470](#)

[select on page 609](#)

[System on page 651](#)

line

Specifies the position and spacing of mesh lines.

Syntax

```
line
  location=<n> [<m>|<cm>|<um>|<nm>] (x | y | z)
  [clear] [mgoals]
  [spacing=<n>] [<m>|<cm>|<um>|<nm>]
  [spacing.method= even | regular | smooth]
  [tag=<c>] [virtual.spacing]
```

Arguments

clear

Clears lines in preparation for a new structure definition, or removes all ticks stored for the UseLines method (see [UseLines: Keeping User-Defined Mesh Lines on page 201](#)).

location

Location along the chosen axis. Default unit: μm .

mgoals

Lines for Sentauros Mesh (continuum) meshes are stored separately. By default, line commands are applied to continuum meshes. Use negative values for these arguments to *not* apply mesh lines. For example, for a line command to apply only to a continuum mesh, use !mgoals.

spacing

Local grid spacing. Each mesh line has a characteristic required spacing. Lines are graded from one spacing to the next over the interval. The default is a spacing equal to the largest interval between the neighboring lines. Default unit: μm .

spacing.method

Specifies the method to use when distributing the ticks created by the spacing argument:

- **even**: Distributes the ticks evenly, trying to approximate the spacing specified at the beginning of the interval.
- **regular**: Distributes the ticks evenly using the exact spacing specified at the beginning of the interval, and leaving the last interval with an approximate size if there is no more room to accommodate the requested size.

A: Commands

line

- `smooth`: Distributes the ticks to have a smooth grading of spacing between lines.

Default: `smooth`.

tag

Lines can be labeled for later reference by `region` commands. The label can be any word.

virtual.spacing

If specified, this option enables the creation of virtual ticks between lines. The mesh refinement will snap to these virtual ticks whenever possible. When Sentaurus Mesh attempts to produce a refinement that is finer than the spacing provided by the virtual ticks, it will proceed to split the spacing in half using the default bisection algorithm.

x, y, z

Orientation of the mesh line. Specifying `x` places a mesh line at a constant `x`-value. A series of `line x` commands would specify the horizontal grid locations during the simulation.

Description

The `line` command is used with the `init` and `region` commands to create the initial mesh. Only rectilinear structures can be specified with the `line` and `region` commands, that is, rectangular regions in two dimensions and cuboid-shaped regions in three dimensions.

Sentaurus Interconnect uses the unified coordinate system (UCS):

- `x` is the direction normal to the wafer with positive-`x` oriented into the bulk of wafer.
- `y` is perpendicular to the `x`-direction and lies along the wafer surface; `y` is in the lateral direction.
- The `z`-direction is used for three dimensions, and the direction is given by $X \times Y$.

By default, Sentaurus Interconnect delays promoting a structure until it is necessary (by use of a higher dimensional mask). The lines specifying the higher dimensions are stored until they are needed. During the `init` command, the `line` and `spacing` information is expanded into mesh ‘ticks’ that are stored in the PDB. These ticks are used every time a mesh is created if `UseLines` is switched on (see [UseLines: Keeping User-Defined Mesh Lines on page 201](#)).

When you specify the `spacing` argument, the `line` command generates a series of lines or ticks that is incorporated into the mesh refinement as lines that completely cut through the device. This is useful to generate the `spacing` for the initial structures generated by the `init` and the `region` commands. After the `init` command, however, if new lines are specified and `UseLines` is switched on, the `spacing` argument is ignored, and only one tick or mesh line can be added at a time.

However, if the `virtual.spacing` option is specified, the `spacing` argument will be used to build a series of *soft* lines or ticks. All user-defined refinement coordinates will snap to those lines, thereby creating a more predictable spacing pattern compared to the one generated by the default bisection algorithm.

To create an entirely new structure, the command `line clear` must first be issued to remove old lines and mesh ticks before issuing new `line`, `region`, and `init` commands.

Examples

There are three user-specified y-lines and two user-specified x-lines. Taking the y-lines as an example, there is a finer spacing in the center than at the edges. After processing, Sentaurus Interconnect produces a mesh with x-lines at 0.0, 0.42, 0.69, 0.88, 1.0, 1.12, 1.31, 1.58, and 2.0. Around the center, the spacing is 0.12, approximately what will be requested. At the edge, the spacing is 0.42 because that is as coarse as it could become without having an interval ratio greater than 1.5 (a fixed quantity). If the interval ratio was allowed to be 9, for example, there would be one interval of 0.9 and one interval of 0.1 on each side. In this example, specifying a spacing of 1 at the edges is redundant because that is what the spacing of the user-specified lines was already:

```
line x location= 0 spacing= 0.02 tag= surf
line x location= 3 spacing= 0.5 tag= back
line y location= 0 spacing= 1 tag= left
line y location= 1 spacing= 0.1
line y location= 2 spacing= 1 tag= right
```

In the following example, a structure with two bricks is created using the `line` and `region` commands. After the structure is created, a set of virtual lines is enabled to guide the refinement. If the `virtual.spacing` option is not used, the default bisection algorithm would place the refinement at powers of 2 between the given line locations:

```
mgoals accuracy=0.002

line x loc=0.0 spacing=0.1 tag=top
line x loc=0.2 spacing=0.1 tag=siflat
line x loc=1.0 spacing=0.1 tag=bottom
line y loc=0.0 spacing=0.1 tag=left
line y loc=0.1 spacing=0.1 tag=mid
line y loc=0.2 spacing=0.1 tag=right

region Silicon xlo=siflat xhi=bottom
region Silicon xlo=top xhi=siflat ylo=left yhi=mid

init !DelayFullD

line clear
line virtual.spacing
line x loc=0.0 spacing=0.01
line x loc=0.2 spacing=0.01
```

A: Commands

line

```
line x loc=1.0 spacing=0.01
line y loc=0.0 spacing=0.01
line y loc=0.1 spacing=0.01
line y loc=0.2 spacing=0.01

refinebox clear
refinebox xrefine = 0.1 yrefine = 0.1
refinebox interface.materials = Silicon min.normal.size = 0.0075 \
    normal.growth.ratio = 1.5

deposit oxide iso thickness =0.1
struct tdr=virtualspacing
```

See Also

[init on page 458](#)

[region on page 601](#)

line_edge_roughness

Adds line edge roughness (LER) to named masks.

Syntax

```
line_edge_roughness
  correlation.length=<n> [<m> | <cm> | <um> | <nm>]
  masks= <list>
  max.segment.length=<n> [<m> | <cm> | <um> | <nm>]
  normal= "Y" | "Z"
  standard.deviation=<n> [<m> | <cm> | <um> | <nm>]
  [max.tries=<n>] [min.radius=<n>]
  [random.reseed] [random.seed=<n>] [smooth.points=<n>]
```

Arguments

`correlation.length`

Specifies the correlation length of the randomized LER. Corresponds approximately to the concept of *wavelength*. Default value and unit: 20 μm .

`masks`

Specifies as a list the names of masks to receive LER defined by the other arguments in the `line_edge_roughness` command.

`max.segment.length`

Specifies the maximum segment length. Mask edges are subdivided into segments that are approximately this size or smaller before LER deviation is added to each. Default value and unit: 1 μm .

`max.tries`

Specifies the maximum number of LER mask generation attempts. For nonzero values, detection of nearly collinear points is performed after LER generation, and the LER process is restarted if decimation occurs based on the `mgoals accuracy` value. The default value is 0, meaning that the LER mask is accepted as it is, with no decimation detection.

`min.radius`

When `normal` is not specified, where two mask edges receiving LER meet, the corner is first rounded before LER is applied. This allows a well-defined application of LER and avoids discontinuous jumps in the resulting mask shape. The rounding radius is the greater of `min.radius` and twice the `correlation.length`. Default unit: μm .

A: Commands

line_edge_roughness

normal

Defines the normal axis. Only mask segments normal to this axis receive LER. This axis is also the reference axis along which the LER deviation is added to the given mask segment. The default is to add LER to all edges of the mask.

random.reseed

Before the calculation of LER, the random number generator is reseeded to ensure each call of `line_edge_roughness` results in randomized noise that is uncorrelated with other calls of `line_edge_roughness`.

To switch off random reseeding, use `!random.reseed` to reproduce the same LER from call to call. Default: true.

random.seed

Used to reproduce specific LER calculations from one run to the next by setting the same random seed in both runs. Ignored when `!random.reseed` is used.

smooth.points

Alters the LER function at the structure boundaries, allowing for better mirror boundary conditions when uniting symmetry-reduced structure parts at those boundaries. Defines the number of segments in the discrete LER function, counting from the boundary edge of the mask, which must be adjusted or smoothed. Default: 0 (implying no smoothing).

standard.deviation

Specifies the standard deviation of the randomized LER. Corresponds approximately to the concept of added noise *amplitude*. Default value and unit: 2 μm .

Description

This command adds LER to the named masks, along the mask edges normal to the given normal axis ("Y" or "Z"). LER can be added to a mask only once. See [Boolean Masks on page 254](#).

Examples

Add LER to the mask named `mask1` along mask segments normal to the z-axis. These segments are subdivided into smaller segments of length smaller than or equal to 5 nm. LER is characterized by a `correlation.length` of 25 nm and `standard.deviation` of 5 nm. The random number generator is reseeded automatically before LER is calculated:

```
line_edge_roughness normal= "Z" masks= {mask1} \  
  correlation.length= 25.00<nm> standard.deviation= 5.00<nm> \  
  max.segment.length= 5.00<nm>
```

See Also

[Line Edge Roughness Effect on page 256](#)

load

Interpolates data from a TDR file onto the current mesh.

Syntax

```
load
  tdr=<c> (merge | rename | replace | sum)
  [fast.tdr.ave= <list>]
  [flip (left | right | front | back | up | down)]
  [keep.outside] [material.specific] [offset= {<n> <n>}]
  [save.transform=<c>]
  [shift=<n>]
  [species= <list> actions= <list>]
  [transform= {<n> <n> <n> <n> <n> <n>}]
```

Arguments

`fast.tdr.ave`

Averages the data from a list of TDR files (all with identical meshes to the current mesh) and replaces the current data with the averaged data from the files.

NOTE Do not use `fast.tdr.ave` with any other argument. To use it, specify a list of files, for example:

```
fast.tdr.ave= {mydata1.tdr mydata2.tdr mydata3.tdr}
```

`flip (left | right | front | back | up | down)`

Performs a flip of the data in the indicated direction about the outer boundary before interpolation. This argument must be used with a direction, for example, `flip front`.

`keep.outside`

Specifies that the fields outside the bounding box of the loaded TDR file will remain unchanged. If this option is switched off (false) and the `replace` option is specified, the fields outside the TDR bounding box will be deleted (assuming the TDR bounding box is smaller than the current device bounding box). Default: false.

A: Commands

load

`material.specific`

By default, interpolation is material specific, so that data is interpolated from the file if the material in the existing structure matches. Use `!material.specific` to allow interpolation regardless of material.

`merge`

Adds only new datasets that do not currently exist in the structure.

`offset`

For 2D structures only. Offsets the data by a vector before loading it.

`rename`

Adds new datasets and renames them by adding the suffix `__load`.

`replace`

Adds new datasets and replaces existing datasets with new datasets of the same name.

`save.transform`

Saves a TDR file for the transformed structure. You can examine the saved TDR file to ensure the loaded structure is in the required location.

`shift`

Shifts the data laterally before loading it.

`species, actions`

These lists specify field-by-field actions. The field name must be one of those in the loaded TDR file. The actions in the `actions` list can be `merge`, `rename`, `replace`, and `sum`.

`sum`

Adds new datasets and sums matching datasets.

`tdr`

Specifies the name of the command file in the TDR format. Sentaurus Interconnect checks for standard file names with the `.tdr` extension. For 2D structures, a 3D TDR command file is also allowed. The data on the 2D overlap cross-section between the 3D structure and the 2D structure will be interpolated to the current 2D mesh. When calculating the intersection, the 2D structure is assumed to be located at $z=0$.

transform

Provides a general interface for translating or rotating the structure to be loaded before interpolation. When loading 3D results to a 2D structure, since the 2D structure is assumed to be located at $z=0$, to obtain 3D data other than on the cross section at $z=0$ of the 3D structure, this argument can be used to transform the 3D structure to a required location.

In one dimension, one value must be specified: the shift in the x-coordinate.

In two dimensions, six values must be specified: `rxx`, `ryx`, `rxz`, `ryy`, `offsetx`, `offsety`.

In three dimensions, 12 values must be specified: `rxx`, `ryx`, `rxz`, `rxz`, `ryy`, `rzy`, `rxz`, `ryz`, `rzz`, `offsetx`, `offsety`, and `offsetz`. First, the offset is applied, and then the rotation matrix is applied. Sentaurus Interconnect uses a right-hand coordinate system, so the rotation must follow the right-hand rule and the rotation matrix should be specified correctly. For example, in three dimensions, for a 90° rotation about the x-axis, the rotation matrix should have `rxx=1`, `rzy=-1`, `ryz=1`, and the remaining values should be zero.

Description

This command interpolates data from TDR files onto the current mesh. There are several options for handling the new and old datasets. First, the actions can be applied individually to selected datasets using `species` and `actions`. If the `species` list appears, the `actions` list must be specified and must have the same number of members as the `species` list. If this is the case, only the species in the `species` list are taken from the external datasets. If the `species` list does not appear, one of the global actions is used. The default behavior is a global sum where new datasets are added and, if there is an existing dataset with the same name, the external data is added (summed) with the existing dataset.

The other actions that can be performed are:

- `merge` takes only the new datasets that do not currently exist in the structure.
- `rename` renames new datasets by appending `__load` to the name, which can be manipulated with the `select` command as required.
- `replace` replaces current datasets with new datasets of the same name.

Examples

Replace all existing datasets with those in the file `in_sis.tdr`:

```
load tdr= in replace
```

Load the TDR file `in_sis.tdr`, sum `Potential_User` and the existing `Potential_User` (if available), and replace the existing `Temperature` data field by the one in the `in_sis.tdr` file:

```
load tdr= in species= {Potential_User Temperature} actions= {sum replace}
```

A: Commands

load

See Also

[select on page 609](#)

LogFile

Prints a message to the screen and to the log file.

Syntax

```
LogFile  
  <C>  
  [IL0 | IL1 | IL2 | IL3] [PrintCommands]
```

Arguments

<C>

Specifies the message to be printed to the screen and to the log file.

IL0, IL1, IL2, IL3

Specifies the information level.

PrintCommands

Specifies whether commands are printed to both the screen and the log file. To stop the printing of commands, use !PrintCommands. Default: true.

Description

This command prints messages to the terminal window in which Sentaurus Interconnect is running and to the log file. If IL0, IL1, IL2, or IL3 is given and this command is called from within a Sentaurus Interconnect command, the message is printed only if the information level is equal to or greater than the one specified.

Examples

Print the string "Step 25" to the screen and the log file:

```
LogFile "Step 25"
```

Print the contents of the Tcl variable DebugInfo only if info=1 or higher has been specified in the calling command:

```
LogFile IL1 "$DebugInfo"
```

A: Commands

LogFile

Do not print the `sel z=1e10 name=Test` command and subsequent commands to the screen and the log file:

```
LogFile !PrintCommands  
sel z=1e10 name=Test
```

mask

Creates a mask for subsequent use in `etch`, `deposit`, or `photo` commands.

Syntax

```
mask
  [bbox] [bool=<c>] [clear] [covered.status]
  [cut.x=<n> (materials= <list> | regions= <list>)]
  [get.segments] [get.segments.z]
  [list] [name=<c>] [negative]
  (
    [polygons= <list>] [segments= <list>] |

    [left=<n>] [<m>|<cm>|<um>|<nm>]
    [right=<n>] [<m>|<cm>|<um>|<nm>]
    [front=<n>] [<m>|<cm>|<um>|<nm>]
    [back=<n>] [<m>|<cm>|<um>|<nm>]

  )
```

Arguments

`bbox`

Returns the mask bounding box. The command returns a list of lists where the values are in centimeters: {ymin zmin} {ymax zmax}.

`bool`

Performs Boolean operations between masks. It cannot be used with `polygons` and `negative`. See [Boolean Masks on page 254](#).

`clear`

Clears the list of all masks. If `name` is specified, it clears only that mask.

`covered.status`

Used to obtain information about the coverage of the simulation domain. The following strings may be returned:

- `covered`: The mask completely covers the simulation domain.
- `uncovered`: The mask does not cover the simulation domain at all.

A: Commands

mask

- `partial.2d`: The mask partially covers the domain, but in a way that the mask does not promote the simulation dimension (that is, the mask does not vary in the z-direction over the simulation domain).
- `partial`: The mask partially covers the simulation domain, and its use in etching or deposition forces the simulation to three dimensions.

`cut.x`, `materials`, `regions`

The `cut.x` argument is only available in two dimensions. It must be used with either `regions` or `materials` to create a mask. The mask is created by taking a cut through the set of regions created by a union of regions listed in `regions` and regions of one of the materials listed in `materials`. The cut is taken at the x-coordinate specified by `cut.x`, and the resulting outline is used to create the mask. Default unit of `cut.x`: μm .

`get.segments`, `get.segments.z`

Retrieves segments that result from cutting the mask at $z=\text{get.segments.z}$. The default value of `get.segments.z` is the midpoint of the simulation domain in the z-direction.

`left`, `right`, `front`, `back`

Specify the corners of one rectangle. The rectangle is added to the current list for the mask. If several rectangles must be specified for a mask, several `mask` commands must be used with the same name. Default unit: μm .

`list`

Prints a list of all currently defined masks. If `name` is specified, it prints information for that mask only.

`name`

Name of a mask. If used with `clear` or `list`, only the specified mask will be reported or removed. If defining a new mask, `name` must be specified.

`negative`

Inverts the type of mask. By default, points inside the mask are considered masked. For example, the command `mask name=xyz negative` inverts an existing mask `xyz`.

For an empty negative mask, all points are considered masked, and the simulation domain is fully covered by the mask.

`polygons`

Specifies a mask as a list of named polygons. The named polygons must have been defined using `polygon` commands.

segments

Specifies a list of coordinates of mask segments. Several mask segments can be specified at the same time. The first coordinate defines the beginning of a segment, the second coordinate defines the end of the segment, the third defines the beginning of the second segment, and so on. In 3D simulations, mask segments are extended across the entire structure in the z-direction. Default unit: μm .

Description

This command manages and creates masks for use with subsequent `etch`, `deposit`, or `photo` commands. Mask definitions are stored in TDR files and re-stored when loading a TDR file in the `init` command. Masks can be defined by rectangles, polygons, and segments, or they can be read using the IC WorkBench Edit/View Plus–TCAD Sentaurus interface (see [Chapter 13 on page 285](#)).

Masks are created additively. If more than one `mask` command is issued with the same name, the union of the specified masks is assumed. To change a mask, first clear it and then assign a new specification (in two separate calls to the `mask` command).

Examples

Define a mask named `field`:

```
mask name= field left= 0.0 right= 10.0
```

The position of this mask is the same as in the previous example:

```
mask name= mgoals segments= {0.0 10.0}
```

Return information about all masks in array format:

```
mask list
```

Return a list of mask names:

```
array set maskinfo [mask list]
LogFile "Mask names: [array names maskInfo]"
```

Print all information about the mask named `PolygonMask`:

```
array set polyInfo $maskInfo(PolygonMask)
LogFile "Contents of PolygonMask: [array names polyInfo]"
```

Print the number of polygons in the mask named `PolygonMask`:

```
LogFile "Number of polygons in PolygonMask: [llength $polyInfo(polygons)]"
```

A: Commands

mask

Print the first polygon in the mask named PolygonMask:

```
LogFile "polygon 0 in PolygonMask: [lindex $polyInfo(polygons) 0]"
```

See Also

[deposit on page 401](#)
[etch on page 414](#)
[photo on page 544](#)
[point on page 555](#)
[polygon on page 559](#)

mater

Returns a list of all materials in the current structure, and adds new materials to a global list.

Syntax

```
mater
  [add] [alt.matername=<c>]
  [bbox | bbox.cm | bbox.um] [cropped.bbox]
  [cubic | hexagonal | orthorhombic]
  [horizontal.orient= {<n> <n> <n>}]
  [Interface]
  [lattice.const=<n>] [lattice.const.b=<n>] [lattice.const.c=<n>]
  [like.interpolate] [list.all]
  [max= {<n> <n> <n>}] [min= {<n> <n> <n>}]
  [name=<c>] [new.like=<c>]
  [polytype=<c>]
  [syntax.check.value=<c>]
  [vertical.orient= {<n> <n> <n>}]
```

Arguments

`add`

Adds a new material specified by name to the global material list.

`alt.matername`

Specifies the name of the material that should be written in the TDR file for regions of the newly defined material.

For example, when writing the TDR file, if the newly defined material is called `MySilicon` and `alt.matername=Silicon`, Sentaurus Interconnect uses `Silicon` for those regions of `MySilicon`.

This is useful when transferring the structure to device simulation.

`bbox`, `bbox.cm`, `bbox.um`

If specified, the `mater` command returns the maximum extents of the material in two points in centimeter or micrometer.

`cropped.bbox`

Returns the cropped bounding box of a material within a user-defined bounding box.

A: Commands

mater

cubic, hexagonal, orthorhombic

Specifies the lattice system as either cubic, hexagonal, or orthorhombic for a crystalline material as specified by name.

horizontal.orient

Specifies the flat orientation for crystalline material as three Miller indices. The default value of `horizontal.orient` depends on the wafer orientation:

- If `vertical.orient = {0 0 1}`, `horizontal.orient = {1 1 0}`.
- If `vertical.orient = {1 1 0}` or `{1 1 1}`, `horizontal.orient = {1 -1 0}`.

For all other nonstandard wafer orientations, a flat orientation that is orthogonal to the wafer orientation must be specified.

NOTE The arguments `horizontal.orient` and `vertical.orient` supersede `notch.direction` and `wafer.orient` specified in the `init` command for this particular material (as specified by name).

Interface

Returns a list of interface materials in the current structure.

lattice.const

Specifies the lattice constant for the crystalline material. Default value and unit: $5.431 \times 10^{-4} \mu\text{m}$.

lattice.const.b

Specifies the lattice constant *b* for the crystalline material. If `lattice.const.b` is not specified, it is assumed to be the same as `lattice.const`. Default unit: μm .

lattice.const.c

Specifies the lattice constant *c* for the crystalline material. If `lattice.const.c` is not specified, it is assumed to be the same as `lattice.const`. Default unit: μm .

like.interpolate

Usually, the interpolation code interpolates data from and to materials that are *like* each other. Use `!like.interpolate` to prevent such interpolation. See [Like Materials: Material Parameter Inheritance on page 23](#).

list.all

Lists all the materials defined.

`max, min`

Specify the bounding box of a material. These arguments compute the cropped bounding box of a material.

`name`

Name of the material.

`new.like`

Name of the existing material from which all default values are inherited. For newly created materials, `pdb` parameters for this material are checked first and, if not found, the 'Like' material parameters are used (see [Like Materials: Material Parameter Inheritance on page 23](#)).

`polytype`

Specifies the polytype for the crystalline material as specified by name. For cubic or orthorhombic lattice systems, the valid polytypes are `Sc`, `Bcc`, `Fcc`, and `Zincblende`. For a hexagonal lattice system, the valid polytypes are `2H`, `3C`, `4H`, and `6H`.

`syntax.check.value`

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

`vertical.orient`

Specifies the wafer orientation for crystalline material as three Miller indices. Default value: `vertical.orient= {0 0 1}`.

NOTE The arguments `vertical.orient` and `horizontal.orient` supersede `wafer.orient` and `notch.direction` specified in the `init` command for this particular material (as specified by name).

Description

This command returns a list of all materials in the current structure. The format of the list is compatible with the material specification for the program. Bulk material names are returned if no arguments are given. Interface materials can be obtained by using `Interface`.

This command also computes the cropped bounding box of a material that lies within a user-specified bounding box defined by `max` and `min`. The `name` argument is given as input.

A: Commands

mater

Examples

Add germanium material to the global material list and make it inherit the default values from silicon material:

```
mater add name= Germanium new.like= Silicon
```

Specify the crystal properties for hexagonal GaN:

```
mater name= GaN hexagonal polytype= 2H \  
  lattice.const= 3.189e-4 lattice.const.c= 5.185e-4 \  
  vertical.orient= {0 0 1} horizontal.orient= {1 1 0}
```

math

Sets the numeric and matrix parameters. Parameters set with the `math` command are stored in TDR files by default.

Syntax

```

math
  [AMS.NegErrCntrl] [FTS.NegErrCntrl] [LocTrnErrCntrl] [NegErrCntrl]
  [auto | compatible]
  [compute | Flow]
  [dimension= 1 | 2 | 3]
  [extrapolate]
  [pardiso | ils] [scale]

  [fullNewton | modNewton]
  [newtonDeriv] [newtonRate1=<n>] [newtonRate2=<n>]
  [newtonStats=<i>] [newtonSteps1=<i>] [newtonTries1=<i>]

  [coord.dfise]
  [coord.transform coord.translate (coord.read | coord.write)]
  [coord.ucs] [coord.xyz] [coord.yxz] [coord.-zyx]

  [maxNumberOfDomains=<i>]
  [NumberOfElementsPerDomain=<i>]
  [numThreads=<i>]
  [numThreadsAssembly=<i>]
  [numThreadsBoxMethod=<i>]
  [numThreadsILS=<i>]
  [numThreadsInterp=<i>]
  [numThreadsMGoals=<i>]
  [numThreadsPardiso=<i>]
  [numThreadsSnMesh=<i>]
  [parallel.license= go.serial | go.abort | go.keep | go.recheck | go.wait]
  [threadStackSize=<i>]

  [milne | difference] [tr_bdf | euler]
  [use.interpolated.geom.coeff] [voronoitriangle]

```

Arguments: Solver Selection

`AMS.NegErrCntrl`

Allows stricter error control for each solve time step after an adaptive meshing step by calculating the error from negative updates instead of damped results.

A: Commands

math

auto, compatible

These options modify the conditional numerical reproducibility (CNR) for Intel CPUs. Floating-point computations on an Intel processor might not give bit-for-bit identical results for equivalent computations, even though the same hardware and binary are used. The fused multiply-add (FMA) instruction is a common source of differences. A small difference can be amplified in applications where the boundary moves (for example, oxidation). Therefore, differences can be minimized by specifying the `compatible` option. Default: `compatible`.

NOTE The CNR value can be set only at the beginning of the simulation before any arithmetic operation is performed. The `compatible` mode can significantly degrade the performance of the simulation if the PARDISO solver is used.

compute, Flow

Specifies the type of equation to which the command specification applies. If omitted, it applies to all equation types.

dimension

Specifies the dimensionality to which the command specification applies. If omitted, it applies to all dimensions.

extrapolate

If this option is specified, extrapolated solutions can be used as the initial guess for the backward differentiation formula (BDF) step. Extrapolated solutions are calculated using solutions at the previous time step and at the first trapezoidal rule (TR1) step.

FTS.NegErrCntrl

Allows stricter error control for the first solve time step by calculating the error from negative updates instead of damped results.

LocTrnErrCntrl

Allows stricter error control for each solve time step by modifying the handling of negative updates:

- 1 ($|upd|/org \leq abs$)
- 0 ($|upd|/org + \leq abs$)

LocTrnErrCntrl can be switched on for individual solution variables in specific materials using:

```
pdbSetBoolean <mater> <solution> LocTrnErrCntrl 1
```

NegErrCntrl

Allows stricter error control at each Newton iteration step by calculating the error from negative updates instead of damped results. NegErrCntrl can be switched on for individual solution variables in specific materials using:

```
pdbSetBoolean <mater> <solution> NegErrCntrl 1
```

pardiso, ils

Specifies the type of linear solver to apply to the system:

- `pardiso` selects the parallel direct solver PARDISO, which is based on the LU factorization with pivoting of the matrix. PARDISO decomposes the matrix.
- `ils` selects the iterative linear solver ILS, including preconditioners, iterative methods, scaling, and convergence criteria. (You can change the default settings of ILS parameters by specifying `pdbSet Math` commands.) To set ILS parameters in the parameter database, see [Setting Parameters of the Iterative Solver ILS on page 330](#).

scale

Applies row/column scaling to the matrix in an attempt to make it better conditioned. This is a recommended argument. No scaling is performed if the modified Newton scheme (`modNewton`) is used.

Arguments: Newton Method

fullNewton, modNewton

Specifies the type of nonlinear equation solver to use:

- `fullNewton` performs a matrix factorization at each step.
- `modNewton` tries to reuse one matrix factorization for several solve steps.

A: Commands

math

The full Newton method may be more robust, but it can use more solution time than the modified Newton method:

- `modNewton` is the default in two dimensions for both PARDISO and ILS.
- `modNewton` is the default for ILS in three dimensions.

`newtonDeriv`

Allows the Jacobian computation to be switched on during the modified Newton step. Default: `false`.

`newtonRate1`

For the modified Newton method, if the solution for any of the Newton steps 1 through `newtonSteps1` is `newtonRate1` or more times better than the previous step, the next step can be a solve-only step. Otherwise, the next step will perform a matrix factorization. Default: 4.0.

`newtonRate2`

For the modified Newton method, if the solution for any of the Newton steps `newtonSteps1+1` onwards is `newtonRate2` or more times better than the previous step, the next step can be a solve-only step. Otherwise, the next step will perform a matrix factorization. Default: 32.0.

`newtonStats`

Prints information on Newton iterations.

`newtonSteps1`

For the modified Newton method, any of the Newton steps 1 through `newtonSteps1` must improve the solution by the factor `newtonRate1` over the previous step. Otherwise, the next step will be a full Newton step. For Newton steps `newtonSteps1+1` onwards, the solution at each step must improve by the factor `newtonRate2`. Otherwise, the next step will be a full Newton step. Default: 12.

`newtonTries1`

Number of first modified Newton step breakdowns allowed before switching to the full Newton method. Default: 2.

Arguments: Time Discretization

`milne, difference`

Controls whether the next time step is estimated using the Milne's device or the divided difference method. Default: `milne`.

`tr_bdf, euler`

Specifies the type of time discretization scheme to use. The options are TR-BDF(2) or the backward Euler method. Default: `tr_bdf`.

Arguments: Parallel Processing

Sentaurus Interconnect provides parallel processing for the matrix assembly, the box method, and the linear solvers by generating multiple threads to accelerate simulations on multicore shared-memory computers. By default, only one processor (thread) is used.

NOTE The number of threads must not exceed the number of actual CPUs (cores) of the computer.

NOTE Observe the following general recommendations to obtain the best results from a parallel run: Speedup is only obtained for sufficiently large problems. In general, the mesh should have at least 10000 nodes. Three-dimensional problems are good candidates for parallelization.

NOTE You must run a parallel job on an unloaded computer. As soon as multiple jobs compete for processors, performance decreases significantly (a parallel job could run even longer than a serial one).

NOTE The parallel execution of the matrix assembly on the solvers PARDISO and ILS produces different rounding errors. Therefore, the number of Newton iterations in particular may change.

NOTE Parallel performance scalability of the different modules (such as assembly and linear solver) can vary dramatically.

NOTE You do not need to set the OpenMP environment variable `OMP_NUM_THREADS`. You need only specify the number of threads required in the `math` command.

To use more than one thread, specify the following arguments of the `math` command in the command file:

`maxNumberOfDomains`

Modifies the maximum number of domains each level of partition can have (see [Partitioning and Parallel Matrix Assembly on page 332](#)).

`NumberOfElementsPerDomain`

Modifies the number of elements that must go to each domain (see [Partitioning and Parallel Matrix Assembly on page 332](#)).

A: Commands

math

numThreads

Specifies the number of parallel threads for Sentaurus Interconnect. Applies to matrix assembly, the box method, and the solvers PARDISO and ILS.

To run matrix assembly, the box method, or the solvers with a different number of threads, specify the following arguments:

- numThreadsAssembly
- numThreadsBoxMethod
- numThreadsILS
- numThreadsInterp
- numThreadsMGoals
- numThreadsPardiso
- numThreadsSnMesh

Separately, these arguments have priority over numThreads.

numThreadsAssembly

Number of threads used for matrix assembly.

numThreadsBoxMethod

Number of threads used for the box method.

numThreadsILS

Number of threads for the ILS solver.

numThreadsInterp

Number of threads used for interpolation. Because of the memory-intensive nature of interpolation, the performance benefit of multithreading may saturate, but it depends strongly on the simulation. If you observe saturation, reduce the value of this argument.

numThreadsMGoals

Number of threads used for MGOALS-related operations.

numThreadsPardiso

Number of threads when running PARDISO.

numThreadsSnMesh

Number of threads used when running Sentaurus Mesh.

`parallel.license`

If you run a simulation in parallel mode but the number of parallel licenses is insufficient, Sentaurus Interconnect proceeds in serial mode (default behavior or if `parallel.license=go.serial` is specified), or waits for parallel licenses (`parallel.license=go.wait`), or aborts (`parallel.license=go.abort`). The option `parallel.license=go.recheck` checks for parallel licenses at each parallel step, regardless of whether or not the licenses were available at the previous step.

If `parallel.license=go.keep`, Sentaurus Interconnect checks out a number of parallel licenses using the current value of `numThreads=<i>` at the next parallel step, but it will not return the parallel licenses to the server until the end of the simulation flow. In other words, the subsequent parallel license check will be disabled, and all modules will use the same number of parallel licenses in the entire run.

NOTE When the parallel licenses have been checked out, the new command `math numThreads=<i>` will not change the number of parallel licenses currently in use.

`threadStackSize`

Stack size for each thread. Default stack size is $2^{18} = 262144$ bytes (see [Partitioning and Parallel Matrix Assembly on page 332](#)).

Arguments: Coordinate System Input and Output Selection`coord.dfise`

Reads or writes files using the DF–ISE coordinate system. This must be used only to revert coordinate systems. If the dimension of the structure changes, `math coord.dfise` must be recalled.

`coord.transform, coord.translate, coord.read, coord.write`

Both `coord.transform` and `coord.translate` allow a general coordinate transformation through specification of a rotation matrix defined as follows:

```
coord.transform= {a11 a12 a13 a21 a22 a23 a31 a32 a33}
```

where a_{ij} (i =row, j =column) are the members of the rotation matrix, and:

```
coord.translate= {x y z}
```

specifies a translation vector.

Both `coord.transform` and `coord.translate` must be used with either `coord.read` or `coord.write` to indicate the transformation is specifying the transformation for reading or writing, respectively. When specifying `coord.write`, the inverse of the specified transformation is applied when reading. When `coord.read` is specified, the inverse transformation is applied when writing.

A: Commands

math

`coord.ucs, coord.xyz`

Reads and writes files in the UCS.

`coord.yxz`

Same as `coord.dfise` in two dimensions.

`coord.-zyx`

Same as `coord.dfise` in three dimensions.

Arguments: General

`use.interpolated.geom.coeff`

Switches on the method to be used for interpolating geometric coefficients for TRBDF, which reduces the number of box method calls by one third.

`voronoitriangle`

Switches on the internal box method calculation.

Description

This command is used to specify the:

- Different coordinate systems.
- Number of threads and the arguments used for parallel processing on shared-memory computers.
- Default options on the matrix packages to be used for different equations.
- Arguments for the Newton method.
- Different time discretization schemes.

Examples

Use the ILS solver for mechanics in the 2D case:

```
math flow dim= 2 ils
```

Use the PARDISO solver with two threads for the PDE system in the 2D case and specify nested dissection (ND) ordering for PARDISO:

```
math compute dim= 2 pardiso numThreadsPardiso= 2 scale  
pdbSetDouble Pardiso.Ordering 2
```

NOTE To run in parallel mode, the solvers PARDISO and ILS must be used with ND ordering for both the 2D and 3D cases. For example, to specify ND ordering, use:

```
pdbSetDouble Pardiso.Ordering 2  
pdbSet Math compute 2D ILS.symmOrdering nd
```

Select the solver ILS for all equations in two dimensions. Print Newton statistics at the end of each `compute` command. In the first case, the modified Newton method and the TR-BFDF(2) methods are used. In the second case, the Euler and the full Newton methods are specified:

```
math dimension= 2 ils newtonStats= 1  
math dimension= 2 ils newtonStats= 1 euler fullNewton
```

mgoals

Modifies the default parameters for geometric operations available in the MGOALS module.

Syntax

```
mgoals
[accuracy=<n>] [<m>|<cm>|<um>|<nm>]
[analytic.thickness=<n>] [<m>|<cm>|<um>|<nm>]
[aniso.etching.fragment.tol=<n>]
[aniso.etching.protect.materials]
[dx=<n> dy=<n> dz=<n>]
[fill.buried] [fill.buried.interface=<material>]
[fill.buried.match.surrounding]
[force.analytic | force.full.levelset | force.full.levelset.depo |
  force.full.levelset.etch]
[fourier.local.diffusivity]
[full.resolution=<n>]
[keep.closed.voids]
[max.number.levelset.cells=<n>]
[min.gas.thickness=<n>] [<m>|<cm>|<um>|<nm>]
[min.levelset.size=<n>] [<m>|<cm>|<um>|<nm>]
[print.params]
[reinitfrequency=<n>] [reinititerations=<n>]
[remove.floating.regions]
[repair.2d] [repair.2d.deposit] [repair.2d.etch] [repair.2d.photo]
[repair.3d] [repair.3d.deposit] [repair.3d.etch] [repair.3d.photo]
[repair.angle=<n>] [repair.crystal.depo.facets]
[resolution=<n>]
[trapezoidal.algorithm=<n>] [use.brep.2d]
[M-2016.12-SP2 | M-2016.12-SP1 | M-2016.12 | L-2016.03-SP2 | L-2016.03-SP1 |
  L-2016.03 | K-2015.06-SP2 | K-2015.06-SP1 | K-2015.06 | J-2014.09-SP2 |
  J-2014.09-SP1 | J-2014.09 | I-2013.12-SP2 | I-2013.12-SP1 | I-2013.12 |
  H-2013.03-SP2 | H-2013.03-SP1 | H-2013.03 | G-2012.06-SP2 | G-2012.06 |
  F-2011.09-SP1 | F-2011.09 | E-2010.12 | D-2010.03]
```

Arguments

accuracy

Specifies the error that can be tolerated in transferring the new interface definition from the level-set grid to the simulation grid. There is a compromise between smoothness and the number of grid points. Smoother grids need more points on curved regions. Default value and unit: $1.0 \times 10^{-5} \mu\text{m}$.

`analytic.thickness`

For etching and deposition steps of layers of thickness of 1 nm or less, an analytic method performs the etching because thin etches using the level-set method can be prohibitively CPU and memory intensive.

For very large or very small structures, a 1-nm cutoff may be inappropriate, so this argument can be used to modify the thickness where the analytic method is used. Default unit: μm .

`aniso.etching.fragment.tol`

Removes fragments remaining from 3D anisotropic etching. The tolerance measures the ratio of the volume and the surface of a region. Default: 1.0e-6.

`aniso.etching.protect.materials`

When switched on, the 3D anisotropic algorithm attempts to protect areas shadowed by buried materials. Default: false.

`dx, dy, dz`

Explicitly set the level-set grid spacing in each direction. If set, these arguments override the automatic setting of `dx`, `dy`, and `dz`, which uses `resolution`.

`fill.buried`

For deposition, material is deposited on the surface exposed to the upper gas region. With `fill.buried` specified, deposition also occurs inside the buried gas bubbles that may exist.

`fill.buried.interface`

Specifies the material to fill gas bubbles at material interfaces. The material must already be present in the simulation.

NOTE This argument applies only to 3D deposition.

`fill.buried.match.surrounding`

Similar to `fill.buried`, but instead of depositing the deposition material in gas bubbles, this option fills gas bubbles that are completely surrounded by a single material with that material, removing the bubbles from the domain entirely. Gas bubbles at material interfaces remain unfilled but can be filled using `fill.buried.interface`.

NOTE This argument applies only to 3D deposition.

A: Commands

mgoals

`force.analytic`

When performing isotropic etching or deposition, this argument permits the use of an analytic algorithm even when a boundary collision will occur. Otherwise, self-intersections will force the use of the level-set algorithm. For very large structures or very small etching or deposition thicknesses, the level-set algorithm may consume too much memory and time.

`force.full.levelset`, `force.full.levelset.depo`, `force.full.levelset.etch`

Defines the general level-set time-stepping algorithm as the default algorithm for both etching and deposition, or etching only, or deposition only, respectively.

`fourier.local.diffusivity`

Controls the artificial diffusion parameter. If `fourier.local.diffusivity` is specified, the solution is more accurate but the corners are less sharp. For complex Fourier rates, switch off `fourier.local.diffusivity` to enhance stability.

`full.resolution`

Usually, the full-time stepping level-set method is used in situations where more intricate boundaries will be generated. Default unit: μm .

The full-time stepping level-set method is needed for Fourier, crystallographic, and multimaterial etching types, and for etching with shadowing on. This argument allows a separate resolution setting for these cases. The default value is 0.025 compared to 0.1 for resolution.

`keep.closed.voids`

In the case of crystallographic deposition, enclosed voids may continue to grow and fill up. For efficiency, the algorithm checks for such voids only at the beginning of the deposition. Specifying this option instructs the algorithm to check for enclosed voids at each deposition time step, but at the cost of additional computational time.

`max.number.levelset.cells`

Specifies the maximum number of cells used by the level-set mesh. If this limit is exceeded, a warning message is issued. Default: 1.0×10^9 (or $1000 \times 1000 \times 1000$ cells).

`min.gas.thickness`

Minimum thickness of the gas layer at the top of the simulation structure. Default value and unit: $0.1 \mu\text{m}$.

`min.levelset.size`

Specifies the minimum size for the level-set mesh. Usually, the level-set mesh size scales with the operation according to the resolution factor and the etching or deposition thickness. However, for thin etching or deposition steps, this may lead to a small level-set mesh causing excessive memory use and simulation time. Often, it is not necessary (for thin layers, a mesh size between $\text{thickness}/2.0$ and $\text{thickness}/3.0$ is usually sufficient). This argument limits the mesh size and, therefore, limits computational expense. Default value and unit: $1.0 \times 10^{-4} \mu\text{m}$.

`print.params`

Prints the current MGOALS parameters.

`reinitfrequency`

Level-set reinitialization is performed every `reinitfrequency` time step in level-set operations. A reinitialization algorithm is run to condition the level-set distance function to reduce the effect of *contour bunching*, which can cause etching distances to be less than expected. The default value is 0, which means that no reinitialization is performed.

`reinititerations`

The internal reinitialization algorithm reinitializes first the 0 level set and works outwards from the front with higher numbers of iterations. Default: 1. This argument only comes into operation if `reinitfrequency` is nonzero.

`remove.floating.regions`

Determines whether MGOALS automatically removes regions that are not attached to the bottom of the structure. Default: true.

`repair.2d`, `repair.2d.deposit`, `repair.2d.etch`, `repair.2d.photo`
`repair.3d`, `repair.3d.deposit`, `repair.3d.etch`, `repair.3d.photo`

Controls the default behavior of the boundary repair operation in two and three dimensions. The options also control whether repairs are enabled in etching and deposition. By default, boundary repairs are disabled in two dimensions and enabled in three dimensions.

`repair.angle`

Controls the dihedral angle at which repairs are performed. The algorithm attempts to repair any surface section with a dihedral angle less than `repair.angle`. Default value and unit: 1° .

`repair.crystal.depo.facets`

During crystallographic deposition, boundary conditions near neutral walls can lead to a moving front that is perpendicular to the wall, resembling a break in facets. This option

A: Commands

mgoals

allows additional control by adding a repair step to make the surface near neutral walls more smooth.

resolution

Specifies the minimum number of level-set cells across the thickness of a deposited or etched layer. For example, `resolution=0.2` implies five cells. Default: 0.1.

trapezoidal.algorithm

Sets the algorithm used for trapezoidal etching:

- A value of 2 selects the preferred, newer algorithm, which improves the quality of produced shapes, resulting in better meshes after etching.
- A value of 1 selects the earlier algorithm.

Default: 2.

use.brep.2d

Switches on the brep structure mode for two dimensions when handling structural changes such as 2D etching and 2D deposition. Default: false.

The boundary representation (brep) structure mode in two dimensions handles structural changes similarly to how structural changes are handled by default in three dimensions. A brep of the structure is used and modified rather than the volume mesh.

Using brep reduces runtimes by avoiding unnecessary meshing operations and increases stability and accuracy by eliminating both boundary simplification and variable interpolation associated with remeshing between structural modification operations.

M-2016.12-SP2, M-2016.12-SP1, M-2016.12, L-2016.03-SP2, L-2016.03-SP1, L-2016.03, K-2015.06-SP2, K-2015.06-SP1, K-2015.06, J-2014.09-SP2, J-2014.09-SP1, J-2014.09, I-2013.12-SP2, I-2013.12-SP1, I-2013.12, H-2013.03-SP2, H-2013.03-SP1, H-2013.03, G-2012.06-SP2, G-2012.06, F-2011.09-SP1, F-2011.09, E-2010.12, D-2010.03

Sets the backward compatibility of parameters and algorithms to the specified version. Support is available for all the listed versions except Version D-2010.03, for which there is only partial support.

Arguments: Deprecated

The following arguments are deprecated since they belong to the deprecated MGOALS mesh generator: `max.box.angle`, `max.lateral.size`, `max.neighbor.ratio`, `minedge`, `min.normal.size`, `normal.growth.ratio`, and `offsetting.maxlevel`.

Instead, use the equivalent Sentaurus Mesh `pdb` parameters (see [Table 17 on page 188](#)).

The following arguments are deprecated and have no equivalent parameters in Sentaurus Mesh: `maxangle`, `nlayers`, `sliver.split`, `sliver.smooth`, and `sliver.swap`.

It is suggested that you remove these arguments from command files since they may not be accepted in future releases.

Description

This command allows you to define parameters for MGOALS-related operations.

Examples

Explicitly set the vertical level-set mesh spacing to $0.01\ \mu\text{m}$ and the horizontal level-set mesh spacing to $0.02\ \mu\text{m}$. Reinitialization of the level-set distance function is performed every five time steps, and every reinitialization is performed to an internal iteration accuracy of four iterations:

```
mgoals dx= 0.01 dy= 0.02 reinitfrequency= 5 reinititerations= 4
```

mobility

Calculates the mobility enhancement for a planar device type or a 3D FinFET device type.

Syntax

```
mobility
  device.model= finfet | planar
  region.name=<c>
  [carrier.type= 0 | 1]
  [channel.orient= 1 | 2]
  [piezo.coefficients= {P11=<n>, P12=<n>, P44=<n>}]
```

Arguments

`carrier.type`

Specifies the carrier type as either electron (0) or hole (1). Default: 0.

`channel.orient`

Specifies the fin orientation. Use `channel.orient=1` for a (110) orientation of the fin. Use `channel.orient=2` for a (100) orientation of the fin. Default: 1.

NOTE If a planar device type is specified, `channel.orient` and `carrier.type` are not required.

`device.model`

Specifies the type of device as either a planar device or 3D FinFET.

`piezo.coefficients`

Specifies the piezo coefficients P11, P12, and P44 for the material.

`region.name`

Specifies the name of the region where the mobility enhancement is calculated.

Description

The `mobility` command calculates the mobility enhancement for a planar device type or a 3D FinFET device type.

Examples

Calculate the mobility enhancement in region `Silicon` using the planar model and the specified piezo coefficients:

```
mobility device.model=planar region.name=Silicon \  
  piezo.coefficients= {P11=-6.6, P12=1.1, P44=-138.1}
```

mode

Specifies the simulation mode.

Syntax

```
mode
  [current | electrostatic] [gas] [mechanics] [thermal]
  (
    [auto.segment] |
    [bias.type=<c>] |
    ([capacitance | capacitance= {<c> <c> ...}] |
    [resistance | resistance= {<c> <c> ...}] |
    [thermal.RC.network= {"<c>[=<power>]"} ("<ci> ... <cm>")
      [{"<cn> ... <cp>"}]]) |
    [thermal.resistance | thermal.resistance= {<c> <c> ...}]) |
    [ground] |
    [thermal.global.model=<c>]
  )
```

Arguments

auto.segment

Use with resistance analysis to automatically segment the structure and to place contacts automatically at conductor-conductor interfaces. If `boundary.contacts` is specified in the `option` command, contacts are also created automatically at the outer boundary of conductor regions.

bias.type

If no value is set or is set to `ground`, each contact in the listed contacts is solved with a unit supply and all the rest are set to `ground`. If the value is set to `!ground`, each pair of contacts in the listed contact is solved with a unit and zero supplies, and all the rest are set to `!ground`. This argument can be used for capacitance, resistance, or thermal resistance simulations. Default: `ground`.

NOTE The simulation time might increase significantly due to more solving combinations when the value is set to `!ground`.

capacitance

Calculates the capacitance matrix for the listed contacts. All unused contacts are grounded and lumped into one contact called `GROUND_RCX`. If no contact list is given, that is, only `capacitance` is specified, it calculates the capacitance matrix between conducting layers in the structure.

NOTE The contact list must consist of only region names or only contact names.

current, electrostatic

Select `current` to perform electrical current analysis in conducting regions.

Select `electrostatic` to perform electrostatic analysis in dielectric regions.

NOTE The `current` and `electrostatic` modes cannot be set simultaneously. All other simulation modes can be set with other modes.

gas

Sentaurus Interconnect solves the equation for Potential in the materials `Gas` and `BackMat` in the `electrostatic` and `capacitance` modes (in the `current` and `resistance` modes, the equation for Potential in those materials is never solved). If `!gas` is specified, the equation for Potential in the materials `Gas` and `BackMat` is also not solved in the `electrostatic` and `capacitance` modes. Default: `true`.

ground

If specified, this option grounds all unused contacts during `capacitance`, `resistance`, or `thermal resistance extraction`. Default: `true`.

mechanics

Performs stress analysis.

resistance

Calculates the resistance matrix for the listed contacts. All unused contacts are grounded and lumped into one contact called `GROUND_RCX`. If no contact list is given, that is, only `resistance` is specified, it calculates the resistance matrix between user-defined contacts in the structure.

NOTE The contact list must consist of only region names or only contact names.

thermal

Performs thermal analysis in all regions.

`thermal.global.model`

Extracts the temperature field from the global model, interpolates the field on to the local refined model, applies the temperature to the outer surfaces of the local model as boundary conditions, and performs thermal analysis in all regions.

A: Commands

mode

`thermal.RC.network`

Calculates the thermal RC networks for the listed contacts. Contacts that are not listed are excluded from the simulation. The position of the contact groups in the list of contact names is important:

The first group of contacts must consist of one contact name ("`<c> [=<power>]`"), which is the name of the excitation contact. Only one excitation contact must be given. The optional value `<power>` can be specified after the name of the excitation contact. It corresponds to the excitation power (in W) and must be a positive number. Default value and unit: 1.0e-3 W.

The second group of contacts is a list of ambient contacts ("`<ci> ... <cm>`"), which stay at ambient temperature. The ambient temperature is taken from the PDB parameter `EnvTemp [pdbGet EnvTemp]`, which is 26.844°C by default. At least one ambient contact must be specified. The list must not include the name of the excitation contact.

The third group is an optional list of floating contacts ("`<cn> ... <cp>`") for which interactive-heating thermal RC networks are calculated. They stay at a condition of zero heat flux. It must not include the name of the excitation contact or the names of any ambient contacts. If no floating contact is specified, all other contacts except the excitation contact and the ambient contacts are ignored and only a self-heating thermal RC network for the excitation contact is calculated.

`thermal.resistance`

Calculates the thermal resistance matrix for the listed contacts. All unused contacts are grounded and lumped into one contact called `GROUND_RCX`. If no contact list is given, that is only `thermal.resistance` is specified, it calculates the thermal resistance matrix between user-defined contacts in the structure.

NOTE The contact list must consist of only region names or only contact names.

Description

Electric current analysis, electrostatic analysis, mechanics (stress analysis), and thermal analysis are solved.

Examples

Perform stress analysis and electrical current analysis:

```
mode mechanics current
```

Perform stress analysis and thermal analysis:

```
mode thermal mechanics
```

See Also

[Chapter 3 on page 45](#)
[Chapter 5 on page 121](#)
[Chapter 6 on page 131](#)
[solve on page 629](#)

optimize

Optimizes the specified parameters to achieve the required target values.

Syntax

```
optimize
  model.function=<c>
  model.parameters= {<c1> <c2> <c3> ...}
  (target= <list> | target.file=<c>)
  [history=<c>]
  [max.iter=<n>]
  [min.abs= <list>]
  [min.rel= <list>]
  [param.init= {<c1>=<n> <c2>=<n> <c3>=<n> ...}]
  [param.log= {<c1> <c2> ...}]
  [param.lower= {<c1>=<n> <c2>=<n> <c3>=<n> ...}]
  [param.upper= {<c1>=<n> <c2>=<n> <c3>=<n> ...}]
  [tolerance=<n>]
  [weight= <list>]
```

Arguments

history

Specifies the file where the history of the optimization is stored. The history of the parameters, the corresponding target results, and the RMS errors are displayed in columns with each row corresponding to a loop.

max.iter

Specifies the maximum number of iterations allowed in the optimization loop. Default: 500.

min.abs

Lists the minimum absolute error for each target. If specified, the list must be the same length as the number of targets. It is the minimum target value for which the absolute error is used to calculate the target error during optimization. This value is compared with the absolute target value. The default is given by `pdbGetDouble Optimizer min.abs`, which defaults to 1.e-10 for all targets.

min.rel

Lists the minimum relative error for each target. If specified, the list must be the same length as the number of targets. It is the minimum target ratio for which the relative error is used to calculate the target error during optimization. This value is compared with the

ratio of the absolute target value to the maximum absolute target value. The default is given by `pdbGetDouble Optimizer min.rel`, which defaults to 1.e-10 for all targets.

`model.function`

Name of a user-defined Tcl procedure. The arguments of the procedure are those specified in `model.parameters` and are in the same order as in `model.parameters`. The procedure computes and returns the results of the model.

`model.parameters`

Lists the names of the parameters to be optimized.

`param.init`

Lists the initial values of the parameters to be optimized. If not specified, the default value is 0.5.

`param.log`

Lists the names of any parameter whose value you want to vary logarithmically during optimization.

`param.lower`

Lists the lower bound values of the parameters to be optimized. If not specified, the default value is 0.0.

`param.upper`

Lists the upper bound values of the parameters to be optimized. If not specified, the default value is 1.0.

`target`

Lists the required target data that the optimization will attempt to achieve by varying the values of parameters listed in `model.parameters`.

NOTE If you use `target` for a fitting problem, you can set up the independent variable data using a global Tcl variable and use it in your user-defined Tcl procedure specified by `model.function`.

You can specify either `target` or `target.file`, but not both.

A: Commands

optimize

target.file

Specifies a text file (SIMS data, for example) that can be used to set target values. The file has two columns of data: The first column contains independent variable data, and the second column contains the corresponding target data. The independent variable and the target data must be in pairs. For convenience, the independent variable is saved in a global Tcl variable named `opt.independent.var` when loading the target file. You can use it directly in your user-defined Tcl procedure specified by `model.function`.

You can specify either `target` or `target.file`, but not both.

tolerance

Specifies the root-mean-square (RMS) error for convergence criterion. The optimization terminates when the RMS error of targets is smaller than `tolerance`. The default value is given by `pdbGetDouble Optimizer Tolerance`, which defaults to 0.1.

weight

Lists the weighting factors for targets. If specified, the list must be the same length as the number of targets. The weights control the importance of individual targets in calculations of the error during optimization. For a fitting problem, you do not need to specify `weight` because all targets have the same importance.

The default is given by `pdbGetDouble Optimizer Weight`, which defaults to 1.0 for all targets.

Description

This command optimizes the specified parameters to achieve the required target values. Before using the `optimize` command, you must define a Tcl procedure that computes the results of the model, given a set of valid model parameters. The result of the command is a Tcl list with the optimized parameters in the format:

```
{<c1> <n> <c2> <n> ... rmsError(%) <n> TotalLoops <n>}
```

Examples

The following example is a procedure for a fitting problem. The global Tcl variable `opt.independent.var` was set to store the independent variable data after the input target file is loaded. You can use it directly in the user-defined Tcl procedure:

```
fproc optFunc {cja vja mja} {
    global {opt.independent.var}
    foreach value ${opt.independent.var} {
        lappend z [expr $cja/(1+$value/$vja)**$mja]
    }
    return $z
}
```

The optimize command with the above procedure would be:

```
optimize model.function= optFunc model.parameters= {cja vja mja} \  
  param.init= {cja=0.2 vja=5. mja=0.5} \  
  param.lower= {cja=0.1 vja=2. mja=0.1} \  
  param.upper= {cja=0.4 vja=7. mja=0.9} \  
  target.file= model.dat \  
  max.iter= 500 \  
  tolerance= 0.1 \  
  history= test.dat
```

option

Sets the storage and run options.

Syntax

```
option
[boundary.contacts]
[current= <list>]
[file.name=<c>]
[lumped.materials= <list>]
[max.optimization.iter=<i>]
[optimization.history.file=<c>]
[optimization.tolerance=<n>]
[param.init= {R1=<n> C1=<n> R2=<n> C2=<n> ...}]
[param.log= {R<i> C<i>}]
[param.lower= {R1=<n> C1=<n> R2=<n> C2=<n> ...}]
[param.upper= {R1=<n> C1=<n> R2=<n> C2=<n> ...}]
[probe.time= {<n> <n>}]
[resistance= <list>]
[struct= ["not set" |
  [{
    [Adaptive] [alt.maternames] [bnd] [contacts] [csse]
    [deform] [deform.scale=<n>]
    [FullD] [Gas] [interfaces]
    [mshcmd] [pdb] [pdb.only]
    [sat] [satfile=<c>] [simplify=<n>]
    [tdr=<c>] [tdr.bnd=<c>]
    [visual.1D]
    ([x=<n>] [<m>|<cm>|<um>|<nm>]
     [y=<n>] [<m>|<cm>|<um>|<nm>]
     [z=<n>] [<m>|<cm>|<um>|<nm>]) }} ]
[temperature= <list>]
[thermal.RC.network.size=<i>]
[voltage= <list>]
```

Arguments

boundary.contacts

Specify this option to create contacts automatically at the outer boundary of conductor regions in distributed RC simulations. Default: false.

current

List of contacts or pair of circuit element and a node, where total current-versus-time information will be stored.

file.name

Name of output file to store data (or use [WritePlx on page 681](#)).

lumped.materials

Specifies a list of conductor materials to be merged in a distributed RC simulation.

max.optimization.iter

Maximum number of iterations for fitting the parameters of thermal RC networks.
Default: 500.

optimization.history.file

Specifies the prefix of the file name for storing the optimization logs and used during thermal RC network extraction. For example, the optimization logs are saved in files with the following names:

```
<file_name_prefix>_<ci>-<cj>_opt_sis.log
```

Here, <ci> is the name of the excitation contact, and <cj> is the name of the response contact (<ci> can be equal to <cj>).

If this argument is not specified, the optimization logs are not saved.

optimization.tolerance

Specifies the optimization precision (tolerance) for computing the thermal RC network elements in rms%. Default: (0.1 rms%).

param.init

Specifies a list of initial parameter values for thermal RC network extraction. $R_{<i>}$ are specified in K/W, and $C_{<i>}$ are specified in J/K. If this argument is not specified, all $R_{<i>} = 1.0$ K/W and all $C_{<i>} = 1.0e-7$ J/K. If some initial values are set manually and some are not, the remaining parameters, which were not set manually, receive values that are the default for param.init of the optimize command, that is, 0.5.

param.log

Specifies a list of names of any parameter for thermal RC network extraction whose value you want to vary logarithmically during optimization. The remaining parameters that are not specified here are varied linearly. If param.log is not specified, all $R_{<i>}$ and $C_{<i>}$ are varied logarithmically.

param.lower

Specifies a list of the lower bound values of parameters during thermal RC network extraction. $R_{<i>}$ are specified in K/W, and $C_{<i>}$ are specified in J/K. If this argument is not specified, all $R_{<i>} = 1.0e-6$ K/W and all $C_{<i>} = 1.0e-20$ J/K. If some lower bound

A: Commands

option

values are set manually and some are not, the remaining parameters, which were not set manually, receive the values that are the default for `param.lower` of the `optimize` command, that is, 0.0.

`param.upper`

Specifies a list of the upper bound values of parameters during thermal RC network extraction. $R_{<i>}$ are specified in K/W, and $C_{<i>}$ are specified in J/K. If this argument is not specified, all $R_{<i>} = 1.0e6$ K/W and all $C_{<i>} = 1.0e4$ J/K. If some upper bound values are set manually and some are not, the remaining parameters, which were not set manually, receive the values that are the default for `param.upper` of the `optimize` command, that is, 1.0.

`probe.time`

List of time points the solution will be forced to move through if specified.

`resistance`

List of pair of contacts where resistance-versus-time information will be stored.

`struct`

Specifies a list of arguments for the `struct` command that will be executed at the simulation times specified by the `probe.time` argument of the `option` command. The names of the saved files will have the suffix `_<time>s`, where `<time>` is the current simulation time (in seconds) in exponential format without a decimal point, consisting of its first seven significant digits.

If no value is given for the `struct` argument, the previously specified value will be used. To completely switch off the `struct` command that was previously set by an `option` command, specify `struct= "not set"`. All standard arguments of the `struct` command are supported in the `option` command. For descriptions of the arguments that can be listed in the `struct` argument, see [struct on page 643](#).

NOTE If zero time is specified for `probe.time`, the corresponding TDR file saved at `time=0` may not have the mechanics solution properly updated at zero time. For example, internal stresses set by the `stressdata` command will not be rebalanced at zero time. Whereas, thermal stresses will be rebalanced when `Mechanics StressHistory 1` is set in the PDB. In addition, for moving-mesh problems, the mechanics solution corresponding to the first time step will be saved at zero time.

`temperature`

List of contacts or circuit nodes where temperature-versus-time information will be stored.

`thermal.RC.network.size`

Specifies the number of RC cells in the thermal RC network. Default: 3.

`voltage`

List of contacts or circuit nodes where total voltage-versus-time information will be stored.

Description

This command stores time-versus-total current, or time-versus-total voltage at contacts, or time-versus-resistance between pairs of contacts. For circuit simulations, it stores time-versus-total current between a circuit element and node and time-versus-total voltage at a node. It may save TDR and SAT files at simulation times equal to `probe.time` values. In addition, the size of the thermal RC network and the optimization parameters for extraction of the elements of the thermal RC networks can be specified.

Examples: Usage for Structure Only

Store time-versus-resistance information between the contacts `upper` and `lower` during time-dependent simulations:

```
option resistance= { {upper lower} {c1 c2} }
```

For resistance information of multiple pairs of contacts, use:

```
option resistance= { {upper lower} }
```

Store time-versus-total current at the contacts `upper` and `lower`, and time-versus-total voltage at the contacts `upper` and `lower`:

```
option current= {upper lower} voltage= {upper lower}
```

Store time-versus-total temperature at the contacts `tm0` and `tm1`:

```
option temperature= {tm0 tm1}
```

Examples: Usage for Mixed Mode

Store time-versus-total current between the pairs of circuit element `r1` and node `left`, and also between circuit element `V2` and node `grnd`:

```
option current= { {r1 left} {V2 grnd} }
```

Store time-versus-total voltage at the nodes `n1` and `n2`:

```
option voltage= {n1 n2}
```

A: Commands

option

Store time-versus-temperature at the nodes n1 and n2:

```
option temperature= {n1 n2}
```

Force the solution to move through the time points specified:

```
option probe.time= {0.08<s> 0.2<min>}
```

Examples: Usage of struct argument

Store TDR files at certain simulation times, and save the `test_8000000e-09s_sis.tdr` and `test_1200000e-05s_sis.tdr` files (without using the PDB) during the corresponding subsequent `solve` command:

```
option probe.time= {0.08<s> 0.2<min>} struct= {tdr=test !pdb}  
solve time=1.0<min>
```

Completely suppress the `struct` argument in the `option` command:

```
option struct= "not set"
```

See Also

[optimize on page 522](#)

[struct on page 643](#)

[WritePlx on page 681](#)

paste

Assembles 2D or 3D simulations by incorporating fragments from a TDR file.

Syntax

```
paste
  tdr=<c>
  direction= "left" | "right" | "front" | "back"
  [Adaptive]
```

Arguments

Adaptive

If specified, `Adaptive` switches on adaptive meshing. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

direction

Selects the side where to paste the incoming simulation. In three dimensions, the new structure can be pasted on the left, right, front, or back side. In two dimensions, only the left or right sides are used.

tdr

Name of the file to be imported and pasted into the current simulation.

Description

This command reads a TDR file containing valid geometry and appends it (pastes it) to the current structure. The new structure is displaced automatically by the correct amount to properly fit at the specified side, but the structures are not stretched automatically. If the dimensions of nongas materials at the pasting sides are not the same, the command fails and quits.

The dimension of all parts must match: You cannot mix two and three dimensions. In addition, in two dimensions, only the `direction` values "left" and "right" are supported.

The `paste` command allows for the assembly of complex 2D or 3D structures by reading the different pieces from TDR files and putting all of them together.

A: Commands

paste

Examples

Append the structure from the file `propertyx_sis.tdr` to the right side (maximum y-coordinate) of the current structure:

```
paste tdr= "propertyx" direction= "right"
```

See Also

[Inserting Polygons on page 274](#)

[struct on page 643](#)

pdbDelayDouble

Retrieves an expression for a double parameter that will be evaluated at each time step during the solution of the thermal and electrical equations.

Syntax

```
pdbDelayDouble <c> <c> ...
```

Arguments

<c>

This argument can be any double parameter declared in the parameter database and any double parameter declared by the user in user-defined models. In all cases, the argument must be specified with the full hierarchical path for the double parameter.

Description

This command is typically called from Alagator to retrieve a parameter expression. Since among other things, the temperature can change during a solve step, the evaluation of Arrhenius expressions must be delayed until the temperature is known. This command provides this functionality.

Examples

Return an expression for D0 (not a value):

```
pdbDelayDouble Si B D0
```

pdbdiff

Compares the current structure with one from a TDR file.

Syntax

```
pdbdiff <c> <c>
```

Arguments

<c>

Specifies the full path or prefix of the TDR files to be compared. The prefix is the file name without `_sis.tdr`.

Description

The command reports any differences between the parameters stored and any differences in value.

Examples

Compare the pdb differences between `n1_sis.tdr` and `n2_sis.tdr`:

```
pdbdiff n1 n2
```

pdbExprDouble

Retrieves an expression for a double parameter without evaluating.

Syntax

```
pdbExprDouble <c> <c> ...
```

Arguments

<c>

This argument can be any double parameter declared in the parameter database and any double parameter declared by the user in user-defined models. In all cases, the argument must be specified with the full hierarchical path for the double parameter.

Description

This command is typically called from Alagator to retrieve a parameter expression. If the parameter depends on solution names, data fields, and so on, the evaluation of the expression must be delayed until the solution time. This command provides this functionality.

Examples

Return an expression for Bulk (not a value):

```
pdbExprDouble Si Mechanics BulkModulus
```

A: Commands

pdbGet and Related Commands

pdbGet and Related Commands

All these commands retrieve database parameters:

- `pdbGet`
- `pdbGetArray`
- `pdbGetBoolean`
- `pdbGetDouble`
- `pdbDelayDouble`
- `pdbGetDoubleArray`
- `pdbGetElement`
- `pdbGetFunction`
- `pdbGetString`
- `pdbGetSwitch`
- `pdbGetSwitchString`

Only `pdbGet` has syntax checking.

Syntax

```
pdbGet <c> <c> ...
```

Arguments

<c>

This argument can be any double parameter declared in the parameter database and any double parameter declared by the user in user-defined models. In all cases, the argument must be specified with the full hierarchical path for the double parameter.

Description

These commands are used to obtain parameters that reside in the property database, which is hierarchical and is indicated by passing a series of strings to the command. In the command file, the command `pdbGet` must replace all other `pdbGet*` commands because the type of the parameter and the syntax are checked automatically.

If a parameter does not exist in the directory, the tool exits and prints a list of parameters that can be found. Normal aliasing is applied to each string before the parameter is retrieved from the database.

The following commands all return 0 if the parameter is not found:

- `pdbGetArray`
- `pdbGetBoolean`
- `pdbGetDouble`
- `pdbDelayDouble`
- `pdbGetDoubleArray`
- `pdbGetElement`
- `pdbGetFunction`
- `pdbGetString`
- `pdbGetSwitch`

The command `pdbGetSwitchString` returns nothing if the parameter is not found.

These commands have a slight performance advantage and will not exit if a parameter has not been defined, so they are preferred for Alagator scripting.

The command `pdbGetSwitch` returns an integer value of a switch, and the command `pdbGetSwitchString` returns the string value. For example, if a switch has the choices a, b, or c, and a is chosen, `pdbGetSwitch` returns 0, and `pdbGetSwitchString` returns a.

Examples

Retrieve the current value of `StressHistory`. The parameter `StressHistory` is known, but if it is spelled incorrectly, Sentauros Interconnect exits and prints a list of known parameters at the `Mechanics` level:

```
pdbGet Mechanics StressHistory
```

Retrieve `StressHistory` without syntax-checking. The command returns 0 if not found. Sentauros Interconnect exits if there is a type mismatch between `StressHistory` and `Boolean` (which is not the case in this example):

```
pdbGetBoolean Mechanics StressHistory
```

pdbIsAvailable

Checks whether the given pdb command is available.

Syntax

```
pdbIsAvailable <c> <c> ...
```

Arguments

<c>

Specifies the pdb command.

Description

If the pdb command exists, the `pdbIsAvailable` command returns 1; otherwise, it returns 0.

Examples

Return 1 if the command "Silicon MyData" is available:

```
pdbIsAvailable Silicon MyData
```

pdbLike

Creates a new `pdb` parameter like an existing parameter in a material.

Syntax

```
pdbLike <c> <c> <c>
```

Arguments

<c>

The first argument must be the name of the material.

The second argument must be the name of the new `pdb` parameter.

The third argument must be the name of an existing `pdb` parameter.

Description

This command is used to create a new node in the parameter database where the sub-parameters of this node are inherited from an existing node in the database. It is typically used when creating a new solution variable to inherit parameters from an existing solution variable.

Examples

Create a new parameter called `MyPotential` in silicon:

```
pdbLike Silicon MyPotential Potential
```

`MyPotential` inherits all the parameters defined for `Potential` in silicon including the user-defined ones.

See Also

[solution on page 626](#)

pdbSet and Related Commands

All of the following commands set database parameters:

- `pdbSet`
- `pdbSetArray`
- `pdbSetBoolean`
- `pdbSetDouble`
- `pdbSetDoubleArray`
- `pdbSetElement`
- `pdbSetFunction`
- `pdbSetString`
- `pdbSetSwitch`

Only `pdbSet` has syntax checking.

Syntax

```
pdbSet <c> <c> ... <value>
```

Arguments

<c>

This argument can be any parameter declared in the parameter database and any parameter declared by users in user-defined models. In all cases, the argument must be specified with the full hierarchical path for the parameter.

<value>

The value associated with the type of the parameter. For example, a double value must be given for a `Double` parameter type.

Description

These commands are used to set parameters that reside in the property database, which is hierarchical and is indicated by passing a series of strings to the command. In the command file, the command `pdbSet` must replace all other `pdbSet*` commands because the type of the parameter and the syntax are checked automatically. If a parameter does not exist in the directory, the tool exits and prints a list of parameters that can be found. Normal aliasing is applied to each string before the parameter is retrieved from the database for all these commands.

The following commands all create a new parameter if one does not already exist:

- `pdbSetArray`
- `pdbSetBoolean`
- `pdbSetDouble`
- `pdbSetDoubleArray`
- `pdbSetFunction`
- `pdbSetString`

These commands have a slight performance advantage, and Sentaurus Interconnect does not exit if a parameter has not been defined, so they are preferred for Alagator scripting. The property database uses the CGS system of units, except for activation energies [eV].

The command `pdbSet` checks the type of variable trying to be set and checks that type against the `<value>` passed. The command `pdbSetDouble` takes a double for a value and, similarly, `pdbSetString` takes a string and `pdbSetBoolean` takes a Boolean (1 or 0). The command `pdbSetSwitch` will set a value for existing switches. If a switch is not found, a new one will be created.

The command `pdbSetArray` defines the array for string data.

The data type `DoubleArray` has a special format and can be modified in several different ways depending on the changes required.

The command `pdbSetElement` modifies the value of one element in an array.

NOTE The arguments for `pdbSetDouble`, `pdbSetDoubleArray`, and `pdbSetBoolean` must evaluate to numeric data. Calls to procedures or to the `pdbDelayDouble` command in the arguments may cause errors if they are not constructed correctly.

Examples

Set the `Emissivity` parameter in copper, and exit with a list if it is not found:

```
pdbSet Copper Emissivity 1
```

Now, for the non-syntax-checked version:

Set `Emissivity`, and create `Emissivity` if it does not already exist (in this example, `Emissivity` would exist):

```
pdbSetDouble Copper Emissivity 1
```

A: Commands

pdbSet and Related Commands

Create a new DoubleArray, index 0 = 0.1, and so on:

```
pdbSetDoubleArray Si MyVar MyArray {0 0.1 1 0.2 3 0.3}
pdbSetArray MyArray {0 abc 1 def 2 ghi}
pdbGetArray MyArray                ;# print "0 abc 1 def 2 ghi"
pdbGetElement MyArray 1             ;# print "def"
pdbSetElement MyArray 1 jkl         ;# modifies A[1] data from "def" to "jkl"
```

pdbUnSet-Related Commands

All these commands unset database parameters:

- `pdbUnSetBoolean`
- `pdbUnSetDouble`
- `pdbUnSetDoubleArray`
- `pdbUnSetString`

Syntax

```
pdbUnSetBoolean <c> <c> ...
```

```
pdbUnSetDouble <c> <c> ...
```

```
pdbUnSetDoubleArray <c> <c> ...
```

```
pdbUnSetString <c> <c> ...
```

Arguments

<c>

This argument can be any parameter declared in the parameter database and any parameter declared by users in user-defined models. In all cases, the argument must be specified with the full hierarchical path for the parameter.

Description

This command temporarily removes parameters from the parameter database during the simulation.

Examples

Remove the `Derived.Materials` list of `InGaAs` material:

```
pdbUnSetString InGaAs Derived.Materials
```

photo

Creates a photoresist layer of the specified thickness outside the mask.

Syntax

```
photo
  [Adaptive] [mask=<c>] [repair] [sde= {<c>}]
  [thickness=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

Adaptive

If specified, `Adaptive` switches on adaptive meshing. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

mask

Name of the mask to be used to create the photoresist. The photoresist is deposited in the openings of the mask.

repair

In 3D MGOALS mode, small regions are removed automatically by default. Sometimes, this causes small gas bubbles in the structure or other problems. Use `!repair` to switch off the small region removal.

sde

String used to specify parameters and algorithms for 3D simulations using Sentaurus Structure Editor. By default, `mask` and `thickness` are translated into appropriate Sentaurus Structure Editor commands. If an algorithm is specified using `sde`, it overwrites the algorithm used by default for isotropic or anisotropic etching, for example:

```
photo thickness= 2<um> mask= mask1 sde= {"algorithm" "lopx"}
```

thickness

Specifies the thickness of the photoresist. Default value and unit: 2.0 μm .

Description

The mask must have been defined using a `mask` command. If the photoresist must be deposited inside of the mask, the `negative` argument must be defined in the `mask` command.

Examples

Create a resist layer of 1 μm thickness. The resist layer material will appear in open areas of the mask `mask1`, that is, it will be the negative of `mask1`:

```
photo thickness= 1<um> mask= mask1
```

Create a resist layer of 2 μm thickness. The resist layer will have the same polarity as `mask2`:

```
photo mask= mask2
```

See Also

[mask on page 493](#)

plot.1d

Plots a 1D cross section.

Syntax

```
plot.1d  
  [boundary] [clear] [close] [color=<c>] [fix.ratio] [label=<c>]  
  [max= <list>] [min= <list>]  
  [name=<c>] [rescale] [symb=<c>] [title=<c>]  
  [x=<n>] [<m>|<cm>|<um>|<nm>]  
  [y=<n>] [<m>|<cm>|<um>|<nm>]  
  [z=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

boundary

Specifies that any material boundaries that are crossed must be drawn in as vertical lines on the plot. Default: false.

clear

Specifies whether the graphics screen must be cleared before the graph is drawn. Default: true (the screen is cleared).

close

Closes the plot window.

color

Specifies the line color for the plot. It can be any color supported by X11 hardware and named in the color database.

fix.ratio

Specifies the x-, y-axis ratio to be fixed. Default: false.

label

Specifies the name of the line in the legend of the plot window. The default is the name of the current dataset.

max

List of numeric values that will be the ends of the x- and y-axis. The first argument is the x-value and the second is the y-value. A single value is always interpreted as the x-value. The default is the maximum extent of the current structure.

min

List of numeric values that will be the ends of the x- and y-axis. The first argument is the x-value and the second is the y-value. A single value is always interpreted as the x-value. The default is the minimum extent of the current structure.

name

Name of a data field. This allows plots to be created without using the `select` command. Default: `Z_Plot_Var`.

rescale

Rescales the plot to fit the entire simulation domain.

symp

Specifies a symbol type to be drawn on the cross-sectional line. Each point is drawn with the specified symbol. It defaults to no symbol. Whatever character is entered is placed at each data point on the plot.

title

Specifies the title of the plot window.

x, y, z

Specify the constant values of a line along which sectioning is performed. In one dimension, these arguments are not necessary. In two dimensions, only one of x or y can be specified for a given device. Specifying x produces a horizontal slice through the device and y specifies a vertical slice. An easy way to remember is that the cross section is taken at the constant value specified. For a 3D simulation, two of these three values must be specified. Default unit: μm .

Description

This command plots cross sections vertically or horizontally through the device with arguments to provide for initialization of the graphics device and plotting of axes. This command can optionally draw vertical lines whenever a material boundary is crossed.

A: Commands

plot.1d

Examples

Clear the screen, draw a set of axes, and draw the data along a horizontal cross section at $x = 1.0 \mu\text{m}$. Each point is drawn with symbol 1:

```
plot.1d x= 1.0 symb= 1 clear
```

Draw a horizontal cross section at $x = 2.0 \mu\text{m}$ on the previous set of axis. The line is labeled Lateral in the legend:

```
plot.1d x= 2.0 clear label= Lateral
```

See Also

[select on page 609](#)

[slice on page 623](#)

plot.2d

Plots a 2D xy graphic.

Syntax

```
plot.2d
  [boundary] [clear] [close] [col.bound=<c>] [col.grid=<c>]
  [edges] [faces] [fill] [fix.ratio] [gas] [grid] [label.bound]
  [max= <list>] [min= <list>]
  [nodes] [rescale] [title=<c>]
  [vector=<c>]
  [vlength=<n>] [<m>|<cm>|<um>|<nm>]
  [vmax=<n>] [<m>|<cm>|<um>|<nm>]
  [x=<n> | y=<n> | z=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

boundary

Specifies that the device outline and material interfaces must be drawn. Default: false.

clear

Specifies that the graphics screen must be cleared before the graph is drawn. Default: true (the screen is cleared).

close

Closes the plot window.

col.bound

Specifies the color of the boundary. Any valid X11 color can be specified.

col.grid

Specifies the color of the grid. Any valid X11 color can be specified.

edges

Prints the edge indices on the plot. Default: false.

faces

Prints the face indices on the plot. Default: false.

A: Commands

plot.2d

fill

Specifies that the device must be drawn with the proper aspect ratio. If `fill` is false, the device is drawn with the proper aspect ratio. If `fill` is true, the device is expanded to fill the screen. Default: false.

fix.ratio

By default, the x to y ratio is now fixed. This can be switched off using `!fix.ratio`.

gas

Specifies that the grid in the gas must also be plotted. Default: false (no gas grid is shown).

grid

Specifies that the numeric grid on which the problem was solved must be drawn. Default: false.

label.bound

Name of the material in the lower-left corner of the material region.

max

List of numeric values that will be the ends of the x- and y-axis, respectively. The first argument is the x-value and the second is the y-value. A single value is always interpreted as the x-value. The default is the maximum extent of the current structure. Default unit: μm .

min

List of numeric values that will be the ends of the x- and y-axis, respectively. The first argument is the x-value and the second is the y-value. A single value is always interpreted as the x-value. The default is the minimum extent of the current structure. Default unit: μm .

nodes

Prints the node indices on the plot. Default: false.

rescale

Rescales the plot to fit the whole simulation domain.

title

Specifies the title of the plot window.

vector

Name of a vector field. This indicates arrows proportional to the size of the vector and in the direction of the vector at each node. This argument does not work in 3D simulations.

vlength

Scales the length of the vectors so that the maximum vector has length `vlength`. Default value and unit: $0.1 \mu\text{m}$.

vmax

Use this as the maximum velocity instead of searching for it. Default unit: μm .

x, y, z

For 2D simulations, these arguments are unnecessary. In three dimensions, one of these three must be specified to indicate the outline through the structure. Default unit: μm .

Description

Usually, this command is used to look at material boundaries and grids; however, it also can be used to plot a flow field. This command can be executed immediately before a `contour` command to allow isoconcentration lines to be plotted in context with the structure.

To obtain standard color and other settings for the `plot.2d` window, use the following command from the command line:

```
unix:> xrdb -merge ${STROOT}/tcad/${STRELEASE}/lib/score/XFloops
```

Examples

Draw the triangular grid and axis. Each material is plotted in a different color:

```
plot.2d grid
```

Draw the material interfaces with the minimum x- and y-values of $2.0 \mu\text{m}$ and $5.0 \mu\text{m}$:

```
plot.2d boundary min= {2 5}
```

Draw the material interfaces and place symbols at each coordinate in the mesh:

```
plot.2d boundary diamonds
```

Plot the `Velocity` vector field. The maximum arrow drawn will have a length of $0.1 \mu\text{m}$. The plot surface will not be cleared:

```
plot.2d vector= Velocity vlength= 0.1 !clear
```

A: Commands

plot.2d

See Also

[bound on page 373](#)

[Compatibility on page 380](#)

[contour on page 387](#)

[select on page 609](#)

[slice on page 623](#)

plot.xy

Prepares an xy plot to draw on.

Syntax

```
plot.xy  
  [clear]  
  [max= <list>] [min= <list>]  
  [x.axis=<c>] [y.axis=<c>] [x.log] [y.log]
```

Arguments

clear

Clears the existing plot surface. Default: true.

max

List of numeric values that will be the ends of the x- and y-axis, respectively. The first argument is the x-value and the second is the y-value. A single value is always interpreted as the x-value. The default is the maximum extent of the current structure.

min

List of numeric values that will be the ends of the x- and y-axis, respectively. The first argument is the x-value and the second is the y-value. A single value is always interpreted as the x-value. The default is the minimum extent of the current structure.

x.axis, y.axis

Specify the labels for the x-axis and y-axis.

x.log, y.log

Specify whether there is a linear or log axis. If x.log or y.log is selected, the logarithm of the values on the `point.xy` command are taken. The axis also will have log scale form.

Description

This command configures a 2D plot surface for use with the `point.xy` command. This prepares the axis scaling and labels, and controls the log axes.

Using this command and the `point.xy` command could simulate all other commands in this section.

A: Commands

plot.xy

Examples

Prepare a plot area:

```
plot.xy min= {1 0.0} max= {3600 0.75} x.axis= Time y.axis= Thickness
```

See Also

[point.xy on page 557](#)

[select on page 609](#)

point

Creates a point, for example, for a mask polygon.

Syntax

```
point
  [clear]
  [coord= {
    <n> [<m> | <cm> | <um> | <nm>]
    <n> [<m> | <cm> | <um> | <nm>]
    <n> [<m> | <cm> | <um> | <nm>] }]
  [list] [name=<c>]
```

Arguments

clear

Clears the list of all points. If name is specified, it clears only this point information.

coord

Defines the coordinates of the point. For a 3D point, all three coordinates must be specified. If only two are defined, a 2D layout point in the yz plane is assumed. Default unit: μm .

list

Returns the list of currently defined points. If name is given, it prints the information for this point only.

name

Name of the point.

Description

This command defines a point in three dimensions or a 2D point in the yz plane. It can be used to construct polygons to define masks.

Examples

Define a 2D point with the coordinates $y = 0$ and $z = -1.5$:

```
point name= p1 coord= {0 -1.5}
```

A: Commands

point

Print the list of defined points:

```
LogFile [point list]
```

See Also

[mask on page 493](#)

[polygon on page 559](#)

point.xy

Adds a line segment to a plot.

Syntax

```
point.xy  
  x=<n> y=<n>  
  [color=<c>] [move] [name=<c>] [symb=<c>]
```

Arguments

color

Specifies the color for the line. It can be any color supported by X11 hardware and named in the color database.

move

Instead of drawing from the last point, the graphics pen is placed at this point without moving. Using `move` with `symb` draws scatter plots.

name

Name of a line, so that points can be added to the line at a later time. The name can be any valid character string and is used in the plot legend. If the named line does not exist, it is created.

symb

The first character of this string is used to mark the line. If no symbol is specified, none will be used. If a symbol is specified once for a line, it is used for all lines. Default: `x`.

x, y

Specify the values to be added to the plot.

Description

This command adds segments to a specified line on a plot surface using X-windows-based plotting (`plot.1d` or `plot.2d`). The command is used to plot calculated values, data, or direct outputs from the simulation. The values can be added to any named line.

A: Commands

point.xy

Examples

Add x- and y-values to the line named Thickness:

```
point.xy x= 60.0 y= 0.1 name= Thickness
```

See Also

[interface on page 470](#)

[interpolate on page 473](#)

[plot.xy on page 553](#)

[select on page 609](#)

polygon

Creates a polygon, for example, for a mask.

Syntax

```

polygon
  clear |
  list |
  (name=<c>
    [external.sde] |
    [tdr=<c>] [materials= {<mat1> ... matn}] |
    [max= {
      <y>[<m>|<cm>|<um>|<nm>]
      <z>[<m>|<cm>|<um>|<nm>]}}]
    [min= {
      <y>[<m>|<cm>|<um>|<nm>]
      <z>[<m>|<cm>|<um>|<nm>]}}]
    [rectangle] |
    [points= {<point1> <point2> ... <pointn>}] [rectangle] |
    [tdr=<c>] [regions= {<reg1> ... <regn>}] |
    [segments= {
      <y_1>[<m>|<cm>|<um>|<nm>]
      <z_1>[<m>|<cm>|<um>|<nm>]
      <y_2>[<m>|<cm>|<um>|<nm>]
      <z_2>[<m>|<cm>|<um>|<nm>]
      ...
      <y_n>[<m>|<cm>|<um>|<nm>]
      <z_n>[<m>|<cm>|<um>|<nm>]}}]
  )
  [xy]

```

Arguments

clear

Clears the list of all polygons. If name is specified, it clears only the named polygon.

external.sde

Creates a polygon from an external Sentaurus Structure Editor structure (see [Sentaurus Structure Editor Interface on page 277](#)).

list

Returns a list of all polygons. If name is given, it returns the information for this polygon only.

A: Commands

polygon

materials

Specifies a material or list of materials that will be read when using `tdr`.

max

Maximum point for a rectangular box. It must be used with `rectangle` to create a rectangular polygon. The default is the structure bounding box maximum. Default unit: μm .

min

Minimum point for a rectangular box. It must be used with `rectangle` to create a rectangular polygon. The default is the structure bounding box minimum. Default unit: μm .

name

Name of the polygon.

points

Lists the point names used to specify the polygon. A minimum of three must be specified. The points must have been specified using the `point` command. The polygon is closed implicitly by connecting the first and last points.

`points` also can be used with `rectangle` to specify a rectangular polygon. In this case, two points must be given: the minimum and maximum points of the rectangle.

rectangle

Must be specified with `max` and `min` to define a rectangular box. Alternatively, two named points can be given (using `points`) corresponding to the minimum and maximum of the rectangle.

regions

Specifies a region or list of regions to be used when reading the polygon when using `tdr`.

segments

Lists the line segments in the `yz` plane (or the `xy` plane when `xy` is specified) used to specify a polygon in three dimensions. The polygon is closed implicitly by connecting the first and last points. A minimum of three segments must be given. Default unit: μm .

tdr

Name of the file from which to read the polygon. If you use `tdr`, you must specify `xy`. It allows you to use `materials` and `regions` to further specify which polygon to be read from the TDR file.

`xy`

Defines the polygon in the `xy` plane instead of the default `yz` plane. When using `xy`, the segments are defined as $\{x_1\ y_1\ \dots\ x_n\ y_n\}$, and `min` and `max` as `x y`. You must use `xy` when specifying `tdr`. Specifying `xy` typically means that the polygon will be used for insertion rather than for masking.

Description

This command defines a mask or uses the polygon during an insertion. One of the following must be used to create a polygon:

- `points`
- `rectangle`
- `segments`
- `tdr`

If named points are not given explicitly when forming polygons, they are generated automatically during the creation of the polygon.

Examples

Create three identical rectangles using points and coordinates:

```
point name= p1 coord= {0.0 0.0}
point name= p2 coord= {0.0 -0.5}
point name= p3 coord= {0.5 -0.5}
point name= p4 coord= {0.5 0.0}

polygon name= Box1 points= {p1 p2 p3 p4}
polygon name= Box2 points= {p2 p4} rectangle
polygon name= Box3 min= {0.0 -0.5} max= {0.5 0.0} rectangle
```

Define an L-shaped polygon using 1D line segments:

```
polygon name= LShape \
  segments= {0.0 -1.5 0.0 -0.5 0.5 -0.5 0.5 1.5 1.5 1.5 1.5 -1.5}
```

Read the aluminum material structure in the file `points_bnd.tdr` as a polygon called `box`:

```
polygon name= "box" xy tdr= "points_bnd.tdr" materials= "Aluminum"
```

Print the list of polygons that have been defined:

```
LogFile [polygon list]
```

Delete `Box3`:

```
polygon name= Box3 clear
```

A: Commands

polygon

See Also

[Inserting Polygons on page 274](#)

[insert on page 464](#)

[mask on page 493](#)

[point on page 555](#)

polyhedron

Creates and stores 3D polyhedra, for later insertion with the `insert` command or for use with the `profile` command.

Syntax

```
polyhedron
  clear |
  list |
  (
    name=<c>
    (
      brick= {
        <minx> [<m> | <cm> | <um> | <nm>]
        <miny> [<m> | <cm> | <um> | <nm>]
        <minz> [<m> | <cm> | <um> | <nm>]
        <maxx> [<m> | <cm> | <um> | <nm>]
        <maxy> [<m> | <cm> | <um> | <nm>]
        <maxz> [<m> | <cm> | <um> | <nm>] } |
      original.polyhedron=<c> |
      (polygons= <list> |
        min=<n> [<m> | <cm> | <um> | <nm>]
        max=<n> [<m> | <cm> | <um> | <nm>]) |
      (tdr=<c> [materials= <list>] | [regions= <list>])
    )
    [bbox | bbox.cm | bbox.um]
    [external.sde]
    [offset= {<x> <y> <z>}]
    ([rotate.angles= {<psi> <theta> <phi>}] [rotate.center= {<x> <y> <z>}])
    ([scale=<n>] [scale.center= {<x> <y> <z>}])
    ([stretch] [direction= x | -x | y | -y | z | -z] [length=<n>]
    [location=<n>])
  )
```

Arguments

`bbox`, `bbox.cm`, `bbox.um`

If `bbox` is specified, the `polyhedron` command returns the maximum extents of the named polyhedron in two points. If `bbox.cm` is specified, the command returns the maximum extents of the polyhedron in centimeter. If `bbox.um` is specified, the command returns the maximum extents of the polyhedron in micrometer.

A: Commands

polyhedron

brick

Creates a rectangular prism, given its two corners as `<minx>` `<miny>` `<minz>` and `<maxx>` `<maxy>` `<maxz>`. Default unit: μm .

clear

Removes all the previously defined polyhedra from memory.

external.sde

Creates a polyhedron from an external Sentaurus Structure Editor structure (see [Sentaurus Structure Editor Interface on page 277](#)).

list

Lists the currently defined polyhedra.

materials

It is used only with `tdr` and lists the materials to be included in the file. In addition to explicit material names, `bulk.materials` is available to specify all nongas materials.

max

Maximum x-coordinate for extrusion (see `polygons`). Default unit: μm .

min

Minimum x-coordinate for extrusion (see `polygons`). Default unit: μm .

name

Name of the polyhedron to be created.

offset

Specifies the offset to be applied before saving the named polyhedron.

original.polyhedron

Creates a new polyhedron from the specified original polyhedron before applying the transformation.

polygons

This argument can be used in two contexts:

- When specifying a list of polygons that form a polyhedron, it builds such a polyhedron.

- When specifying one planar axis-oriented polyhedron, it extrudes (using `min` and `max`) that polygon in the x-direction to form a polyhedron.

`regions`

It is used only with `tdr` and lists which regions of the TDR boundary file are included.

`rotate.angles`, `rotate.center`

By default, the location of the rotation center is the geometric center of the polyhedron, which is $\{(x_{min}+x_{max})/2 \ (y_{min}+y_{max})/2.0 \ (z_{min}+z_{max})/2.0\}$. The minimum and maximum coordinates of the polyhedron can be found using the `bbox` option. You can redefine the rotation center using the `rotate.center` argument.

The rotation uses the so-called intrinsic rotation of the Tait–Bryan angles, where `<psi>`, `<theta>`, and `<phi>` are angles rotated around the x-, y-, and z"-axis, respectively. For more information, go to https://en.wikipedia.org/wiki/Euler_angles.

`scale`, `scale.center`

If `scale` is specified, the polyhedron is enlarged or reduced proportionally. By default, the scale center is the geometric center of the polyhedron, but you can redefine the scale center using the `scale.center` argument.

`stretch`, `direction`, `length`, `location`

If the `stretch` option is specified, the polyhedron is stretched a specified length, along the specified direction, at the specified location. By default, `direction=x`, `length=0`, and `location=0`.

`tdr`

Name of the TDR boundary file from which to read all the polyhedra.

Description

This command creates a polyhedron and stores it under the name specified. Different mechanisms can be used to create the polyhedron. It can be read from an existing polyhedron or from a TDR boundary file; it can be defined as a brick or defined from the beginning using polygonal faces; it can be created as an extruded polygon. In addition, several transformations can be applied to the polyhedron, such as offset, rotate, scale, and stretch. When a polyhedron is defined, it can be used to perform polyhedron insertion using the `insert` command, or it can be used as a geometric parameter in the `polyhedron` argument of the `profile` command.

NOTE If a polyhedron is created from a TDR file (`tdr=<c>`) and is intended for use in the `profile` command, you must specify either the `materials` or `regions` argument. If no materials or regions are specified, `materials=bulk.materials` is required.

A: Commands

polyhedron

NOTE The final result depends on the order of the transformations, if any. If `offset`, `rotate`, `scale`, and `stretch` are all specified in the same command, Sentaurus Interconnect executes the transformations in the following sequence starting with `offset`, then `rotate`, `scale`, and finally `stretch`.

Examples

Load the polyhedra containing silicon and gas from the boundary file `sphere_bnd.tdr` with the name `sphere`:

```
polyhedron name= sphere tdr= sphere_bnd.tdr materials= {Silicon Gas}
```

Create a polyhedron named `prism` and extrude from $x = -6 \mu\text{m}$ to $x = 2 \mu\text{m}$ an already existing polygon called `triangle`:

```
polyhedron name= prism polygons= {triangle} min= -6 max= 2
```

Using the polygons `face1`, `face2`, `face3`, and `face4` (they must be correctly defined), build a polyhedron called `tetrahedron`:

```
polyhedron name= tetrahedron polygons= {face1 face2 face3 face4}
```

Define a rectangular prism (brick shape) called `smallPrism` by using its two corners, that is, `<minx>= -6 μm` , `<miny>= -4 μm` , `<minz>= -2 μm` , `<maxx>= -1 μm` , `<maxy>= 4.5 μm` , `<maxz>= 1 μm` :

```
polyhedron name= smallPrism brick= {-6 -4 -2 -1 4.5 1}
```

Stretch and scale a prism:

```
polyhedron name=new_prism original.polyhedron=prism scale=2 stretch \  
length= 0.5 direction= -x
```

Rotate a prism:

```
polyhedron name= prism polygons= {triangle} min= -6 max= 2 \  
rotate.angles= {30 60 90}
```

See Also

[Inserting Polyhedra on page 274](#)

[insert on page 464](#)

[profile on page 583](#)

power_ramp

Defines a power or power-density profile for use with the `supply` command.

Syntax

```
power_ramp
  (clear | list | name=<c>)
  [current.time=<n>] [<hr>|<min>|<s>]
  ([delNP=<n>[<W>] | delNPD=<n>[<W/cm3>]] |
  [delP=<n>[<W>] | delPD=<n>[<W/cm3>]])
  [density.ramprate=<n>[<W/cm3/s>] | ramprate=<n>[<W/s>]]
  [hold] [last]
  [p.density.final=<n>[<W/cm3>] | p.final=<n>[<W>]]
  [power=<n>[<W>] | power.density=<n>[<W/cm3>]]
  [reset.init.time]
  [time=<n>] [<hr>|<min>|<s>]
```

Arguments

`clear`

Deletes the global list of power or power-density ramps. When defining profiles, this operation unites the new definition with any prior profiles of the same name.

`current.time`

Returns the value of the ramp for the specified time. Default unit: minute.

`delNP, delNPD`

Specify either:

- `delNP` to define the maximum power step during a power ramp-down.
- `delNPD` to define the maximum power-density step during a power-density ramp-down.

Default unit: W for `delNP` and Wcm^{-3} for `delNPD`.

`delP, delPD`

Specify either:

- `delP` to define the maximum power step during a power ramp-up.
- `delPD` to define the maximum power-density step during a power-density ramp-up.

Default unit: W for `delP` and Wcm^{-3} for `delPD`.

A: Commands

power_ramp

density.ramprate, ramprate

Specify either:

- `density.ramprate` to set the power-density change during the simulation. Default value and unit: $0 \text{ W cm}^{-3} \text{ s}^{-1}$.
- `ramprate` to set the power change during the simulation. Default value and unit: 0 W s^{-1} .

hold

During this segment, allows the `solve` command the opportunity to specify the time of the segment.

last

Defines the final component of the power or power-density profile. After this final component, no more additions to the ramp are permitted.

list

Generates a list of power or power-density profiles. It returns a Tcl list and can be operated on as such. The default action for commands is to print the return. If no handling is required, this option prints a list of names of defined power or power-density profiles. If name is specified, only the `power_ramp` command is listed along with details about the ramps.

name

Specifies the name of the parameter used to identify the power or power-density ramp. This name is specified in a `supply` command.

p.density.final, p.final

Specify either:

- `p.density.final` to set the final power density for a power-density ramp-up or ramp-down.
- `p.final` to set the final power for a power ramp-up or ramp-down.

Either argument is used if `ramprate` is not given. The ramp time is calculated automatically.

power, power.density

Specify either:

- `power` to set the simulation power rate.
- `power.density` to set the simulation power-density rate.

`reset.init.time`

Starts each power or power-density step with the same initial time step.

`time`

Specifies the simulation time. Default unit: minute.

Description

This command specifies multiple power or power-density ramps and holds. It can be used to construct a complex power or power-density sequence to be simulated with the `solve` command.

Examples

Define the power profile named `pr1` with a power rate of 0.5 W s^{-1} :

```
power_ramp name= pr1 power= 0.1 ramprate= 0.5<W/s> time= 1<s>
```

Define the power-density profile named `pdr1` with a power-density rate of $0.5 \text{ W cm}^{-3} \text{ s}^{-1}$:

```
power_ramp name= pdr1 power.density= 0.2<W/cm3> ramprate= 0.5<W/cm3/s> \  
time= 1<s>
```

See Also

[supply on page 647](#)

print.1d

Prints values along a 1D cross section.

Syntax

```
print.1d
  [gas] [interfaces] [<material>] [name=<c>] [region=<c>]
  [syntax.check.value=<c>]
  [x=<n>] [<m>|<cm>|<um>|<nm>]
  [y=<n>] [<m>|<cm>|<um>|<nm>]
  [z=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

gas

By default, gas values are not reported. This argument allows the gas mesh to be included in the extracted data.

interfaces

Prints interface data from the field specified by name. Values from all interfaces are displayed on the screen and are organized by interface. For each point on the interface, a set of numbers is displayed as follows:

- {x value} in one dimension
- {x y value} in two dimensions
- {x y z value} in three dimensions

where x, y, and z are the coordinates of the interface point, and value is the value of the specified field.

<material>

Name of the material for which the data fields are printed.

name

Name of a data field. This allows printing without using the `select` command. The default is to use the field specified in the most recent `select` command.

region

Name of the region for which the data fields are printed.

`syntax.check.value`

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

`x, y, z`

Specify the constant values of a line along which sectioning is performed. In one dimension, these parameters are not necessary. In two dimensions, only one of `x` or `y` can be specified for a given device. Specifying `x` produces a horizontal slice through the device and `y` specifies a vertical slice. An easy way to remember this is that the cross section is taken at the constant value specified. For a 3D simulation, two of these three arguments must be specified. Default unit: μm .

Description

This command is particularly useful for creating input for another xy plot. A Tcl list is returned for all values. This allows subsequent processing (for example, integration) of the resulting profile.

Examples

Print the selected value at `x` equal to $1.0\ \mu\text{m}$:

```
print.1d x= 1.0
```

Print the data field named `Arsenic` along a vertical line at a lateral position of $1.0\ \mu\text{m}$:

```
print.1d y= 1.0 name= Arsenic
```

See Also

[plot.1d on page 546](#)
[select on page 609](#)
[System on page 651](#)

A: Commands

print.data

print.data

Writes data in x-, y-, and z-format.

Syntax

```
print.data
  [name=<c>] [NODE | EDGE] [outfile=<c>]
  [xlo=<n>] [<m> | <cm> | <um> | <nm>] [xhi=<n>] [<m> | <cm> | <um> | <nm>]
  [ylo=<n>] [<m> | <cm> | <um> | <nm>] [yhi=<n>] [<m> | <cm> | <um> | <nm>]
  [zlo=<n>] [<m> | <cm> | <um> | <nm>] [zhi=<n>] [<m> | <cm> | <um> | <nm>]
```

Arguments

name

Name of a data field. This allows printing without using the `select` command. Default: `Z_Plot_Var`.

NODE, EDGE

Specifies either a node-based field or an edge-based field. Default: `NODE`.

outfile

Name of output file. The file is opened for writing, and any previous content is destroyed.

xlo, ylo, zlo, xhi, yhi, zhi

Specify a 3D bounding box. Only data within these limits is printed. Default value and unit: `0 μm`.

Description

The file format is the x-position, y-position, and z-position. This command is used primarily to write a data field for use with more sophisticated 3D plotting tools.

Examples

Print the data field named `Boron`:

```
print.data outfile= foo name= Boron
```

See Also

[select on page 609](#)

printCapacitanceMatrix

Prints the capacitance matrix for the listed contacts.

Syntax

```
printCapacitanceMatrix  
  (capacitance.matrix [<c1> <c2>] | charge.matrix | spice.matrix |  
   spice.total [<c1>])  
  [<filename>.csv] [<filename>.spi] [simple.print]
```

Arguments

capacitance.matrix [<c1> <c2>]

Specifies that only capacitance matrix must be displayed and returned. If the contact names <c1> and <c2> are given, it returns the capacitance value between <c1> and <c2>.

charge.matrix

Specifies that only the charge matrix must be displayed and returned.

<filename>.csv

Specifies the name of the comma-separated value (CSV) file to which information is written. The file name must include the .csv extension.

<filename>.spi

Specifies the name of the SPICE-formatted file to which information is written. The file name must include the .spi extension.

simple.print

Displays and returns the results without separators.

spice.matrix

Specifies that only the SPICE matrix must be displayed and returned.

spice.total [<c1>]

Specifies that only the total SPICE values must be displayed and returned. If the contact name <c1> is specified, it returns the SPICE value at <c1>.

A: Commands

printCapacitanceMatrix

Description

This command displays and returns the capacitance matrix along with the charge and SPICE model capacitances. It must be used after capacitance analysis is finished.

Examples

Display and return the capacitance matrix along with the charge and SPICE model capacitances:

```
printCapacitanceMatrix
```

Display the capacitance matrix and return an array of `capacitance(contact, contact)` values to `userArray`:

```
array set userArray [printCapacitanceMatrix capacitance.matrix]
```

Display and return the SPICE value for `topContact`:

```
printCapacitanceMatrix spice.total topContact
```

Display and return the values without separators:

```
printCapacitanceMatrix simple.print
```

Display and write the capacitance matrix in CSV format to the file `cMatrix.csv`:

```
printCapacitanceMatrix capacitance.matrix cMatrix.csv
```

See Also

[Electrostatic Analysis: Capacitance Matrix on page 685](#)

printRC

Prints the resistance or capacitance netlist in distributed RC simulations.

Syntax

```
printRC
  [exclude.C= <list>]
  [<filename>.csv] [<filename>.spi]
  [model1 | model2 | model3 | model4]
```

Arguments

`exclude.C`

Specifies a list of materials–regions or material–region pairs whose capacitance should be excluded.

`<filename>.csv`

Specifies the name of the CSV file to which information is written. The file name must include the `.csv` extension.

`<filename>.spi`

Specifies the name of the SPICE-formatted file to which information is written. The file name must include the `.spi` extension.

`model1, model2, model3, model4`

Selects a model to link resistance and capacitance netlists. Default: `model1`.

Description

This command displays and returns the linked RC netlist in a distributed RC simulation. It must be used after the distributed RC analysis is finished.

Examples

Display the linked RC netlist:

```
printRC
```

Display and write the linked RC netlist in SPICE format to the file `frc.spi`:

```
printRC frc.spi
```

A: Commands

printRC

Display the linked RC netlist excluding capacitance between Cu1 and Cu3, between Region_1 and Region_3, and between Cu2 and any other regions:

```
printRC exclude.C= {"Cu1 Cu3" "Region_1 Region_3" "Cu2"}
```

Display the linked RC netlist with model2:

```
printRC model2
```

See Also

[Distributed RC Extraction on page 688](#)

printResistanceMatrix

Prints the resistance matrix for the listed contacts.

Syntax

```
printResistanceMatrix  
  (current.matrix [<c1> <c2>] | resistance.matrix | spice.matrix |  
   spice.total [<c1>])  
  [<filename>.csv] [<filename>.spi] [simple.print]
```

Arguments

current.matrix [<c1> <c2>]

Specifies that only the current matrix must be displayed and returned. If the contact names <c1> and <c2> are given, it returns the current value between <c1> and <c2>.

<filename>.csv

Specifies the name of the CSV file to which information is written. The file name must include the .csv extension.

<filename>.spi

Specifies the name of the SPICE-formatted file to which information is written. The file name must include the .spi extension.

resistance.matrix

Specifies that only the resistance matrix must be displayed and returned.

simple.print

Displays and returns the results without separators.

spice.matrix

Specifies that only the SPICE matrix must be displayed and returned.

spice.total [<c1>]

Specifies that only the total SPICE values must be displayed and returned. If the contact name <c1> is specified, it returns the SPICE value at <c1>.

A: Commands

printResistanceMatrix

Description

This command displays and returns the resistance matrix along with the conductance and SPICE model resistances. It must be used after resistance analysis is finished.

Examples

Display and return the resistance matrix along with the conductance and SPICE model resistances:

```
printResistanceMatrix
```

Display the resistance matrix and return an array of `resistance(contact,contact)` values to `userArray`:

```
array set userArray [printResistanceMatrix resistance.matrix]
```

Display and return the SPICE value for `topContact`:

```
printResistanceMatrix spice.total topContact
```

Display and return the values without separators:

```
printResistanceMatrix simple.print
```

Display and write the current matrix in CSV format to the file `rMatrix.csv`:

```
printResistanceMatrix current.matrix rMatrix.csv
```

See Also

[Electrical Current Analysis: Resistance Matrix on page 683](#)

printThermalRC

Computes and prints the parameters of thermal RC networks for the listed contacts, and prints the thermal impedances to files for the listed contacts.

Syntax

```
printThermalRC  
  [{<c1> <c2> <c3>}] [<c>.csv] [<c>.plx] [<c>.spi] [simple.print]
```

Arguments

{<c1> <c2> <c3>}

Specifies a list of contacts for which the parameters of the thermal RC network will be extracted and printed to the screen or (as well as thermal impedance versus time curves) may be saved to files. If not specified, all of the available contacts (that is, the excitation contact and all the floating contacts specified in the corresponding `mode` command) are used. Default: false.

<c>.csv

Specifies the name of the CSV file to which the extracted thermal RC network parameters are written. The file name must include the `.csv` extension.

<c>.plx

Specifies the name of the PLX-formatted file to which thermal impedance versus time (in seconds) curves and thermal RC network model curves are written.

<c>.spi

Specifies the name of the SPICE-formatted file to which the extracted thermal RC network parameters are written. The file name must include the `.spi` extension.

simple.print

Specify this option to display the results on screen and put them in the log file without separators. When this option is not specified, the default is to display the results on screen and to put the results in the log file with separators.

A: Commands
printThermalRC

Description

This command computes and prints the extracted parameters of thermal RC networks for the listed contacts. In addition, it prints the thermal impedance versus time (in seconds) curves to a specified .plx file for the listed contacts.

NOTE This command must be executed in the `thermal.RC.network` mode after the corresponding transient thermal analysis is finished.

Examples

Extract parameters of thermal RC networks for all available contacts, and print them to the screen and to the log file:

```
printThermalRC
```

See Also

[Thermal Analysis: Thermal RC Network on page 693](#)

printThermalResistanceMatrix

Prints the thermal resistance matrix for the listed contacts.

Syntax

```
printThermalResistanceMatrix  
  (heat.matrix [<c1> <c2>] | resistance.matrix | spice.matrix |  
   spice.total [<c1>])  
  [<filename>.csv] [<filename>.spi] [simple.print]
```

Arguments

<filename>.csv

Specifies the name of the CSV file to which information is written. The file name must include the .csv extension.

<filename>.spi

Specifies the name of the SPICE-formatted file to which information is written. The file name must include the .spi extension.

heat.matrix [<c1> <c2>]

Specifies that only the heat matrix must be displayed and returned. If the contact names <c1> and <c2> are given, it returns the value between <c1> and <c2>.

resistance.matrix

Specifies that only the resistance matrix must be displayed and returned.

simple.print

Displays and returns the results without separators.

spice.matrix

Specifies that only the SPICE matrix must be displayed and returned.

spice.total [<c1>]

Specifies that only the total SPICE values must be displayed and returned. If the contact name <c1> is specified, it returns the SPICE value at <c1>.

A: Commands

printThermalResistanceMatrix

Description

This command displays and returns the thermal resistance matrix along with the thermal conductance and SPICE model thermal resistances. It must be used after thermal resistance analysis is finished.

Examples

Display and return the thermal resistance matrix along with the thermal conductance and SPICE model thermal resistances:

```
printThermalResistanceMatrix
```

Display the thermal resistance matrix and return an array of thermalresistance(contact, contact) values to userArray:

```
array set userArray [printThermalResistanceMatrix resistance.matrix]
```

Display and return the SPICE value for topContact:

```
printThermalResistanceMatrix spice.total topContact
```

Display and return the values without separators:

```
printThermalResistanceMatrix simple.print
```

Display and write the heat matrix in CSV format to the file tMatrix.csv:

```
printThermalResistanceMatrix heat.matrix tMatrix.csv
```

See Also

[Thermal Analysis: Thermal Resistance Matrix on page 691](#)

profile

Reads a data file and constructs a data field.

Syntax

```
profile
  name=<c>
  [Adaptive] [concentration=<n>] [delay.doping.3d] [gaussian.blur]
  [infile=<c>]
  [lateral.function= {xlo= {<n> ...} xhi= {<n> ...} sigma= {<n> ...}
  LDF.reshaping= {<c> ...}}]
  [logarithmic | linear]
  [mask=<n>] [mask.discretization.size=<n>]
  [<material>] [material.specific]
  [max= {<n> <n> <n>}] [min= {<n> <n> <n>}]
  [offset= {<n> <n> <n>}] [<m>|<cm>|<um>|<nm>]
  [polyhedron=<c>]
  [region=<c>] [replace | sum]
  [x.sigma=<n>] [xcoord=<n>] [xscale=<n>]
  [y.sigma=<n>] [ymin=<n>] [yscale=<n>] [z.sigma=<n>]
```

Arguments

Adaptive

Specifies whether this profile step is performed with adaptive remeshing. This functionality is not supported when `infile` is a TDR file, or when the `polyhedron` argument is specified. Default: [pdbGet Grid Adaptive].

concentration

Specifies the concentration of the field at the specified `xcoord` or inside the specified `polyhedron`. When it is paired with the `xcoord` argument, the `concentration` argument is a list in which the number of list elements must match that of `xcoord`. When it is specified with the `polyhedron` argument, the `concentration` argument is a simple numeric value.

delay.doping.3d

Specifies whether the placement of the dopant concentration is delayed until the next remeshing step in the 3D simulation. This may be useful if there are several consecutive `profile` commands, which can be put together and performed in one adaptive remeshing step.

This functionality is not supported when `infile` is a TDR file, or when the `polyhedron` argument is specified. Default: false.

A: Commands

profile

gaussian.blur

Specifies whether the 2D or 3D doping distribution is generated by combining the 1D depth profile (specified by `infile`) with 2D mask information (specified by `mask`). The lateral distribution is computed by convolving the mask information with the Gaussian function.

infile

Name of the file to be read. If it is an ASCII data file, the file must be in a two-column format with depth (in μm) in column 1 and the variable in column 2. It also will read the output of the `print.1d` command, which includes Tcl braces for list processing and the material name. If it is a TDR file, it must be in the same dimension as the simulated device structure and requires `.tdr` as the file extension.

lateral.function

Specifies the lateral straggling (`sigma`'s) and the lateral distribution function (LDF) for the profile reshaping (`LDF.reshaping`) as a function of depth in a piecewise step manner.

The `xlo`, `xhi`, and `sigma` arguments specify that, between `xlo` and `xhi`, the lateral straggling is `sigma`. These three lists must have exactly the same number of items and must cover the entire depth of the device. For simplicity, however, if `sigma` is a constant (that is, a single value), `xlo` and `xhi` can be omitted.

The `LDF.reshaping` argument specifies the file names of the LDF correction factors in `.plx` file format. The first item of the list specifies the file name for the first pair of `xlo` and `xhi`, and the second item specifies the file name for the second pair of `xlo` and `xhi`, and so on (the file names can be repeated if they are the same). For simplicity, however, if the correction factor is 1 (that is, no correction), the file name can be omitted.

logarithmic, linear

Interpolates data using either logarithmic or linear interpolation. The default is `logarithmic`, which is usually more accurate for concentration profiles.

mask

Specifies the name of the mask used for Gaussian blurring.

mask.discretization.size

During computation of the Gaussian convolution, a 2D mask is divided into small slices in the z -direction. This argument specifies the maximum discretization size for each slice. Default value and unit: $1\text{e-}6$ cm.

<material>

Name of the material to which the field profile is applied. See [Specifying Materials on page 18](#).

material.specific

By default, when loading a TDR file, interpolation is material specific, so that data is interpolated from the file only if the material in the existing structure matches. Use `!material.specific` to allow interpolation regardless of material.

max

List of numeric values defining the x-, y-, and z- coordinates of the lower-right front corner of the 1D, 2D, or 3D rectangular box into which the profile is imported. For 1D, 2D, and 3D structures, a list of one, two, or three numbers is required, respectively. The possible maximum number is used for missing numbers. Default unit: μm .

min

List of numeric values defining the x-, y-, and z- coordinates of the upper-left back corner of the 1D, 2D, or 3D rectangular box into which the profile is imported. For 1D, 2D, and 3D structures, a list of one, two, or three numbers is required, respectively. The possible minimum number is used for missing numbers. Default unit: μm .

name

Name of the data field. This argument allows for the creation of arbitrary fields, for example, a field called `Measured`.

offset

List of numeric values that specify the offsets in the x-, y-, and z-direction, respectively. The missing values are treated as 0. These values will be subtracted from the x-, y-, and z-coordinate, respectively, when creating the data field from the imported field. This argument allows a profile to be shifted. Default value and unit: $0 \mu\text{m}$.

polyhedron

Specifies a name as defined by the following `polyhedron` command (see [polyhedron on page 563](#)):

```
polyhedron name=<c> (brick= {...} | original.polyhedron=<c> |
polygons= {...} | tdr=<c>)
```

If a polyhedron is created by specifying `brick`, you can use different values of standard deviations (`x.sigma`, `y.sigma`, and `z.sigma`) in each of the three principal axes. However, if a polyhedron is created by specified either `tdr` or `polygons`, only a single roll-off function (`x.sigma`) is allowed. The roll-off is computed using the nearest distance to the polyhedron.

region

Name of the region to which the profile is applied.

A: Commands

profile

replace, sum

These options specify how a newly added profile concentration is used:

- Specify `replace` to replace the existing dataset with the new concentration of the same name inside the polyhedron or in the rectangular box (specified by the `min` and `max` arguments). Outside the polyhedron or the rectangular box, to ensure a smooth transition, the roll-off profile replaces the existing concentration only if the concentration of the roll-off profile is greater than the existing concentration.
- Specify `sum` to add the new concentration to the matching dataset. This is the default.

x.sigma

Standard deviation of erfc falloff from a rectangular box in the x-direction. It must be specified if a rectangular box or polyhedron is specified. Default unit: μm .

xcoord

Coordinate in the x-direction where the concentration will be defined. Default unit: μm .

xscale

The `profile` command assumes the x-dimension is in micrometers. This argument allows you to scale the depth dimension if necessary. For example, if the depth is in ångströms, 1×10^{-4} must be specified. Default: 1.0.

y.sigma

Standard deviation of erfc falloff from a rectangular box in the y-direction. If it is not specified, it takes the value of `x.sigma`. Default unit: μm .

ymin

Minimum-acceptable value of the data field. Values less than `ymin` in the data field are set to `ymin`. This is useful for data that may approach zero when using logarithmic interpolation.

yscale

The scaling factor for the y-dimension (usually, the concentration) of the 1D profile. Default: 1.0.

z.sigma

Standard deviation of erfc falloff from a rectangular box in the z-direction. If it is not specified, it takes the value of `y.sigma`. Default unit: μm .

Description

This command provides different ways of introducing dopant concentration into a device:

- It reads data fields from an ASCII data file or a TDR file, and replaces or adds them to the structure. It allows you to read a doping profile from a SIMS measurement. In this case, if the simulated structure is 2D or 3D, the data field is created uniformly in the lateral direction. This command also allows you to read a field from a TDR file with the same dimension as the simulated structure. You also can limit the extent of the imported profile within a rectangular box by specifying `min` or `max`, or both. Outside this box, the profile falls off with a complementary error function (`erfc`) with standard deviations given by `x.sigma`, `y.sigma`, and `z.sigma` in the x-, y-, and z-direction, respectively.
- It places a uniform dopant concentration inside a polyhedron without triggering remeshing. The dopant and concentration are specified by the name and `concentration` arguments, respectively. The polyhedron must be already defined using the `polyhedron` command:

```
polyhedron name=<c> (brick= {...} | original.polyhedron=<c> |
  polygons= {...} | tdr=<c>)
```

If a polyhedron is created by specifying `brick`, you can use different values of standard deviations (`x.sigma`, `y.sigma`, and `z.sigma`) in each of the three principal axes. However, if a polyhedron is created by specifying either `tdr` or `polygons`, only a single roll-off function (`x.sigma`) is allowed. The roll-off is computed using the nearest distance to the polyhedron.

- It computes a 2D or 3D dopant distribution by combining a 1D depth profile (specified by either `infile` or both the `xcoord` and `concentration` arguments) and 2D mask information. The 1D depth profile is obtained elsewhere (for example, a 1D process simulation or from a SIMS profile) and is stored in a file specified by `infile` in ASCII format. The 2D lateral dopant profile is generated by convolving the mask with a Gaussian function in two dimensions¹, essentially smoothing the sharp edges of the mask. Depth-dependent lateral straggling and profile reshaping are specified as lists inside the `lateral.function` argument. The doping profile placement can be adaptive or nonadaptive. In three dimensions and with `Adaptive` specified, you can delay the profile placement until the next remeshing step by specifying the `delay.doping.3d` option.

Examples

Read a temperature profile, and scale the depth by 1×10^{-4} to convert ångström to micrometer:

```
profile name= Temperature infile= temp.pro xscale= 1.0e-4
```

See Also

[print.1d on page 570](#)

1. For more information, go to https://en.wikipedia.org/wiki/Gaussian_blur.

RangeRefineboxes

Creates a set of refinement boxes based on a mask, and a set of range and extent parameters. All boxes share a set of global refinement settings, but each box can have additional local refinement settings.

Syntax

```
RangeRefineboxes
boxes= {
    drange=<drange1> [<box-specific_arguments>]
    [drange=<drange2> [<box-specific_arguments>]]
    ...
}
mask=<c>
name=<c>
range=<c>
[<other_arguments>]
```

Arguments

<box-specific_arguments>

Any argument of the `refinebox` command. These arguments are applied only to the individual refinement box. They can overwrite global parameters defined in the `<other_arguments>` section.

boxes

A Tcl list containing a set of Tcl lists. Each Tcl list contains specific parameter settings for one individual refinement box.

drange

Specifies the primary extent of an individual refinement box, which extends from $x_{min} = range - drange$ to $x_{max} = range + drange$.

mask

Specifies the mask under which the refinement is to be applied.

name

Specifies the root name for the set of refinement boxes. Each individual refinement box inherits a name of the form `<c>_<n>`, where `<c>` is the value of the `name` argument and `<n>` is a counter.

<other_arguments>

Any valid argument of the `refinebox` command.

`range`

Specifies the center of the primary extent for all refinement boxes in the set.

Description

The refinement is applied to the area under the specified mask. The lateral extent is controlled by the same arguments as in the `refinebox` command, for example, `mask.edge.refine.extent` and `mask.edge.mns`.

The primary extent is defined by `range` and `drange`. The `range` argument is common to all refinement boxes. The `drange` argument can be set for each related refinement box separately.

As optional global default arguments, any argument from the `refinebox` command is allowed. These arguments are applied to all refinement boxes of the set. For each individual refinement box, an additional `refinebox` command can be set, which can overwrite globally defined arguments or add new arguments. There are no limits on how many refinement boxes can be in the set. The individual refinement boxes inherit the root name (specified by the `name` argument) with a numeric suffix counter.

NOTE The `RangeRefineboxes` command makes one call to the `refinebox` command per individual refinement box.

NOTE You can use the `DeleteRefinementboxes` command to remove the entire set of refinement boxes created with the `RangeRefineboxes` command.

Examples

Create a set of refinement boxes:

```
mask name= M1_p segments= {6 10}
array set moments "model dualpearson rp 0.078379 stdev 0.029416 gamma 0.607428
  beta 3.812230 ratio 0.712042 rp2 0.267171 stdev2 0.181446 gamma2 2.819940
  beta2 25.463300"
set range $moments(rp)
set sigma $moments(stdev)
eval RangeRefineboxes name= "RM1" mask= "M1_p" range= $range \
  boxes= \{ \
    {drange= $sigma xrefine= [expr $sigma/4.0] yrefine= [expr $sigma/4.0] \
      extend= 0} \
    {drange= [expr 4*$sigma] xrefine= $sigma yrefine= $sigma \
      extend= [expr 2*$sigma]} \
```

A: Commands

RangeRefineboxes

```
{drange= [expr 2*$sigma] mask.edge.mns= [expr $sigma/8.0] \  
  mask.edge.refine.extent= [expr 2*$sigma] mask.edge.ngr= 1} } \  
extend= 1.0 xrefine= 0.5 yrefine= 0.5 info= 2
```

See Also

[DeleteRefinementboxes on page 400](#)

[refinebox on page 591](#)

mat.lattice.lkmc

Material used for lattice creation in LKMC.

nucleation.material

Material on which nucleation occurs during nonselective epitaxy in LKMC.

refinebox

Sets the local grid parameters and performs a grid refinement using the MGOALS module.

Syntax

```

refinebox
  [3d] [Adaptive] [clear] [double.side]
  [list]
  [(mask=<c> extrusion.min=<n> extrusion.max=<n>) [extend=<n>]
   ([mask.corner.mns=<n>] [mask.corner.ngr=<n>]
    [mask.corner.refine.extent=<n>] |
    [mask.edge.mns=<n>] [mask.edge.ngr=<n>] [mask.edge.refine.extent=<n>])]
  [<material>] [materials= <list>]
  [max= <list>] [min= <list>]
  [name=<c>]
  [print]
  [regions= <list>]
  [xrefine= <list>] [<m>|<cm>|<um>|<nm>]
  [yrefine= <list>] [<m>|<cm>|<um>|<nm>]
  [zrefine= <list>] [<m>|<cm>|<um>|<nm>]

  [interface.mat.pairs= <list>]
  [interface.materials= <list>]
  [interface.region.pairs= <list>]
  [interface.regions= <list>]
  [max.lateral.size=<n>] [<m>|<cm>|<um>|<nm>]
  [min.normal.size=<n>] [<m>|<cm>|<um>|<nm>]
  [normal.growth.ratio=<n>]
  [offsetting]
  [offsetting.maxlevel=<i>]

  [DelPSC
    [interface.mat.pairs= <list>]
    [interface.materials= <list>]
    [interface.region.pairs= <list>]
    [interface.regions= <list>]
  ]

  [abs.error= {<field1>=<n> <field2>=<n> ...}]
  [def.abs.error=<n>]
  [def.max.asinhdiff=<n>]
  [def.max.difference=<n>]
  [def.max.dose.error=<n>]
  [def.max.gradient=<n>]
  [def.max.logdiff=<n>]
  [def.rel.error=<n>]

```

A: Commands

refinebox

```
[max.asinhdiff= {<field1>=<n> <field2>=<n> ...}]
[max.difference= {<field1>=<n> <field2>=<n> ...}]
[max.dose.error= {<field1>=<n> <field2>=<n> ...}]
[max.gradient= {<field1>=<n> <field2>=<n> ...}]
[max.logdiff= {<field1>=<n> <field2>=<n> ...}]
[max.value=<n>] [min.value=<n>]
[refine.add.fields= <list>]
[refine.dir.factor= <list>]
[refine.expr=<c>]
[refine.field.expr= {<field1>=<c> <field2>=<c> ...}]
[refine.fields= <list>]
[refine.max.edge= <list>]
[refine.min.edge= <list>]
[refine.rm.fields= <list>]
[refine.type=<c>]
[rel.error= {<field1>=<n> <field2>=<n> ...}]
[target.length=<n>] [<um>] [target.length.scaling=<n>]
```

Arguments

3d

Specifies the refinement box for only three dimensions, or for only one dimension and two dimensions. The default behavior is to always apply the refinement box. If `3d` is specified, the refinement box only applies to three dimensions. If `!3d` is specified, the refinement box only applies to one dimension and two dimensions.

Adaptive

Specifies an adaptive refinement box. Adaptive refinement boxes are used during all remeshing operations (such as `deposit`, `etch`, `photo`, `transform`) but will not be used during solve unless adaptive meshing is switched on (by using `pdbSet Grid Adaptive 1`).

clear

When used without other arguments, `clear` deletes all previously defined refinement boxes. When used with `name`, only the named refinement box is deleted.

double.side

If `!double.side` is specified with `offsetting`, Sentaurus Mesh interprets `interface.mat.pairs` and `interface.region.pairs` in a nonsymmetric fashion. The default is `double.side`, that is, the specification of a material or region pair x_1/x_2 is interpreted by Sentaurus Mesh as if the parameters were defined symmetrically for both x_1/x_2 and x_2/x_1 .

`extend`

Optional extension when using a mask-driven refinement. This value can be positive (or negative) and extends (shrinks) the refinement isotropically in y and z. The original mask remains unchanged. Default unit: μm .

`extrusion.max`, `extrusion.min`

Maximum and minimum coordinates in the x-axis where the refinement will be applied when using `mask`. Default unit: μm .

`list`

Lists the defined refinement boxes.

`mask`

Uses an existing mask name as an extra constraint to where the refinement will be applied. If the refinement contains another spatial constraint (for example, using `min` and `max`), the final application region is the intersection of the other constraints and the specified extruded mask.

This argument requires specifying the box length in x (lacked by the mask) using `extrusion.min` and `extrusion.max`, and optionally allows the use of `extend`, which allows for the definition of layout (mask)-driven refinements.

`mask.corner.mns`

Specifies the minimum mesh size near the mask corner (actual edge length may be up to two times smaller than this setting). This argument must be used with `mask.corner.refine.extent` to have an effect. Default unit: μm .

`mask.corner.ngr`

Specifies the growth rate of refinement away from the mask corner. This argument must be used with `mask.corner.refine.extent` to have an effect. Default value and unit: $1.0 \mu\text{m}$ (no growth).

`mask.corner.refine.extent`

Specifies the distance from the mask corner over which corner-based refinement occurs. It must be specified to obtain mask corner-based refinement. Default unit: μm .

`mask.edge.mns`

Specifies the minimum mesh size near the mask edge (actual edge length may be up to two times smaller than this setting). This argument must be used with `mask.edge.refine.extent` to have an effect. Default unit: μm .

A: Commands

refinebox

`mask.edge.ngr`

Specifies the growth rate of refinement away from the mask edge. This argument must be used with `mask.edge.refine.extent` to have an effect. Default value and unit: 1.0 μm (no growth).

`mask.edge.refine.extent`

Specifies the distance from the mask edge over which edge-based refinement occurs. It must be specified to obtain mask edge-based refinement. Default unit: μm .

`<material>`

Limits the refinement box to a particular material. By default, the refinement box applies to all materials. See [Specifying Materials on page 18](#).

`materials`

Limits the refinement box to a list of materials. By default, the refinement box applies to all materials. See [Specifying Materials on page 18](#).

`max, min`

Limits the extent of the refinement box. Both arguments take a Tcl list of numbers defining the refinement box extent in the x-, y-, and z-axes. You can specify either one or both `min` and `max` with a Tcl list of one, two, or three numbers for each argument. If one number is specified, it is taken to be the limit in the x-axis. If two numbers are specified, they set limits for the x-axis and y-axis. Similarly, three numbers specify a limit in all three axes. Default unit: μm .

`name`

Name of the refinement box.

`print`

Prints information for all refinement boxes. If `name` is specified, only the named refinement box information is printed.

`regions`

Limits the refinement box to a list of regions. By default, the refinement box applies to all regions.

`xrefine`

List of three numbers defining the element sizes in the x-direction at the top, middle, and bottom of the box. Default unit: μm .

yrefine

List of three numbers defining the element sizes in the y-direction at the left, middle, and right of the box. Default unit: μm .

zrefine

List of three numbers defining the element sizes in the z-direction at the front, middle, and back of the box. Default unit: μm .

Arguments: Interface Refinement Control

interface.mat.pairs

A set of pairs of materials where interface meshing will be switched on (1st and 2nd, 3rd and 4th, and so on).

interface.materials

All interfaces that contain any of the materials listed here are refined using the `min.normal.size` criterion. By default, no interface refinement is applied.

interface.region.pairs

A set of pairs of regions where interface meshing will be switched on (1st and 2nd, 3rd and 4th, and so on). This region-based interface specification is supported only for Sentaurus Mesh offsetting, that is, when `offsetting` also is given.

interface.regions

Used only in conjunction with `offsetting` or `offsetting.maxlevel` to switch on Sentaurus Mesh offsetting or to specify `offsetting.maxlevel` on a regionwise basis for Sentaurus Mesh offsetting.

max.lateral.size

Specifies the maximum lateral spacing at the interface. Default unit: μm .

min.normal.size

Specifies the minimum edge spacing at interfaces for this box. Default unit: μm .

normal.growth.ratio

Specifies the edge-to-edge growth ratio moving away from an interface.

offsetting

When `offsetting` is specified along with `interface.materials`, `interface.mat.pairs`, `interface.regions`, or `interface.region.pairs`, the

A: Commands

refinebox

Sentaurus Mesh offsetting algorithm is used to generate offsetting layers at the given interface.

`offsetting.maxlevel`

Specifies the number of offsetting layers at the interface when Sentaurus Mesh offsetting is used, specified either by material or region using `interface.materials` or `interface.regions`, respectively.

Arguments: Boundary Rediscretization Using DelPSC Algorithm

`DelPSC`

Specify this option to apply the Delaunay refinement for piecewise smooth complex (DelPSC) algorithm to rediscretize curved surfaces on the boundary at the beginning of the Sentaurus Mesh simulation. This is equivalent to the following setting:

```
pdbSet Grid SnMesh Apply.Brep.DelPSC true
```

`interface.mat.pairs`

Restricts the application of the DelPSC algorithm to interfaces specified by pairs of materials (first and second, third and fourth, and so on). For example:

```
interface.mat.pairs= {Silicon Oxide}
```

`interface.materials`

Restricts the application of the DelPSC algorithm to interfaces of the specified materials. For example:

```
interface.materials = {Silicon}
```

`interface.region.pairs`

Restricts the application of the DelPSC algorithm to interfaces specified by pairs of regions (first and second, third and fourth, and so on). For example:

```
interface.region.pairs= {Substrate field_ox}
```

`interface.regions`

Restricts the application of the DelPSC algorithm to interfaces of the specified regions. For example:

```
interface.regions= {Substrate}
```

Arguments: Adaptive Meshing

`abs.error`

Sets a field-dependent value of the minimum significant field value.

`def.abs.error`

Sets the field-independent default value of the minimum significant field value.

`def.max.asinhdiff`

Sets the field-independent default value of the maximum inverse hyperbolic sine (asinh) difference criteria.

`def.max.difference`

Sets the field-independent default value of the maximum absolute difference criteria.

`def.max.dose.error`

Sets the field-independent default value of the maximum local dose error criteria.

`def.max.gradient`

Sets the field-independent default value of the maximum gradient criteria.

`def.max.logdiff`

Sets the field-independent default value of the maximum logarithmic difference criteria.

`def.rel.error`

Sets the field-independent default value of the required relative change of a field across an edge.

`max.asinhdiff`

Sets a field-dependent value of the inverse hyperbolic sine difference criteria.

`max.difference`

Sets a field-dependent value of the maximum absolute difference criteria.

`max.dose.error`

Sets a field-dependent value of the maximum local dose error criteria.

`max.gradient`

Sets a field-dependent value of the maximum gradient criteria.

A: Commands

refinebox

`max.logdiff`

Sets a field-dependent value of the maximum logarithmic difference criteria.

`max.value`

Maximum interval value for interval refinement.

`min.value`

Minimum interval value for interval refinement.

`refine.add.fields`

List of fields to be added to the default list of fields considered for adaptive refinement.

`refine.dir.factor`

Applies adaptive refinement more strongly in one direction than another. A factor of 1 has no effect. A factor less than 1 causes smaller edges in that direction.

For example, `refine.dir.factor= {0.1 1.0}` requests that, for a given adaptive refinement expression value, edges in the x-direction be 10 times smaller than edges in the y-direction.

`refine.expr`

Specifies a refinement expression. It takes any valid Alagator expression that produces a node-based result.

NOTE Earlier releases required the `diff()` operator, but now, the `diff` operator must not be used. Similar results can be obtained for earlier releases by removing the `diff` operator.

`refine.field.expr`

Sets a field-dependent refinement expression.

`refine.fields`

Replaces the default list of fields considered for adaptive refinement. Solution variables and terms can appear in the `refine.fields` list. (For a description of a term, see [term on page 658](#).)

`refine.max.edge`

Sets the direction-dependent maximum edge length.

`refine.min.edge`

Sets the direction-dependent minimum edge length.

`refine.rm.fields`

Removes the specified fields from the default list of fields considered for adaptive meshing.

`refine.type`

Specifies the type of criteria to apply for adaptive refinement. Allowed values are `interval` and `error` (default).

`rel.error`

Sets a field-dependent value of the required relative change of the refined field across an edge.

`target.length`

Target length for interval refinement. Default unit: μm .

`target.length.scaling`

Scaling factor used in the calculation of the effective target length for interval refinement.

Description

This command specifies mesh refinement. The following types of refinement box are available:

- Standard: Independent `xrefine`, `yrefine`, `zrefine` settings.
- Interface: Refinement near one or more interfaces.
- Adaptive: Adaptive refinement on fields.
- Plane: Planar refinement for crystal boundaries.
- Bulk mask: Confine refinement to an extruded boundary defined by a mask.
- Mask edge: Confine refinement to a specified distance from a specified mask.

All refinement boxes can be limited by material or spatially by specifying x-, y-, or z- minimum or maximum limits.

Examples

Define two refinement boxes:

```
refinebox min= {-0.25 0.4} max= {0.4 0.6} xrefine= {0.1 0.06 0.1} \  
  yrefine= {0.1 0.01 0.1} oxide  
  
refinebox min= {0.6 0.6} max= {0.8 0.8} xrefine= {0.1 0.03 0.1} \  
  yrefine= {0.1 0.03 0.1} silicon
```

A: Commands

refinebox

Create an adaptive refinement box that applies maximum dose error criteria to the default list of adaptive species, and effectively switch off relative error criteria, which is on by default:

```
refinebox adaptive def.rel.error= 100 def.max.dose.error= 5e9
```

Form the boundary for the refinement by extending (by 0.2 μm) an existing mask called Mask1 extruded from 0 μm to 0.05 μm :

```
polygon name= pol segments= {- .5 - .5 .5 - .5 .5 .5 0 .5 0 0 - .5 0}  
mask name= Mask1 polygons= {pol}  
refinebox name= "ref1" mask= Mask1 yrefine= {0.05 0.075 0.075} \  
extrusion.min= 0 extrusion.max= 0.05 extend= 0.2
```

See Also

[Mesh Refinement on page 170](#)

[mask on page 493](#)

[mgoals on page 510](#)

region

Creates regions, marks substrates, and changes region materials.

Syntax

```
region
  <material>
  xlo=<c> [ylo=<c>] [zlo=<c>]
  xhi=<c> [yhi=<c>] [zhi=<c>]
  [alt.matername=<c>] [bbox | bbox.cm | bbox.um]
  [change.material] [cropped.bbox] [exact.name]
  [field=<c> concentration=<n>]
  [left | right | back | front | top | bottom]
  [list | list.bulk | list.gas | list.interface]
  [material]
  [max= {<n> <n> <n>} min= {<n> <n> <n>}]
  [name=<c>] [new.name=<c> point= {<n> <n> <n>}]
  [substrate] [syntax.check.value=<c>] [volume] [zero.data]
```

Arguments

alt.matername

Specifies the name of the material that should be written in the TDR file for regions of the newly defined material.

For example, when writing the TDR file, if the newly defined material is called `MySilicon` and `alt.matername=Silicon`, Sentaurus Interconnect uses `Silicon` for those regions of `MySilicon`.

This is useful when transferring the structure to device simulation.

bbox, bbox.cm, bbox.um

If specified, the `region` command returns the maximum extents of the region in two points. If `bbox.cm` is specified, it returns the maximum extents of the region in centimeter. If `bbox.um` is specified, it returns the maximum extents of the region in micrometer.

change.material

Changes the material of an existing region (must be used with `name`). Changing the material of selected regions (to and from gas) can be used to change the structure without remeshing. Meshes of material gas are ignored in most steps.

concentration

Specifies the value of the field directly. Default unit: cm^{-3} .

A: Commands

region

`cropped.bbox`

If specified, returns the cropped bounding box of a region that lies within a user-specified bounding box.

`exact.name`

Usually when changing the material of a region, all ancestors of the named region (if there are any) are converted as well as the named region if it exists (see [Regionwise Parameters and Region Name-Handling on page 24](#)).

If `exact.name` is switched on, only a region whose name exactly matches the name argument will have its material changed. Default: `false`.

`field`

Name of a field to be initialized within this region.

`left | right | back | front | top | bottom`

If specified, returns the region names on either the left, right, back, front, top, or bottom of the structure.

`list, list.bulk, list.gas, list.interface`

Used to obtain a Tcl list of regions:

- `list` returns a list of all regions.
- `list.bulk` returns a list of nongas, noninterface regions (that is, all bulk regions).
- `list.gas` returns a list of gas regions.
- `list.interface` returns a list of interface regions.

`<material>`

Material of the region. See [Specifying Materials on page 18](#).

`material`

If specified, it returns the material name of the named region.

`max, min`

User-specified bounding box.

`name`

Name of the region. The name must not contain an underscore (`_`) or a period (`.`) because these characters have special meaning. The name must be different than an existing material name (see [Regionwise Parameters and Region Name-Handling on page 24](#)).

`new.name, point`

Used together to change the name of a region, where `point` must specify a point (a list of doubles) within a region. The point must not be on or very near a border. The argument `new.name` specifies the new name of the region. The name must be different than an existing material name (see [Regionwise Parameters and Region Name-Handling on page 24](#)).

`substrate`

Tags a named region as the substrate for subsequent analysis. Setting `!substrate` clears the substrate tag. If no region name is specified and `!substrate` is set, all substrate tags are cleared.

`syntax.check.value`

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

`xlo, ylo, zlo, xhi, yhi, zhi`

Specify the bounds of the region. The `<c>` value must be one of the tags created in a preceding `line` command.

`volume`

If specified, the `region` command returns the volume of the named region. The units will be in cm^D , where D is the simulation dimension.

`zero.data`

Usually when the material of a region is changed using `change.material`, all data in that region is set to 0. Setting `!zero.data` leaves the data untouched. The default value for this parameter is taken from `pdbGet Grid default.zero.data`, which allows a global setting for this argument.

Description

The `region` command has different applications:

- At the beginning of a simulation, the initial regions are created with the `region` command in concert with the `line` command and the `init` command. The `line` command defines where mesh lines go.
- The `region` command specifies between which mesh lines the regions are created and what material the regions will be, and whether this region will be a substrate. It is used to

A: Commands

region

change the material of a region at any point in the simulation after the structure has been initialized.

- The `region` command can be used to return a cropped bounding box of a region, specified within a user-specified bounding box, defined by `min` and `max` along with a region name.

Examples

Create a new material `MySilicon`, and then change the material of a region named `bulk` to `MySilicon` without changing the data:

```
mater name= MySilicon new.like= Silicon add
region name= bulk MySilicon change.material !zero.data
```

Change the region `Gate` to `Gas` before setting all fields to zero in `Gate` (`zero.data` defaults to true):

```
region name= Gate Gas change.material
```

Create a 2D silicon region using the statements from the example for the `line` command:

```
region silicon ylo= left yhi= right xlo= surf xhi= back
```

Return a cropped bounding box of the region `bulk` that lies within the specified bounding box defined by `min` and `max`:

```
region name= bulk min= {-5.0 0.0 0.0} max= {5.0 1.0 1.0} cropped.bbox
```

See Also

[integrate on page 467](#)

[line on page 481](#)

resistance

Prints the resistance between two contacts.

Syntax

```
resistance <c> <c>
```

Arguments

<c>

Names of the two contacts.

Description

If electrical analysis is not performed, the `resistance` command returns an error message.

Examples

Display the resistance between electrical contacts C1 and C2:

```
resistance C1 C2
```

See Also

[Electrical Current Analysis: Resistance Matrix on page 683](#)

sde

Dispatches commands to Sentaurus Structure Editor (only available for 3D simulations).

Syntax

```
sde
  {<Sentaurus Structure Editor commands>}
  [Adaptive] [external] [logfile=<c>] [off] [on]
  [polyhedron=<c>] [polyhedron.material=<c>]
  [remesh] [SdeCheck]
```

Arguments

Adaptive

If specified, `Adaptive` switches on adaptive meshing if `remesh` is given. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

external

Puts the Sentaurus Structure Editor interface in external mode, which can be used to create polyhedra to be inserted into a Sentaurus Interconnect structure using the 3D MGOALS mode. When the external mode is switched on, all geometry transformations such as etching and deposition are applied to the external Sentaurus Structure Editor structure (see [Insertion: External Mode on page 276](#)).

logfile

Name of the file that will log all the Scheme commands dispatched to Sentaurus Structure Editor. The recommended file extension is `.scm`. The file will contain both the Scheme commands translated from Sentaurus Interconnect `etch`, `deposit`, `strip`, `photo`, and `transform` commands, and the user-specified Scheme commands inside the `sde` command.

The log file can be used for fine-tuning and debugging in a standalone run of Sentaurus Structure Editor such as:

```
sde -l mylogfile.scm
```

off

Switches off the Sentaurus Structure Editor mode if it has been switched on previously. By default, the Sentaurus Structure Editor mode is off. Operations will be performed by MGOALS instead.

on

Switches on Sentaurus Structure Editor for 3D geometry modeling.

polyhedron

Used for external mode only (see `external`). This polyhedron is used to initialize the external Sentaurus Structure Editor interface. The material to be used for this polygon is chosen using `polyhedron.material`.

`polyhedron.material`

Selects the material of the polyhedron that is used to initialize the external mode (see `external` and `polyhedron`). Default: `Silicon`.

remesh

Forces a remesh at the end of the `sde` command.

<Sentaurus Structure Editor commands>

Any number of `sde` commands in the Scheme scripting language. You must enclose the Scheme commands in a pair of braces to protect them from the Tcl command interpreter. The opening brace must be on the same line as the `sde` command, for example:

```
sde {
  (sdepe:depo "thickness" 0.01 "type" "iso" "algorithm" "pt"
    "max-chamfer-angle" 30 "steps" 1 "material" "Oxide")
  (sdeio:save-tdr-bnd "all" "out1_sde.tdr")
}
```

NOTE Commands to orientate Sentaurus Structure Editor based on Sentaurus Interconnect settings are sent first through the interface before user commands. These settings can be deleted if `sde:clear` is specified. Therefore, never specify `sde:clear` in this set of commands.

SdeCheck

Performs a geometry check of every boundary file that is created by Sentaurus Structure Editor. This helps to detect failures in the geometry-modeling part and prevents the Sentaurus Interconnect simulation from continuing after an incorrect boundary representation is found.

Description

This command enables and configures the interface between Sentaurus Interconnect and Sentaurus Structure Editor. When `sde on` is specified, all 3D geometry modeling is performed using Sentaurus Structure Editor. Sentaurus Interconnect will translate geometry-modifying commands to the Sentaurus Structure Editor language and retrieve the resulting modified

A: Commands

sde

structure when necessary. The supported commands are `deposit`, `etch`, `photo`, `strip`, and `transform`.

NOTE Scheme commands can be sent directly to Sentaurus Structure Editor using the `sde` command, but they must be enclosed in a pair of braces to prevent syntax errors in the Tcl interpreter. Several Scheme commands can be specified inside one `sde` command; each of them must start on a new line.

Examples

Enable the use of Sentaurus Structure Editor or geometry modeling, specify the log file for the Scheme commands, and check all boundary files written by Sentaurus Structure Editor:

```
sde logfile= depo.scm on SdeCheck
pdbSet InfoDefault 1
```

Create a cuboid in Sentaurus Structure Editor with tapered sidewalls and save the structure to a TDR file:

```
sde {
  (sdegeo:set-default-boolean "ABA")
  (define r1 (sdegeo:create-cuboid (position 0 0.6 0)
    (position 0.2 0.3 0.5) "Silicon" "Silicon_2"))
  (define facelist (list (car (find-face-id (position 0.1 0.3 0.25)))
    (car (find-face-id (position 0.2 0.5 0.25)))))
  (sdegeo:taper-faces facelist (position 0.2 0.3 0.5) (gvector 0 0 1) 5)
  (sdeio:save-tdr-bnd "all" "out1_sde.tdr")
}
```

See Also

For details about Scheme commands, refer to the *Sentaurus™ Structure Editor User Guide*.

select

Selects the variable for display in all postprocessing commands.

Syntax

```
select
  [delete] [edge.vector] [element] [element.to.gauss] [interfaces]
  [list] [list.all] [<material>]
  ([min | max] | [report.location])
  [name=<c>] [node.evaluate] [permanent] [present]
  [region=<c>] [store] [syntax.check.value=<c>] [value=<c>] [z=<c>]
```

Arguments

`delete`

Deletes the data field specified by name.

`edge.vector`

Computes the weighted field with respect to the edge orientation strongly favoring axis-oriented edges. Used with adaptive meshing.

`element`

Computes the field on elements interpolating fields in the `z` expression if necessary. If `!element` is specified and element fields appear in the `z` expression, those values are interpolated to the nodes first.

`element.to.gauss`

Computes the field on Gaussian points from the element data field specified in `name`. New data fields on Gaussian points will be created if the data field is not found.

`interfaces`

Computes the field or minimum or maximum on interfaces as well as bulk. Default: `true` (include interfaces).

`list`

Returns a list of currently defined and named real data fields. This returns a full Tcl list for use with those commands that require list variables.

A: Commands

select

`list.all`

Returns a list of currently defined and named data fields (for example, real data and vector data). This returns a full Tcl list for use with commands that require list variables.

`<material>`

Specifies the material to which the command applies. Different expressions for the data field initialization in different materials can be used. See [Specifying Materials on page 18](#).

`min, max`

Used with the `name` argument. When `min` or `max` is specified, the `select` command returns the minimum or maximum of the field name. You can limit the query to either a specific material using `<material>` or a specific region using `region`.

`name`

Name of the new data field. Default: `Z_Plot_Var`. This name is used by all the commands when a plot name is not specified. This is a powerful feature, as solution fields also can be created.

`node.evaluate`

Computes the divergence of a vector field at a node.

`permanent`

Returns 1 if the data field is written into permanent storage. If not, it returns 0.

`present`

Returns 1 if the data field with the name defined by `name` exists. If it does not exist, it returns 0.

`region`

Name of the region. Specifies the region to which the command applies. Different expressions for the data field initialization in different regions can be used.

`report.location`

Works with `min` and `max`. Reports the coordinate of the minimum or maximum value of the selected field.

`store`

Sets the data field with the name defined by `name` to be written into permanent storage when a structure file is output. Default: `false`.

`syntax.check.value`

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

`value, z`

Accepts an expression of data fields that are used to build a new data field. The operators `*`, `/`, `+`, `-`, and `^` all work as expected. The vector variables are listed here. The data fields available can be listed using the `list` argument. In addition to the listed data fields, the `x`- and `y`-coordinates can be specified. Several functions also are available to operate on data fields:

- `abs`: Absolute value
- `erf`: Error function
- `erfc`: Complementary error function
- `exp`: Exponential
- `log`: Logarithm
- `log10`: Logarithm base 10
- `sqrt`: Square root

Description

Data can be selected directly in most commands, but it is usually more effective to specify it with the `select` command, which allows for manipulation of data fields and also will list all currently defined data fields. The quantity can be computed on nodes (default) or elements using the `element` argument. In either case, if necessary, interpolation will be performed to obtain the proper value type (to obtain element values from nodal ones or vice versa).

NOTE The `select` command can be abbreviated to `sel`.

NOTE The `select` command always sets or retrieves data in internal units. Internal units are CGS, for example, pressure is dyn/cm^2 .

A: Commands

select

Examples

Select as the plot variable the base 10 logarithm of the MyData concentration:

```
select z= log10(MyData)
```

Select as the plot variable the MyData concentration minus a constant value of 5×10^{14} :

```
select z= (MyData - 5.0e14)
```

Select as the plot variable the difference between the MyData profile and an analytic profile. This data field will be named MyField. The store argument indicates that the doping field must be written into any saved structure files:

```
sel z= (MyData - 1.0e18 * exp ( y * y / 1.0e-8 )) name= MyField store
```

Set the value of the data field Pressure to 10^9 dyn/cm² (CGS units are used internally for mechanics):

```
sel z= 1.0e9 name= Pressure store
```

Delete the MyField data field:

```
select name= MyField delete
```

Calculate the electric field in a new data field called ElectricField. The store argument ensures that the new data field is stored to disk in subsequent calls of the struct command:

```
select z= "-diff(Potential)" edge.vector store name= ElectricField
```

List all available real data fields:

```
select list
```

Calculate the total heat field in a new data field called TotalHeat:

```
select z= "grad(Temperature)" node.evaluate store name= TotalHeat
```

See Also

All postprocessing commands

Set3DDeviceMeshMode

Sets the meshing parameters for the device simulation of complex 3D structures with curved surfaces.

Syntax

```
Set3DDeviceMeshMode
```

Description

This command is used when remeshing for device simulation of complex 3D structures with curved surfaces. It adjusts axis-aligned meshes to be more flexible on curved surfaces in three dimensions. The `Set3DDeviceMeshMode` command sets the following parameters:

For device simulations requiring a box-method Delaunay mesh:

```
pdbSet Grid SnMesh DelaunayType boxmethod
```

For axis-aligned imprinting:

```
pdbSet Grid SnMesh ImprintCoplanarFacesOnly 1           ;# <boolean>
pdbSet Grid SnMesh ImprintCoplanarityAngle 179.9        ;# <degree>
pdbSet Grid SnMesh ImprintAccuracy 1e-4                 ;# <um>
pdbSet Grid SnMesh ImprintCoplanarityDistance 1e-6      ;# <um>
```

For axis-aligned refinement:

```
pdbSet Grid SnMesh EdgeProximity 0.2                   ;# <unitless> default 0.05
pdbSet Grid SnMesh FaceProximity 0.2                   ;# <unitless> default 0.05
pdbSet Grid SnMesh CoplanarityAngle 175                ;# <degree>
pdbSet Grid SnMesh CoplanarityDistance 1e-4            ;# <um>
```

Examples

Use this command with the DelPSC algorithm to create high-quality triangulation on curved surfaces:

```
Set3DDeviceMeshMode
pdbSet Grid SnMesh Apply.Brep.DelPSC 1
pdbSet Grid SnMesh Apply.Brep.DelPSC.Resolution 1e-2    ;# <um>
pdbSet Grid SnMesh Apply.Brep.DelPSC.Accuracy 1e-4      ;# <um>
grid remesh info= 1
```

A: Commands
Set3DDeviceMeshMode

Limitation

Some complex structures have problematic geometric features that the DelPSC algorithm cannot tolerate. In such cases, you can use the `Set3DDeviceMeshMode` command without the DelPSC algorithm, for example:

```
Set3DDeviceMeshMode  
grid remesh info= 1
```

Set3DMovingMeshMode

Sets the meshing parameters and relevant parameters for 3D MovingMesh applications.

Syntax

```
Set3DMovingMeshMode <n>
```

Arguments

<n>

Specifies the minimum feature size in micrometer. Typically, it is the smallest layer thickness of all regions.

Description

This command simplifies the setup of moving-boundary problems by setting several parameters automatically. It checks the size of the structure and sets the appropriate parameters for the length scale. The argument <n> protects small geometric features from removal by geometry cleanup.

Examples

Set up fin bending of a 0.1 μm structure with 0.01 μm fin tip:

```
Set3DMovingMeshMode 0.01
```

See Also

[MovingMesh Algorithm on page 198](#)

A: Commands

SetFastMode

SetFastMode

Omits the solution of the thermal and electrical equations to simulate the device geometry quickly.

Syntax

```
SetFastMode
```

Description

This command runs the simulation quickly without simulating thermal and electrical effects. This can be useful when setting up a command file to confirm quickly that the geometry is satisfactory before simulating more computationally expensive steps.

SetMechanicsMeshMode

Sets the meshing parameters for 3D mechanics simulations.

Syntax

```
SetMechanicsMeshMode
```

Description

This command sets the meshing parameters for 3D mechanics simulation in large structures such as chip-package interfaces. It switches on the DelPSC algorithm for curved surfaces, minimizes axis-aligned imprinting on curved surfaces, prefers isotropic elements when refining surface meshes to conform to volume meshes, and adjusts distance tolerances to appropriate scales.

After this command is executed, subsequent mesh generation commands (for example, `grid remesh`) will generate meshes that are suitable for stress analysis.

NOTE This command only works in 3D simulations.

SetPerformanceMode

Modifies the automatic time-step control method and the parameters to improve simulation time.

Syntax

```
SetPerformanceMode
```

Description

Some of the algorithms are experimental and may not be suitable for all applications. Backward compatibility may not be available from release to release, since algorithms are refined constantly.

Examples

Change the default time-stepping scheme and the controls to allow more aggressive time-stepping:

```
SetPerformanceMode
```


SetPlxList

Sets a list of solution and term names to be passed to the `WritePlx` command.

Syntax

```
SetPlxList [<solution/term names>]
```

Arguments

<solution/term names>

Defines the name list to be passed to the `WritePlx` command.

Description

This command sets the list of fields to be saved in the next call to `WritePlx`. The list can contain solutions or term names.

Examples

Write a `.plx` file with the data fields `Temperature` and `Potential`:

```
SetPlxList {Temperature Potential}  
WritePlx T_and_P.plx
```

See Also

[WritePlx on page 681](#)

SetTDRList

Sets a list of solution or term names to be included when saving TDR format files.

Syntax

```
SetTDRList  
  [<solution/term names>]  
  [Solutions]
```

Arguments

<solution/term names>

Any known fields listed on the command line are added to files saved with `struct tdr=<c>`.

Solutions

Stores all solution variables (necessary for restarting a simulation). Using `!Solutions` switches off all default savings (only fields specified by name will be saved to TDR files).
Default: true.

Description

This command stores solution or term names in a TDR format file.

SetTemp

Sets the temperature value.

Syntax

```
SetTemp  
  <n> [<C> | <K>]
```

Arguments

<n>

Specifies the temperature. Default unit: degree Celsius.

Description

This command sets the temperature value. The value also is saved in a TDR file.

Examples

Set the temperature to 1000°C :

```
SetTemp 1000.0
```

SetTS4MechanicsMode

Sets mechanics-related parameters and models to match those of TSUPREM-4.

Syntax

```
SetTS4MechanicsMode [2008.09 | advanced]
```

Arguments

2008.09

Used for backward compatibility.

advanced

Sets TSUPREM-4 advanced settings.

Description

This command sets mechanics-related parameters and models in Sentaurus Interconnect to match TSUPREM-4 settings. This includes:

- Viscoelastic model and parameters
- Elastic moduli
- Stress relaxation factor setting
- Stress smoothing setting
- Thermal mismatch coefficients

The above parameters are set to match TSUPREM-4 defaults. The results may differ due to different numeric methods.

slice

Extracts a 1D data slice through a 2D to 3D simulation object.

Syntax

```
slice
  [include.interfaces] [<material>]
  [mdist] [mx] [my] [mz] [name=<c>]
  [only.interfaces]
  [p1= <list>]
  [p2= <list>]
  [side=<c>] [syntax.check.value=<c>]
  [value=<n>]
  [x=<n>] [<m>|<cm>|<um>|<nm>]
  [y=<n>] [<m>|<cm>|<um>|<nm>]
  [z=<n>] [<m>|<cm>|<um>|<nm>]
```

Arguments

`include.interfaces`

Includes interface values with the returned data. At an interface, the distance coordinate of the three nodes (two bulk and one interface) will be the same, and the interface value will be inserted between the two neighboring bulk values.

`<material>`

Specifies the material. See [Specifying Materials on page 18](#).

`mdist, mx, my, mz`

Changes the reporting information when interface materials are selected. The interface distance can be reported as projected along one of the three primary axes (`mx`, `my`, `mz`). Alternately, it can be reported as the distance along the extracted line (`mdist`).

`name`

Name of the data field. Default: `Z_Plot_Var`.

`only.interfaces`

Returns interface values exclusively in the returned data. When specified, no bulk values are returned.

A: Commands

slice

p1, p2

Specify the start point and endpoint for the outline. Each argument takes a list of numeric values.

The first, second, and third values in the list are taken as the x-, y-, and z-value, respectively. The missing value will be treated as zero. These arguments allow the `slice` command to extract data along an arbitrary line. The output from the `slice` command is a list of (*distance*, *value*) pairs, where *distance* is measured from the p1 point, and *value* is the extracted value of the selected quantity along the line.

NOTE Error messages will be generated if p1 and p2 are mixed with x, y, z, or value.

side

Takes the value from one of the two bulk materials consisting of the interface or ‘interface’ (literally) itself. If `side` is not specified, ‘interface’ itself is assumed. If `side` is specified as one of the bulk materials, the value of the selected quantity for the bulk material at the interface is returned. This argument is effective only if an interface material is specified.

syntax.check.value

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

value, x, y, z

Specify a outline for up to a four-dimensional solid, so that a 2D return is provided. For 1D simulations, none of these arguments is required. For 2D simulations, one is required. For 3D simulations, two are required. These requirements are reduced by one if an interface material has been specified. The default unit for x, y, and z is μm .

Description

This is an extremely powerful data analysis command. It extracts xy data along a slice through a specified material. It returns a Tcl list of coordinate–value pairs, where the coordinate is the distance [μm] along the reference segment, and the value is the local value of the argument specified either with the `-name` option or, if that is not provided, in the most recent `select` command. For example:

```
select z= Boron
set sliceRet [slice y= 0.5]
foreach { x value } $sliceRet {
    LogFile "$x\t$value\n"
}
```

This command will print and send to the log file the boron profile in x-coordinate value pairs at $y=0.5$.

The command extracts the selected variables as a function of position along a constant line. In one dimension, the command returns the concentration versus depth, for example. It also can extract a constant contour of the data selected and returns the coordinates of the isoconcentration line.

Examples

Return the selected variable as a function of depth at a constant lateral position of $0.01 \mu\text{m}$:

```
slice silicon y= 0.01
```

Return the x- and y-positions of a contour of the selected variable at 16.0:

```
slice silicon value= 16.0
```

Return the value of the selected quantity at the silicon side of the interface as a function of distance from the start of the interface:

```
slice silicon /oxide mdist side= silicon
```

Return the boron concentration along a line passing through points (0, 0) and (1, 2):

```
slice name= Boron p1= {0 0} p2= {1. 2}
```

See Also

[select on page 609](#)

[System on page 651](#)

solution

Obtains and sets solution parameters for generic solutions using the Alagator language.

Syntax

```
solution
  [add] [damp]
  [DiffStep] [linear | logarithmic] [list]
  [material.list= <list>]
  [name=<c>] [needsolution] [negative]
  [nosolve | solve | ifpresent=<c>]
  [present] [store] [unit=<c>]
```

Arguments

add

Creates a new solution.

damp

Applies a damping algorithm to the Newton iteration.

DiffStep

Determines in which solver the solution variable will be solved. DiffStep variables are solved with the PDE solver.

linear, logarithmic

Use linear or logarithmic extrapolation when extrapolating solution results for the next time step (see [math on page 501](#)).

list

Lists all the currently defined solutions.

material.list

List of materials where the solution variable will be solved. If the list is empty, the solution variable will be solved only if the equations are set for a specific material.

name

The character string used for the solution. Capitalization is not ignored, for example, Potential and potential are different. Abbreviations of names are not accepted.

needsolution

Returns true if the solution must be solved.

negative

Allows the specified solution to have negative values.

nosolve, solve, ifpresent

Only one of these options can be used at a time. They control the solution status for the next command:

- `nosolve` means do not solve.
- `solve` switches on the solution status for the next command.
- `ifpresent` sets up a conditional solve.

If all the solutions in the specified list also are being solved, this solution is solved.

present

Returns true if the solution is defined and a data field matches the name.

store

Allows the `solution` command to be stored in a TDR file.

unit

Unit of the solution variable. Default: cm^{-3} .

Description

This command creates and modifies solution names, and sets conditions for their inclusion in the matrix assembly. Solutions also can be listed and checked.

Examples

Create a solution named `Potential` and always solve for it. Allow the solution to have negative values and use damping on the Newton iteration updates:

```
solution name= Potential damp negative solve add
```

Create a solution name `Vac` and always solve for it. Do not use damping and do not allow values to become negative:

```
solution name= Vac !damp !negative solve add
```

A: Commands

solution

Return a list of all solutions:

```
solution list
```

Return a Boolean true if `Vac` has been defined and if there is a data field with the name `Vac`:

```
solution name= Vac present
```

See Also

[math on page 501](#)

[term on page 658](#)

solve

Simulates thermal, electrical, and mechanical analyses.

Syntax

```

solve
  (temp.ramp=<c> | time=<n>[<hr>|<min>|<s>] temperature=<n>[<C>|<K>])
  [Adaptive]
  [delNT=<n>] [<C>|<K>] [delT=<n>] [<C>|<K>]
  [density.increase= <regionName>=<n> | <material>=<n>] [eqnInfo]
  [init=<n>] [<hr>|<min>|<s>] [isolve]
  [maxstep=<n>] [<hr>|<min>|<s>]
  [minT=<n>] [<C>|<K>] [movie=<c>]
  [ramprate=<n>] [<C/s>|<K/s>|<C/min>|<K/min>]
  [reload] [stress.relax]
  [t.final=<n>] [<C>|<K>] [t.final.profile=<c>] [write.temp.file=<c>]

```

Arguments

Adaptive

If specified, `Adaptive` switches on adaptive meshing for this solution step. Parameters for adaptive meshing are described in [Adaptive Meshing During Analysis on page 184](#). The default is the return value of `pdbGet Grid Adaptive`.

delNT

Defines the maximum temperature step during a temperature ramp-down if specified. Default unit: degree Celsius. It also can be defined globally with the command:

```
pdbSet Compute delNT <n>
```

delT

Defines the maximum temperature step during a temperature ramp-up if specified. Default unit: degree Celsius. It also can be defined globally with the command:

```
pdbSet Compute delT <n>
```

density.increase

Value of density increase, which can be specified either per region `<regionName>=<n>` or per material `<material>=<n>`.

eqnInfo

Allows equation updates to be printed to the log file during the Newton iteration.

A: Commands

solve

`init`

First time step. The default is 0.0001 s, which is sometimes inappropriate for defect simulations, particularly in cases of damage. Default unit: minute.

`isolve`

Switches on or off the initial solve for models that require an equation solved to set the initial conditions. It is used mainly to initialize SCM circuit nodes.

`maxstep`

Maximum time step. Default unit: minute.

`minT`

Minimum simulation temperature. If the simulation temperature falls below this value, the thermal and electrical solver is switched off. If it occurs during a ramp, the time-stepping is altered such that the solver switches on or off exactly at this temperature. Default value and unit: 0°C.

`movie`

Specifies actions that occur during the anneal step. For every time step of the solution, the character string value of this argument is executed. Do not use the `movie` argument to specify operations that can change meshes. For example, `movie= {grid remesh}` must not be used.

`ramprate`

Temperature change during anneal. Default value and unit: 0°C/s.

`reload`

Allows thermal and electrical equations to be parsed at each time step.

`stress.relax`

Switches off relaxation of stresses during solution with an inert ambient. Default: true.

`t.final`

Final temperature for a temperature ramp-up or ramp-down. It is used if `ramprate` is not given. The ramp time is calculated automatically. Default unit: degree Celsius.

`t.final.profile`

Name of the final temperature profile for a local temperature ramp-up or ramp-down. The temperature at each node is ramped up or down linearly, in the given time, from the current value to the value specified with the final profile. Default unit: kelvin.

`temp.ramp`

Name of a temperature ramp created with the `temp_ramp` command.

`temperature`

Annealing temperature. Default unit: degree Celsius.

`time`

Annealing time. Default unit: minute.

`write.temp.file`

Name of the file where the thermal profile created during laser annealing is stored. The format of the file is two columns: time (in seconds) and temperature (in degree Celsius). This file can be used to create a `temp_ramp` to allow subsequent simulations to use the computed temperature profile without the need to resimulate laser annealing.

Description

The arguments of the `solve` command set the simulation conditions as well as time-stepping options. Analysis models and parameter settings are performed with the `pdBSet` command or `mode` command. Temperature ramps are specified by first creating a list of ramping steps using the `temp_ramp` command. Then, the ramp is applied with `temp.ramp`. All `temp_ramp` command arguments can be specified with the `solve` command.

Examples

Perform a simple low-temperature solve for 30 s:

```
solve time= 30<s> temp= 900
```

Perform a solve step using the temperature ramp named `spike`:

```
solve temp.ramp= spike
```

Perform a solve step using the current temperature:

```
solve
```

See Also

[temp_ramp on page 655](#)

sptopo

Transfers structures and dispatches commands to Sentaurus Topography.

Syntax

```
sptopo {<Sentaurus Topography commands>}
```

Arguments

<Sentaurus Topography commands>

Any number of Sentaurus Topography commands. Enclose the commands in a pair of braces to protect them from interpretation by the Tcl interpreter.

Description

This command transfers the boundary representation of the current structure and dispatches the commands to Sentaurus Topography. After executing the commands in Sentaurus Topography, the modified structure is retrieved and remeshed in Sentaurus Interconnect.

NOTE A license for Sentaurus Topography must be available, and a version of the Sentaurus Interconnect binary with Sentaurus Topography included must be installed.

Examples

Use Sentaurus Topography to deposit two layers isotropically, that is, oxide with a thickness of 0.005 μm (5 nm) and polysilicon with a thickness of 0.18 μm :

```
sptopo {  
    deposit material= Oxide thickness= 0.005  
    deposit material= PolySilicon thickness= 0.180  
}
```

See Also

Sentaurus™ Topography User Guide

For 3D operations, see [topo on page 660](#).

stdiff

Compares the current structure with one from a TDR file.

Syntax

```
stdiff <c>
```

Arguments

<c>

Specifies the full path or the prefix of a TDR file. The prefix is the file name without `_sis.tdr`.

Description

This command reads the external TDR file, interpolates the data onto the current structure, compares data, and reports if data exceeds the relative error criteria (subject to the absolute error minimum value).

Examples

Compare field values in the current structure in memory with those contained in the file `n1_sis.tdr`:

```
stdiff n1_sis.tdr
```

stressdata

This command:

- Defines the intrinsic stress of materials for use in stress calculations.
- Defines boundary or loading conditions for stress analysis.
- Reports the maximum stress values and their locations.
- Prints the anisotropic material matrix.
- Defines edge dislocation settings.
- Defines submodeling settings.

Syntax

```
stressdata
  [<material> | region=<c>]
  [deposit.intrinsic]
  [sxxi=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [syyi=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [szzi=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [sxyi=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [syzi=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [szxi=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]

  [syntax.check.value]

  [base=<n>] [<m> | <cm> | <um> | <nm>]
  [sxx1=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [syy1=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [szz1=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [sxx2=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [syy2=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]
  [szz2=<n>] [<atm> | <GPa> | <MPa> | <KPa> | <Pa> | <dyn/cm2>]

  [bc.location= Left | Right | Front | Back | Bottom | <c>]
  [bc.proc= {
    [dxP=<c>] | [dyP=<c>] | [dzP=<c>] |
    [fxP=<c>] | [fyP=<c>] | [fzP=<c>] |
    [sxP=<c>] | [syP=<c>] | [szP=<c>] }]
  [bc.rotation.axis= {[xa=<n>] | [ya=<n>] | [za=<n>]}]
  [bc.value= {
    [dx=<n>] | [dy=<n>] | [dz=<n>] | [pressure=<n>] |
    [fx=<n>] | [fy=<n>] | [fz=<n>] |
    [mx=<n>] | [my=<n>] | [mz=<n>] |
    [pdx=<n>] | [pdy=<n>] | [pdz=<n>] |
    [pfx=<n>] | [pfy=<n>] | [pfz=<n>] |
```



```

    [rx=<n>] | [ry=<n>] | [rz=<n>]
    [sx=<n>] | [sy=<n>] | [sz=<n>] }}
[point.coord= {<n> <n> <n>}]

[number=<n>]
[sxx] [syy] [szz] [sxy] [syz] [szx] [hs] [pr] [ps] [vms]

[print.anisotropic.matrix]
[print.equiv.Poisson.matrix]
[print.equiv.ShearMod.xy] [print.equiv.ShearMod.yz]
[print.equiv.ShearMod.zx]
[print.equiv.YoungsMod.x] [print.equiv.YoungsMod.y]
[print.equiv.YoungsMod.z]

[apply.dislocation]
[dislocation.origin= {<n> <n> <n>}]
[name=<c>]
[para.orient= {<n> <n> <n>}]
[perp.orient= {<n> <n> <n>}]
[saveTDR] [stress.relax]

[opt.maxiter=<n>] [opt.mindnrm=<n>] [opt.mindssq=<n>] [opt.tolerance=<n>]
[optimize.dislocation]
[origin.max= {<n> <n> <n>}]
[origin.min= {<n> <n> <n>}]

[resolved.shear.stress]
[slip.direction= {<n> <n> <n>}]
[slip.plane.normal= {<n> <n> <n>}]

[align.cut=<n>] [global.model=<c>] [max= {<n> <n> <n>}] [min= {<n> <n> <n>}]

```

Arguments

<material>

Material in which the stress parameters are to be set.

region

Region where the stresses are to be applied.

deposit.intrinsic

Specifies that the intrinsic stress introduced during material deposition or insertion will be defined.

A: Commands

stressdata

sxxi, syyi, szzi, sxyi, syzi, szxi

Intrinsic stresses. Default unit: dyn/cm^2 .

syntax.check.value

Sets a value to be returned only during syntax-checking mode. Sometimes, the value returned by a command can cause a false syntax-check error because the value returned by the command would not be the value during the normal run mode. Setting this value avoids such problems.

Arguments: Width-Dependent Intrinsic Stress

base

Base width. Default unit: μm .

sxx1, syy1, szz1

Scale factors in linear width-dependent intrinsic stress. Default unit: dyn/cm^2 .

sxx2, syy2, szz2

Scale factors in natural logarithmic width-dependent intrinsic stress. Default unit: dyn/cm^2 .

Arguments: Boundary or Loading Conditions

bc.location

Specifies the area where the boundary or loading conditions are applied.

Left | Right | Front | Back | Bottom refer to the outer boundary surfaces of the simulation domain. If `bc.location` specifies a contact name, the corresponding contact defined through this contact name must exist. If a point-force boundary condition is required, `bc.location` specifies the name of the point-force boundary condition. If a point-displacement rate boundary condition is required, `bc.location` specifies the name of the point-displacement rate boundary condition.

bc.proc

Specifies procedure names for loading conditions. The type can be:

- $dxP/dyP/dzP$ for the name of the displacement rate procedure.
- $fxP/fyP/fzP$ for the name of the total force procedure.
- $sxP/syP/szP$ for the name of the distributed force procedure.

`bc.rotation.axis`

Specifies the coordinates of the point around which rotational boundary conditions or moment boundary conditions are applied. Default unit: cm.

`bc.value`

Specifies the boundary or loading condition types and values. The type can be:

- `dx/dy/dz` for the displacement rate (default unit: cm/s).
- `pressure` for pressure (default unit: dyn/cm²).
- `fx/fy/fz` for total force (default unit: dyne).
- `mx/my/mz` for the moment (default unit: dyn cm).
- `pdx/pdy/pdz` for the point displacement rate (default unit: cm/s).
- `pdfx/pdfy/pdfz` for point force (default unit: dyne).
- `rx/ry/rz` for rotational velocity (default unit: rad/s).
- `sx/sy/sz` for distributed force (default unit: dyn/cm²).

`point.coord`

Specifies the location where the point force or the point displacement rate is applied. Default unit: μm .

Arguments: Maximum Stress List

`number`

Specifies the number of the largest stress values to report.

`sxx, syy, szz, sxy, syz, szx, hs, pr, ps, vms`

Specifies from which stress component (`sxx`, `syy`, `szz`, `sxy`, `syz`, `szx`) or which derived stress (`vms` is the von Mises stress, `ps` is the principal stress, `hs` is the hydrostatic stress, and `pr` is the pressure) to extract the maximum stress values. Values for stress components and principal stresses are computed at element centroid, while values for von Mises stresses, hydrostatic stresses, and pressures are computed at nodes.

NOTE To extract maximum principal stresses, use:

```
pdbSet Mechanics Calculate.Principal.Stress 1
```

A: Commands

stressdata

Arguments: Anisotropic Material Matrix

`print.anisotropic.matrix`

Prints the anisotropic material matrix.

`print.equiv.Poisson.matrix`

Prints the equivalent Poisson ratio matrix.

`print.equiv.ShearMod.xy`

Prints the equivalent shear modulus in the xy plane.

`print.equiv.ShearMod.yz`

Prints the equivalent shear modulus in the yz plane.

`print.equiv.ShearMod.zx`

Prints the equivalent shear modulus in the zx plane.

`print.equiv.YoungsMod.x`

Prints the equivalent Young's modulus in the x-direction.

`print.equiv.YoungsMod.y`

Prints the equivalent Young's modulus in the y-direction.

`print.equiv.YoungsMod.z`

Prints the equivalent Young's modulus in the z-direction.

Arguments: Edge Dislocation

`apply.dislocation`

Indicates that an edge dislocation will be defined.

`dislocation.origin`

Specifies the location of the dislocation core.

`name`

Specifies the name of the edge dislocation.

`para.orient`

Specifies the direction of the edge dislocation.

`perp.orient`

Specifies the Burgers vector in the perpendicular direction to the half plane. The magnitude is the slip distance.

`saveTDR`

Saves the currently defined edge dislocation into a TDR file for visualization.

`stress.relax`

Switches on relaxation of stresses after superposing the dislocation-induced stress field.
Default: true.

Arguments: Edge Dislocation–induced Strain Energy Minimization

`opt.maxiter`

Specifies the maximum number of iterations allowed in the optimization loop. Default: 500.

`opt.mindnrm`

Specifies the change in norm of the parameter vector for convergence. Default: 5e-3.

`opt.mindssq`

Specifies the relative change in the sum of the squares for convergence. Default: 1e-5.

`opt.tolerance`

Specifies the tolerance of target errors. Default: 1e-3.

`optimize.dislocation`

Switches on the elastic strain energy minimization of edge dislocations defined with `!apply.dislocation`.

`origin.max`

List of numbers defining the x-, y-, and z-coordinates of the lower-right front corner of the bounding box for the location of the edge dislocation core.

`origin.min`

List of numbers defining the x-, y-, and z-coordinates of the upper-left back corner of the bounding box for the location of the edge dislocation core.

A: Commands

stressdata

Arguments: Resolved Shear Stress

`resolved.shear.stress`

Switches on the calculation of resolved shear stress.

`slip.direction`

Specifies the slip direction.

`slip.plane.normal`

Specifies the normal of the slip plane.

Arguments: Submodeling

`align.cut`

Specifies the tolerance to be used when performing the cut. When this argument is specified, the cutting algorithm looks for mask edges close to the `min` and `max` coordinates and snaps these coordinates to the mask-edge coordinates if they are closer than `<n>` from each other.

`global.model`

Specifies the file name of the global model from which to submodel; `off` is reserved to switch off submodeling.

`max`

List of numbers defining the x-, y-, and z-coordinates of the lower-right front corner of the cutting box in the unified coordinate system (UCS).

`min`

List of numbers defining the x-, y-, and z-coordinates of the upper-left back corner of the cutting box in the UCS.

Description

This command provides stress analysis parameters for input and output. Zero is the default value for all intrinsic stress parameters. Wherever possible, use the `deposit` command with specified stresses to apply intrinsic stresses.

Examples

Set the yy component of the intrinsic stress in nitride to 1.4×10^{10} dyn/cm²:

```
stressdata nitride syyi= 1.4e10
```

Define the cuboid region for submodeling with the upper-left back corner (-0.6, 0.04, 0.25) and the lower-right front corner (0.0, 0.16, 0.45), and set the global model to `global3d`:

```
stressdata global.model= global3d min= {-0.6 0.04 0.25} max= {0.0 0.16 0.45}
```

strip

Completely removes a layer exposed to the top gas region.

Syntax

```
strip <material> [remesh] [stress.relax]
```

Arguments

<material>

The specified material, if exposed, is completely removed. See [Specifying Materials on page 18](#).

remesh

By default, the structure is remeshed in two dimensions after strip. Setting !remesh prevents remeshing. In three dimensions, the boundary is changed without generating a mesh, so this argument has no effect in three dimensions. Preventing remeshing can save time for very large structures.

stress.relax

Specify this option to switch on relaxation of stresses after the removal operation. Default: false.

Description

In two dimensions, the mesh is regenerated immediately. In three dimensions, only the boundary is modified and the mesh is regenerated later when necessary.

Examples

Remove all oxide regions exposed to the top gas region:

```
strip oxide
```

struct

Writes files containing the structure or the mesh and solutions.

Syntax

```
struct
  [Adaptive] [alt.maternames] [compress.TDR] [contacts]
  [csse] [deform] [deform.scale=<n>]
  [FullD] [Gas] [interfaces]
  [mshcmd] [pdb] [pdb.only]
  [sat] [satfile=<c>] [simplify=<n>]
  [tdr=<c>] [tdr.bnd=<c>]
  [visual.1D]
  ( [x=<n>] [<m>|<cm>|<um>|<nm>]
    [y=<n>] [<m>|<cm>|<um>|<nm>]
    [z=<n>] [<m>|<cm>|<um>|<nm>] )
```

Arguments

Adaptive

In three dimensions, meshing is delayed until it is needed; to save a file, a mesh may need to be created. `Adaptive` controls whether adaptive meshing is used. The default value is the return of `pdbGet Grid Adaptive`.

alt.maternames

If you had chosen alternative names for materials using `mater alt.matername=<c>` or `region alt.matername=<c>`, they are written to the TDR file in addition to the Sentaurus Interconnect material name.

For example, a Silicon region containing a high concentration of Germanium is given the `alt.matername` of `SiliconGermanium` automatically. When writing a TDR file, the material name of this region is `SiliconGermanium`, but the name `Silicon` is stored as well, so that when reloading the TDR file, Sentaurus Interconnect again recognizes this as a Silicon region.

compress.TDR

Applies secondary compression to TDR files. Default: false.

It can also be switched on globally using the command:

```
pdbSet Compress.TDR 1
```

A: Commands

struct

contacts

Writes contacts defined in the `contact` command into the boundary file. Default: true.

csse

Splits a composite solid shell element (CSSE) into multiple layers for visualization (see [Mechanics Simulations With Composite Solid Shell Element on page 71](#)).

deform

Saves the deformation applied structure. By default, the gas mesh is not saved.

deform.scale

Defines the scale by which the deformation will be magnified in the calculation. Default: 1.

FullD

If `FullD` is specified, the mesh is extruded to the maximum dimension allowed in the simulation temporarily before saving the file. After saving the file, the simulation continues in the same dimension as before.

If `!FullD` is specified, the saved files contain mesh and data in the dimension currently used in the simulation.

NOTE When TDR restart files are saved, by default, no extrusion is performed.

Gas

By default, Sentaurus Interconnect writes regions of material gas to TDR files. If `!Gas` is specified, regions of material gas are not saved.

interfaces

Saves interface data in TDR files. Specify `!interfaces` to prevent storing interface data. Default: true.

mshcmd

When specified with `tdr` or `tdr.bnd`, `mshcmd` writes a command (.cmd) file with refinement information readable by Sentaurus Mesh.

pdb

Saves `pdb` parameters along with the geometry and data in a TDR file.

pdb.only

Saves only `pdb` parameters (without geometry and data) in a TDR file.

sat

Enforces or prevents the saving of a Sentaurus Structure Editor restart file. This option works only when Sentaurus Structure Editor is switched on internally or is used as an external tool.

satfile

When using Sentaurus Structure Editor for 3D geometry-modeling steps, a Sentaurus Structure Editor restart file is saved by default. The argument `satfile` defines the file name. The default file extension is `.sat`. This argument works only when Sentaurus Structure Editor is switched on internally or is used as an external tool.

simplify

When saving a 2D or 3D boundary in TDR files, the extracted geometry is simplified before being saved. The `simplify` argument defines the maximum deviation of the simplified boundary from the extracted geometry.

tdr

Saves a file in TDR format. The extension `_sis.tdr` is added automatically. By default, all modifications to the parameter database are written to the TDR file to support splitting and restarting simulations.

By default, TDR files can be used to split and restart simulations. Coordinates and field values are stored with their unscaled internal values.

For information about the TDR format, refer to the *Sentaurus™ Data Explorer User Guide*.

tdr.bnd

Writes a TDR file that contains only the boundary representation.

visual.1D

Applies only to 1D simulations. If specified, Sentaurus Interconnect orders the nodes when writing them in a TDR file, so that the file can be easily visualized with Sentaurus Visual.

x, y, z

Saves a 2D cross section of a 3D structure, or a 1D cutline of any dimension structure in a TDR file. Since the file is stored in TDR format, the `tdr` argument must be used together with these arguments.

For 1D simulations, none of these arguments is required.

For 2D simulations, to save a 1D cutline, one of these arguments is required.

A: Commands

struct

For 3D simulations, to save a 1D outline, two of these arguments are needed. To save a 2D cross section, one of these arguments is required. The saved 2D cross-section TDR file can be used to restart a simulation if it is a z-cut. If you want to use other cuts for restarting simulations, rotate the structure first and then perform a z-cut.

Default unit: .

NOTE Only nodal data, not vector or tensor data, is stored in the saved file.

Description

This command writes the structure and the simulation mesh and field data to one or several files. The data saved is from the current set of solution values.

Examples

Write a TDR file with the current simulation mesh and data. By default, a restart file is written:

```
struct tdr= output
```

Write two files (`output_sis.tdr` and `output_bnd.tdr`) and apply secondary compression to the files:

```
struct tdr= output compress.TDR
```

See Also

[contact on page 381](#)

[integrate on page 467](#)

[math on page 501](#)

supply

Sets the boundary conditions on contacts and the distributed sources on regions.

Syntax

```
supply
  (clear | delete | list | name=<c> | print)
  [ charge=<n>[<Coulomb>] |
    charge.density=<n>[<C/cm3>] power=<n>[<W>|<mW>|<uW>|<nW>] |
    current=<n>[<A>] | current.density=<n>[<A/cm2>] | current.ramp=<c> |
    heat.flux=<n>[<W/cm2>] | power.density=<n>[<W/cm3>] | power.ramp=<c> |
    temp.ramp=<c> | temperature=<n>[<C>|<K>]]
  [contact.name=<c>]
  [heat.transfer=<n><W/cm2/K>]
  [region.name=<c>]
  [<solution>]
  [voltage=<n>[<V>] | voltage.ramp=<c>]
```

Arguments

charge

Specifies a constant charge supply.

charge.density

Specifies a constant charge density–distributed source.

clear

Deletes all supplies in the system.

contact.name

Adds a supply to the specified contact.

current

Specifies a constant current supply.

current.density

Specifies a constant current density supply.

current.ramp

Specifies a ramping current supply.

A: Commands

supply

`delete`

Deletes a specified supply.

`heat.flux`

Specifies a constant heat flux supply.

`heat.transfer`

Specifies the heat emission transfer rate at the contact given with the `contact.name` argument. If `temperature` is specified as well, it is used as the environment temperature.

`list`

Lists the supplies in the system.

`name`

Name of a supply. The name is always composed of a solution variable name followed by a contact name or region name.

`power`

Specifies a constant power supply.

`power.density`

Specifies a constant power density–distributed source.

`power.ramp`

Specifies a ramping power or power-density supply. The power-density supply must be used with a region instead of a contact.

`print`

Prints information about a specified supply.

`region.name`

Adds a heat source to the specified region.

`<solution>`

Specifies the solution name for the supply.

`temp.ramp`

Specifies a ramping temperature supply.

temperature

Specifies a constant temperature supply.

voltage

Specifies a constant voltage supply.

voltage.ramp

Specifies a ramping voltage supply.

Description

This command adds, deletes, lists, and prints:

- Boundary conditions (biases) on contacts.
- Heat sources on noncontact bulk regions.

Each solution variable can have, at most, one supply object associated with any one contact. The name of a supply object is generated automatically by concatenating the name of the pertinent solution variable with the name of the contact or region to which the supply is applied, for example, TemperatureMyThermalContact.

Examples

Apply a 0.1 mV constant voltage to contact Cont1:

```
supply contact.name= Cont1 voltage= 1.0e-4<V>
```

Inject a constant power of 1 mW into thermode therm1:

```
supply contact.name= therm1 power= 1.0e-3<W>
```

Add a constant heat source of 2×10^8 W/cm³ to the region named Copper_1:

```
supply region.name= Copper_1 power.density= 2.0e8<W/cm3>
```

Add a constant heat emission flux with a rate of 1×10^5 W/cm²K at a contact named thermode using an environment temperature of 300 K:

```
supply contact.name= thermode temperature= 300<K> heat.transfer= 1e5
```

Generate a Tcl list of all supplies in the system:

```
supply list
```

Delete all supplies in the system:

```
supply clear
```

A: Commands

supply

Erase the supply affiliated with the solution variable `Potential` on contact `Cont1`, that is, erase any electrical boundary condition set on this contact:

```
supply delete name= PotentialCont1
```

Print all information about supply object `PotentialCont1`:

```
supply print name= PotentialCont1
```

See Also

[Chapter 5 on page 121](#)

[Chapter 6 on page 131](#)

[contact on page 381](#)

[current_ramp on page 393](#)

[temp_ramp on page 655](#)

[voltage_ramp on page 679](#)

System

Adds a circuit element to the simulation.

Syntax

```
System {  
    <model> <name> (<electrode> <electrode>) {<parameters>}  
    Set (<c>=<n>)  
    Thermal (<c> <c> ...)  
}
```

Arguments

<electrode>

Specifies the electrodes. The parentheses are part of the syntax in this case.

<model>

Specifies the model instance from which the circuit element is derived.

<name>

Name of the circuit element.

<parameters>

Specifies the parameters associated with the circuit element.

Set

Sets Dirichlet boundary conditions on the thermodes.

Thermal

Specifies the nodes to be thermal nodes.

Description

This command allows you to add and to describe a circuit element.

Each *system* command can specify multiple instances within the braces. A circuit element is defined by the model instance from which it is derived, a name, its electrodes, and the parameters associated with the circuit element.

A: Commands

System

Examples

Define a resistor `r3` derived from model `Resistor_pset`. The element `r3` is placed between electrodes 1 and 0, and has a resistance of $1000\ \Omega$:

```
System {  
  Resistor_pset r3 (1 0) {resistance= 1000}  
}
```

tclsel

Selects the plot variable for the postprocessing routines.

Syntax

```
tclsel  
  [list] [<material>] [name=<c>]  
  [store] [vec] [z=<c>]
```

Arguments

list

Returns a list of currently defined and named data fields. The real data fields are listed by default. Vector data fields can be listed using `vec`. This returns a Tcl list for use with those commands that require list variables.

<material>

Specifies the material to which the command applies. Different expressions for the data field initialization in different materials can be used. See [Specifying Materials on page 18](#).

name

Name of the new data field. Default: `z_plot_var`.

This is used by all commands when a plot name is not specified. This is a powerful feature, as solution fields also can be created.

store

Controls whether the data field is written into permanent storage when a structure file is output. Default: `false`.

vec

Lists the vector data fields. Default: `false`.

z

Accepts a Tcl expression that are used to build a new data field. All valid Tcl expressions can be used in the string. Existing data fields are defined as Tcl variables, and the expression is evaluated node-by-node with the updated value of the variable. In general, this argument must be enclosed in braces, so that variable substitution is performed when the string is parsed.

A: Commands

tclsel

Description

This command specifies the plot variable for almost all other plot commands. It is a companion to the `select` command, but it differs from the `select` command in that it accepts any general Tcl expression. Data fields are made into Tcl variables and can be accessed with standard Tcl variable methods.

Examples

Select as the plot variable the base 10 logarithm of the arsenic concentration:

```
tclsel z= {log10($Arsenic)}
```

Select as the plot variable the phosphorus concentration minus a constant value of 5×10^{14} :

```
tclsel z= {($Phosphorus - 5.0e14)}
```

Select as the plot variable the difference between the phosphorus and an analytic profile. This data field will be named `Doping`. The `store` argument indicates that the doping field must be written into any saved structure files:

```
tclsel z= {($Phosphorus - 1.0e18 * exp ( $y * $y / 1.0e-8 ))} \  
name= Doping store
```

List all available real and vector data fields:

```
tclsel list vec
```

See Also

All postprocessing commands

temp_ramp

Defines a temperature profile for use with the `solve` or `supply` command.

Syntax

```
temp_ramp
  (clear | list | name=<c>)
  [current.time=<n>] [<hr>|<min>|<s>]
  [delNT=<n> [<C>|<K>] | delT=<n> [<C>|<K>]]
  [density.increase= <regionName>=<n> | <material>=<n>]
  [hold]
  [last]
  [ramprate=<n>] [<C/s>|<K/s>|<C/min>|<K/min>]
  [read.temp.file=<c>]
  [reset.init.time]
  [t.final=<n>] [<C>|<K>]
  [temperature=<n>] [<C>|<K>] |
  [time=<n>] [<hr>|<min>|<s>]
```

Arguments

`clear`

Clears the global list of temperature ramps. When defining profiles, the action is to unite the new definition with any prior profiles of the same name.

`current.time`

Returns the value of the ramp for the given time. Default unit: minute.

`delNT`

Defines the maximum temperature step during a temperature ramp-down if specified. Default unit: degree Celsius. It also can be defined globally with the command:

```
pdbSet Compute delNT <n>
```

`delT`

Defines the maximum temperature step during a temperature ramp-up if specified. Default unit: degree Celsius. It also can be defined globally with the command:

```
pdbSet Compute delT <n>
```

`density.increase`

Applies densification model where the density increases in percentage. The increase value can be specified per region `<regionName>=<n>` (where `regionName` is the name of an

A: Commands

temp_ramp

existing region in the current structure) or per material `<material>=<n>` (where `material` is the name of a material in the current structure).

hold

During this segment of the temperature ramp, `hold` gives the `solve` command the opportunity to specify the time of the segment.

last

Defines the final component of the temperature profile. There will be no more additions to the ramp.

list

Generates a list of temperature profiles. It returns a Tcl list and can be operated on as such. The default action for commands is to print the return, so if no handling is required, this prints a list of names of defined temperature profiles. If a name is specified, the `temp_ramp` command only is listed along with details about the ramps.

name

Name used to identify the temperature ramp. Use this name in a subsequent `solve` command.

ramprate

Temperature change during anneal. Default value and unit: 0.0°C/s .

read.temp.file

Reads a thermal profile from a file. It must not be used with any other thermal specification. To create this profile file during laser annealing, use `write.temp.file` of the `solve` command. The format of the file is two columns: time (in seconds) and temperature (in degree Celsius). Lines beginning with a hash (#) are ignored.

reset.init.time

Starts each annealing step with the same initial time step.

t.final

Final temperature for a temperature ramp-up or ramp-down. It is used if `ramprate` is not given. The ramp time is calculated automatically. Default unit: degree Celsius.

temperature

Annealing temperature. Default unit: degree Celsius.

time

Annealing time. Default unit: minute.

Description

This command specifies multiple-step temperature ramps and holds. It can be used to construct a complex temperature sequence to be simulated with the `solve` command (by specifying `temp.ramp` of the `solve` command).

Examples

Define the temperature profile named `tr1` with a temperature rate of 10 K/s:

```
temp_ramp name= tr1 temp= 20 ramprate= 10<K/s> time= 100<s>
```

See Also

[solve on page 629](#)

term

Defines a new subexpression for use in the equation specification of the Alagator language.

Syntax

```
term
  [add] [clear] [delete] [eqn=<c>] [list]
  [<material>]
  [name=<c>] [print] [store]
```

Arguments

add

Creates a new term. A term with that name will be overwritten.

clear

Removes a term from the current set if the term exists. Otherwise, it clears the content of all terms.

delete

Removes a term from the current set.

eqn

The string defines the equation part of the term. The equation must conform to all the standard constraints of the Alagator language. Terms can be nested; the equation specified here can refer to other terms. Parsing of the equation is performed during the `solve` command, so there is no need for everything to be predefined.

list

Lists all the names of the current terms. This is returned as a Tcl list, so it can be used in conjunction with all the list functionality.

<material>

If a material is specified, the term becomes specific for this material only. This allows the same name to have different equations in different materials. See [Specifying Materials on page 18](#).

name

Reference name for the term. This name is defined and is compared to strings found in the equation parsing. Capitalization is important, and only exact matches are allowed.

print

Prints the equation for the term matching the name specified. If no term matches, 0.0 is returned. If the material name is not given, the first term with the matching name is returned (for example, you may obtain `VTOTAL` in oxide instead of silicon).

store

Allows the `term` command to be stored in a TDR file.

Description

Terms are never required but can offer substantial computational benefit. Each term is evaluated only once during assembly, and the results are cached. If multiple equations refer to a term, the first equation to use it evaluates the expression and the remainder use the cached values. Terms are usually used for expressions that need to appear in several partial differential equations.

Terms can be created, searched, and printed, which allow inquiries about terms to be made in the various callback procedures.

Examples

Create a term named `VTOTAL` in silicon only. The keyword `VTOTAL` will be replaced with the subexpression `Vacancy+VacancyGbc`:

```
term name= VTOTAL add silicon eqn= "Vacancy+VacancyGbc" store
```

Create a term named `Noni` in silicon only. The equation will be the exponential of `Potential` multiplied by `$Vti`. The normal rules for Tcl string variables and executions apply, so that `Vti` must be a currently defined variable. The value will be replaced when the `term` command is executed:

```
term name= Noni add silicon eqn= exp(Potential*$Vti)
```

Return a list of all the current term names defined:

```
term list
```

Return the currently defined equation corresponding to the name `Charge`:

```
term name= Charge print
```

See Also

[solution on page 626](#)

topo

Performs 3D etching and deposition using Sentaurus Topography 3D.

Syntax

```
topo {<Sentaurus Topography 3D commands>}
```

Arguments

<Sentaurus Topography 3D commands>

All arguments of the `topo` command are described in the *Sentaurus™ Topography 3D User Guide*.

Description

Physical etching and deposition are available through the interface to Sentaurus Topography 3D and are executed using the `topo` command.

Commands entered into the `topo` command are passed directly to the Sentaurus Topography 3D library. The exchange of the boundary between Sentaurus Interconnect and Sentaurus Topography 3D is handled automatically and only when required.

NOTE A licence for Sentaurus Topography 3D must be available, and a version of the Sentaurus Interconnect binary with Sentaurus Topography 3D included must be installed.

transform

Cuts, flips, reflects, rotates, stretches, or translates a structure and currently defined refinement boxes.

Syntax

```
transform
  cut [Adaptive] [location=<n>[<m>|<cm>|<um>|<nm>] (left | right | front |
    back | up | down)]
    [max= {
      <n>[<m>|<cm>|<um>|<nm>]
      <n>[<m>|<cm>|<um>|<nm>]
      <n>[<m>|<cm>|<um>|<nm>]}]
    [min= {
      <n>[<m>|<cm>|<um>|<nm>]
      <n>[<m>|<cm>|<um>|<nm>]
      <n>[<m>|<cm>|<um>|<nm>]}]
    [mesh.align] [remesh] |

  flip [Adaptive] [location=<n>] [<m>|<cm>|<um>|<nm>] |

  reflect [Adaptive] [keep.original]
    (left | right | front | back) | (ymin | ymax | zmin | zmax) |

  rotate [Adaptive] [angle=<n> axis= "X" | "Y" | "Z"] |

  stretch [Adaptive] (left | right | front | back)
    [length=<n>] [<m>|<cm>|<um>|<nm>]
    [location=<n>] [<m>|<cm>|<um>|<nm>] [remesh] |

  translate= {
    <n>[<m>|<cm>|<um>|<nm>]
    <n>[<m>|<cm>|<um>|<nm>]
    <n>[<m>|<cm>|<um>|<nm>]}

  [masks]
```

Arguments: Cut

Adaptive

If specified, `Adaptive` switches on adaptive meshing if `remesh` is given. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

A: Commands

transform

cut

Selects the cut operation. Crops a structure using either the `max` and `min` arguments, or the `location` argument.

`location` (`left` | `right` | `front` | `back` | `up` | `down`)

Specifies the coordinate where the cut will be made. In addition, you must select one of `left`, `right`, `front`, `back`, `up`, or `down` to specify an axis-aligned cut at the coordinate given by `location`. Default value and unit: `0.0 μm`.

NOTE You cannot use `location` with the `max` and `min` arguments.

`max`, `min`

Use these two arguments to specify the new bounding box:

`max= {<maxx> <maxy> <maxz>} min= {<minx> <miny> <minz>}`

Default unit: `μm`.

NOTE You cannot use `max` and `min` with the `location` argument.

`mesh.align`

By default, MGOALS cuts the structure at the nearest mesh line and does not perform a remesh. If `!mesh.align` is specified, MGOALS cuts precisely at the specified coordinates and remeshes the structure.

`remesh`

Available only for two dimensions. Forces a remesh after the transformation. However, remeshing is always possible using the `grid remesh` command if required.

Arguments: Flip

`Adaptive`

If specified, `Adaptive` switches on adaptive meshing. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

`flip`

Selects the flip operation (flips from top to bottom). See [Flip and Backside Processing on page 263](#).

location

Selects the x-coordinate about which the structure will be flipped. By default, the middle of the structure is chosen. Subsequent `transform flip` commands will, by default, use the same location for flipping whether the default is used or a chosen `location` is used. In three dimensions, the z-coordinate of the rotation is the middle of the structure in the z-direction. The location of the flip is also the fixed coordinate for mechanics simulations, which is otherwise at the bottom of the structure when no flip has occurred.

Arguments: Reflection

Adaptive

If specified, `Adaptive` switches on adaptive meshing. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

keep.original

Retains the original structure after reflection (having both the original and the reflected structure), or stores only the reflected structure when disabled with `!keep.original`.
Default: true.

left, right, front, back

Selects the side of the simulation domain at which the reflection is performed.

reflect

Selects the reflection operation.

ymin, ymax, zmin, zmax

Specifies the location where the reflection is performed:

- `ymin` is the same as `left`.
- `ymax` is the same as `right`.
- `zmin` is the same as `back`.
- `zmax` is the same as `front`.

Arguments: Rotation

Adaptive

If specified, `Adaptive` switches on adaptive meshing. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

A: Commands

transform

angle

Rotation angle. It must be one of 90, 180, or 270. Angles leading to structures having gas in a side are not allowed. This means that for `axis= "Y"` and `axis= "Z"`, only 180 is allowed. For `axis= "X"`, 90, 180, or 270 are allowed. Two-dimensional simulations are extruded into three dimensions and are then rotated.

axis

Rotation axis. It must be the x-axis, y-axis, or z-axis.

rotate

Specifies the rotation operation.

Arguments: Stretch

Adaptive

If specified, `Adaptive` switches on adaptive meshing. Parameters for adaptive meshing are described in [Adaptive Mesh Refinement on page 177](#). The default is the return value of `pdbGet Grid Adaptive`.

left, right, front, back

Indicates which side of the structure will be moved.

length

Length of stretching. Default value and unit: 0 μm .

location

The y- or z-coordinate where the structure will be stretched. Default value and unit: 0 μm .

remesh

Specifies that a remesh will be performed. Default: true.

stretch

Specifies the stretch operation.

Arguments: Translation

translate

Translates the entire structure by specifying a translation vector:

```
translate= {<translate_x> <translate_y> <translate_z>}
```

Default unit: μm .

Arguments: Transform Masks

masks

If this option is specified, masks will be transformed as well if present. If the `keep.original` option is specified with the `reflect` option, the reflected mask and the original mask will be combined into one mask. Otherwise, the reflected masks will replace the original ones. Default: false.

Description

Previously, the `cut` and `clip` commands had slightly different behavior. Now, they are identical and are referred to as the `cut` command. All these transformations work in both two and three dimensions.

Examples

Both commands reflect the structure and refinement boxes to the right side:

```
transform reflect right
transform reflect ymax
```

Reflect the structure and refinement boxes to the right side and keep the reflected part only:

```
transform reflect right !keep.original
```

Stretch the structure and refinement boxes to the right side. The structure and refinement boxes left of 0.7 remain unchanged; the structure and refinement boxes to the right of 0.7 will be moved by 20 nm:

```
transform stretch location= 0.7 length= 0.02 right remesh
```

Cut the structure and refinement boxes at $y=0.7$. The left part will be removed without remeshing:

```
transform cut location= 0.7 left !remesh
```

Cut the structure and refinement boxes at x between $0\ \mu\text{m}$ and $1\ \mu\text{m}$, and y between $0\ \mu\text{m}$ and $3\ \mu\text{m}$:

```
transform cut min= {0<um> 0<um>} max= {1<um> 3<um>}
```

Shift the structure and refinement boxes up in the x -direction by $1\ \mu\text{m}$:

```
transform translate= {-1 0 0}
```

A: Commands

transform

Flip the structure and refinement boxes from top to bottom about their midpoint if it is the first flip, or store the flip location for subsequent flips:

```
transform flip
```

Rotate the structure and refinement boxes 90° in the x-axis:

```
transform rotate axis= "X" angle= 90
```

See Also

[Handling Stress in Etching and Deposition on page 236](#)
[mgoals on page 510](#)

transform.mask

Cuts, flips, reflects, rotates, stretches, or translates a given mask or all masks.

Syntax

```
transform.mask
  (cut | flip | reflect | rotate | stretch)
  [angle=<n> axis= "X" | "Y" | "Z"]
  [keep.original]
  [length=<n>] [<m>|<cm>|<um>|<nm>]
  [name=<c>] [name.new=<c>]
  [[location=<n>] [<m>|<cm>|<um>|<nm>]
    (left | right | front | back) | (ymin | ymax | zmin | zmax)]
  [max= {
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>] }]
  [min= {
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>] }]
  [translate=
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>] ]]
```

Arguments: General

keep.original

Specifies whether to keep the original mask after a reflection operation. This option only works with the `reflect` option. The reflected mask and the original mask will be combined into one mask if specified. Default: true.

name

Name of the mask to apply the transformation.

name.new

Name of the transformed mask. If not specified, the transformed mask will replace the original one. If specified, a new mask will be created, and the original one will be retained.

Arguments: Cut

cut

Crops the mask to a new bounding box (using `max` and `min`), or crops half of the mask (using either the `left`, `right`, `front`, or `back` option).

`left`, `right`, `front`, `back`

Indicates a cut at a location given by `location`, and specifies which half to remove. These options must not be used with `max` and `min`.

`location`

Specifies the y- or z-coordinate where the cut is to be performed. The `location` argument is used with `left`, `right`, `front`, or `back` to indicate which direction and side to cut. Default value and unit: 0.0 μm .

`max`, `min`

The cut box can be specified by either:

- Both the `max` and `min` arguments:
`max= {maxx maxy maxz} min= {minx miny minz}`
- One of `left`, `right`, `front`, or `back` to specify an axis-aligned cut at the coordinate given by the `location` argument.

Default unit: μm .

NOTE The `max` and `min` arguments must be used together. Do not use them with any of the `left`, `right`, `front`, or `back` options.

Arguments: Flip

flip

Flips a mask (top to bottom).

`location`

Selects the x-coordinate about which the mask will be flipped. By default, the middle of the structure is chosen. Subsequent `transform flip` commands will, by default, use the same location for flipping whether the default is used or a chosen `location` is used. In three dimensions, the z-coordinate of the rotation is the middle of the structure in the z-direction.

Arguments: Reflection

left, right, front, back

Selects the side of the simulation domain at which the reflection is performed.

reflect

Indicates that a reflection will be performed.

ymin, ymax, zmin, zmax

Specify the location where the mask reflection is performed:

- ymin is the same as left.
- ymax is the same as right.
- zmin is the same as back.
- zmax is the same as front.

Arguments: Rotation

angle

Rotation angle.

axis

Rotation axis. It must be x, y, or z. If the rotation axis is y or z, the rotation angle must be 180.

rotate

Indicates that a rotation will be performed.

Arguments: Stretch

left, right, front, back

Indicates which side of the mask will be moved.

length

Length of stretching. Default value and unit: 0 μm .

location

Specifies the y- or z-coordinate where the mask will be stretched. Default value and unit: 0 μm .

A: Commands

transform.mask

stretch

Indicates that a stretch operation will be performed.

Arguments: Translation

translate

Translates the mask by specifying a translation vector:

```
translate= {translate_x translate_y translate_z}
```

Default unit: μm .

Description

A transformed mask is created and replaces the old one by default. The transformation applies to all existing masks, except if `name` is specified. In this case, a transformed mask name also can be specified using `name.new`. If `name.new` is specified, the original mask will be retained, and a new one will be created. The `keep.original` option can be used with `reflect` to combine the reflected mask and the original mask into one mask.

Examples

Reflect all the current masks to the right side:

```
transform.mask reflect right
```

Create a new mask called `smask1` by stretching the existing mask `mask1` to the right side. The area left of 0.7 remains unchanged; the structure to the right of 0.7 will be moved by 20 nm:

```
transform.mask stretch location= 0.7 length= 0.02 right name= mask1 \  
name.new= smask1
```

Cut all the existing masks at $y = 0.7$. The left part will be removed:

```
transform.mask cut location= 0.7
```

Create a new mask named `new2` by copying and shifting `mask2` up in the x-direction by $1 \mu\text{m}$:

```
transform.mask translate= {-1 0 0} name= mask2 name.new= new2
```

Rotate the mask named `mask3` 180° around the y-axis without changing its name:

```
transform.mask rotate axis= "Y" angle= 180 name= mask3
```

transform.refinement

Reflects, stretches, cuts, flips, rotates, or translates a given refinement box or all refinement boxes.

Syntax

```
transform.refinement
  (cut | flip | reflect | rotate | stretch)
  [angle=<n> axis= "X" | "Y" | "Z"]
  [keep.original]
  [length=<n>] [<m>|<cm>|<um>|<nm>]
  [name=<c>] [name.new=<c>]
  [[location=<n>] [<m>|<cm>|<um>|<nm>]
    (left | right | front | back | up | down) | (ymin | ymax | zmin | zmax)]
  [max= {
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>] }
  min= {
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>] }
  [translate=
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>]
    <n> [<m>|<cm>|<um>|<nm>] }]
```

Arguments: General

keep.original

Specifies whether to keep the original after the transformation. When keeping the original refinement, the original is untouched, and a new one is created by transforming the original refinement. Otherwise, the refinement itself is transformed.

name

Name of the refinement to apply the transformation.

name.new

Name of the transformed refinement. If not specified, a default name is given.

Arguments: Cut

cut

Crops the refinement to a new bounding box (using `max` and `min`) or crops half of it (using the `left`, `right`, `front`, `back`, `up`, or `down` arguments).

`left`, `right`, `front`, `back`, `up`, `down`

Indicates a cut at a location given by `location`, and specifies which half is to be removed. These arguments must not be used with `max` and `min`.

`location`

Specifies the x-, y-, or z-coordinate where the cut is to be performed. The `location` argument is used with `left`, `right`, `front`, `back`, `up`, or `down` to indicate which direction and side to cut. Default: 0.0 μm .

`max`, `min`

The cut box can be specified by either:

- Both the `max` and `min` arguments:
`max= {maxx maxy maxz} min= {minx miny minz}`
- One of `left`, `right`, `front`, `back`, `up`, or `down` to specify an axis-aligned cut at the coordinate given by the `location` argument.

Default unit: μm .

NOTE The `max` and `min` arguments must be used together. Do not use them with any of the `left`, `right`, `front`, `back`, `up`, or `down` options.

Arguments: Flip

flip

Flips a refinement (top to bottom).

`location`

Selects the x-coordinate about which the refinement will be flipped. By default, the middle of the structure is chosen. Subsequent `transform flip` commands will, by default, use the same `location` for flipping whether the default is used or a chosen `location` is used. In three dimensions, the z-coordinate of the rotation is the middle of the structure in the z-direction.

Arguments: Reflection

left, right, front, back

Selects the side of the simulation domain at which the reflection is performed.

reflect

Indicates that a reflection will be performed.

ymin, ymax, zmin, zmax

Specify the location where the refinement reflection is performed:

- ymin is the same as left.
- ymax is the same as right.
- zmin is the same as back.
- zmax is the same as front.

Arguments: Rotation

angle

Rotation angle. It must be 90, 180, or 270. Default unit: degree.

axis

Rotation axis. It must be x, y, or z.

rotate

Indicates that a rotation will be performed.

Arguments: Stretch

left, right, front, back

Indicates which side of the refinement will be moved.

length

Length of stretching. Default value and unit: 0 μm .

location

Specifies the y- or z-coordinate where the refinement will be stretched. Default value and unit: 0 μm .

A: Commands

transform.refinement

stretch

Indicates that a stretch operation will be performed.

Arguments: Translation

translate

Translates the refinement by specifying a translation vector:

```
translate= {translate_x translate_y translate_z}
```

Default unit: μm .

Description

A new transformed refinement box is created by default, while the old one is kept. This can be overridden with `!keep.original`. The transformation applies to all existing refinements, except if a name is specified. In this case, a transformed refinement name also can be specified by using `name.new`.

Examples

Create a set of new refinements as reflections of all the current refinements to the right side:

```
transform.refinement reflect right
```

Create a new refinement called `sbox1` by stretching the existing refinement `box1` to the right side. The area left of 0.7 remains unchanged; the structure to the right of 0.7 will be moved by 20 nm:

```
transform.refinement stretch location= 0.7 length= 0.02 right name= box1 \  
name.new= sbox1
```

Cut all the existing refinements at $y = 0.7$. The left part will be removed:

```
transform.refinement cut location= 0.7 !keep.original
```

Create a new refinement `new2` by copying and shifting `r1` up in the x-direction by $1 \mu\text{m}$:

```
transform.refinement translate= {-1 0 0} name= r1 name.new= new2
```

Rotate the refinement `refbox` 270° around the y-axis without changing its name:

```
transform.refinement rotate axis= "Y" angle= 270 name= refbox \  
name.new= refbox !keep.original
```


See Also

[Mesh Refinement on page 170](#)
[Handling Stress in Etching and Deposition on page 236](#)
[refinebox on page 591](#)
[transform on page 661](#)

translate

Translates a named dataset with the specified offset.

Syntax

```
translate
  [<material>]
  [min=<n>]
  [name=<c>]
  [offset= {<n> <n> <n>}]
```

Arguments

<material>

If a material is specified, the dataset is translated in the specified material only. Otherwise, the dataset is translated in all materials. See [Specifying Materials on page 18](#).

min

Minimum value to fill the points with undefined value. Default: 0.0.

name

Name of a dataset. Default: `Z_Plot_Var`.

offset

List of numeric values, where the first, second, and third values in the list are taken as the x-, y-, and z-value, respectively. The missing value is treated as zero.

Description

This command spatially shifts a profile (dataset) with the specified offset. If a material is specified, the profile is shifted in the specified material only. Otherwise, the profile is shifted in all materials. When a profile is shifted, the value at some points may become undefined, in which case, these points are filled with a minimum value as specified by the `min` argument.

Examples

Shift the Boron data field with a shifting vector (0.01 μm , 0.02 μm , 0.0 μm):

```
translate name= Boron offset= {0.01 0.02}
```

update_hoop_radial_stress

Calculates the hoop and radial stresses with an axis of symmetry in the x-direction.

Syntax

```
update_hoop_radial_stress [<n> <n>]
```

Arguments

<n> <n>

The first value is the y-center and second value is the z-center for the calculation. Default value and unit: 0 μm .

Description

This command calculates the hoop stress and the radial stress from a user-specified center point. The fields are saved under `Stress_Theta` and `Stress_R`, respectively. The calculations are valid when there is rotational symmetry.

Examples

Calculate hoop and radial stresses with an axis of symmetry in the x-direction at $y = 0.2 \mu\text{m}$ and $z = 0.1 \mu\text{m}$:

```
update_hoop_radial_stress 0.2 0.1
```

A: Commands

update_principal_strain

update_principal_strain

Calculates the principal strains.

Syntax

```
update_principal_strain
```

Description

This command calculates the first, second, and third principal strains, and stores them as the fields `PrincipalStrain1`, `PrincipalStrain2`, and `PrincipalStrain3`, respectively.

update_principal_stress

Calculates the principal stresses.

Syntax

```
update_principal_stress
```

Description

This command calculates the first, second, and third principal stresses, and stores them as the fields `PrincipalStress1`, `PrincipalStress2`, and `PrincipalStress3`, respectively.

voltage_ramp

Defines a voltage profile for use with the `supply` command.

Syntax

```
voltage_ramp
  (clear | list | name=<c>)
  [current.time=<n>] [<hr>|<min>|<s>]
  [delNV=<n>[<V>] | delV=<n>[<V>]]
  [hold] [last]
  [ramprate=<n>] [<V/s>|<V/min>]
  [reset.init.time]
  [time=<n>] [<hr>|<min>|<s>]
  [v.final=<n>] [<V>]
  [voltage=<n>] [<V>]
```

Arguments

`clear`

Deletes the global list of voltage ramps. When defining profiles, the action is to unite the new definition with any prior profiles of the same name.

`current.time`

Returns the value of the ramp for the given time. Default unit: minute.

`delNV`

Defines the maximum voltage step during a voltage ramp-down if specified.

`delV`

Defines the maximum voltage step during a voltage ramp-up if specified.

`hold`

During this segment, allows the `solve` command the opportunity to specify the time of the segment.

`last`

Defines the final component of the voltage profile. There will be no more additions to the ramp.

A: Commands

voltage_ramp

list

Generates a list of voltage profiles. It returns a Tcl list and can be operated on as such. The default action for commands is to print the return. If no handling is required, this prints a list of names of defined voltage profiles. If a name is specified, `voltage_ramp` only is listed along with details about the ramps.

name

Name of the parameter used to identify the voltage ramp and specify it in a supply command.

ramprate

Voltage change during simulation. Default value and unit: 0 V/s.

reset.init.time

Starts each voltage step with the same initial time step.

time

Simulation time. Default unit: minute.

v.final

Final voltage for a voltage ramp-up or ramp-down. It is used if `ramprate` is not given. The ramp time is calculated automatically.

voltage

Simulation voltage.

Description

This command specifies multiple voltage ramps and holds. It can be used to construct a complex voltage sequence to be simulated with the `solve` command.

Examples

Define the voltage profile named `vr1` with a voltage rate of 0.5 V/s:

```
voltage_ramp name= vr1 voltage= 0.1 ramprate= 0.5<V/s> time= 1<s>
```

See Also

[supply on page 647](#)

WritePlx

Writes a 1D .plx file.

Syntax

```
WritePlx
  <filename>
  [<circuit_instances> | <contact_list>]
  [current | resistance | voltage]
  [include.interfaces]
  [<material>]
  [only.interfaces]
  [x=<n>] [y=<n>] [z=<n>]
```

Arguments

<circuit_instances>, <contact_list>

List of circuit instances or contact names for which the data will be written to the file. The list must be a pair of contacts for the `resistance` option. The circuit instances must be a pair of a circuit element and a circuit node for the `current` option if `current` is being used for the circuit.

current, resistance, voltage

Specifies the data type to be written to the file (see [Storing Time Versus Resistance, Current, and Voltage on page 327](#)).

<filename>

Name of the output file.

NOTE This must be the first argument on the `WritePlx` command line.

include.interfaces

Includes interface values with the returned data. At an interface, the distance coordinate of the three nodes (two bulk and one interface) will be the same, and the interface value will be inserted between the two neighboring bulk values.

<material>

If a material is specified, only the plot from the given material is created. See [Specifying Materials on page 18](#).

A: Commands

WritePlx

only.interfaces

Returns interface values exclusively in the returned data. When specified, no bulk values are returned.

x, y, z

Specify the cut position. For 1D simulations, no cut specification is necessary. For 2D simulations, either x or y must be specified. For 3D simulations, two axes must be specified. It is also possible to shift .plx output files by specifying PlxShift variables. Default unit: μm .

Description

This command makes a 1D profile along a given outline and writes a .plx file of the solutions and terms given in the list provided by the SetPlxList command. If the list is not provided, only present solution names are written. If a material is specified, only data from the given material is used to create the plot.

NOTE The <filename> argument must be the first argument on the WritePlx command line.

Examples

Write a 1D .plx file at the $y = 1.5 \mu\text{m}$ outline:

```
WritePlx 1.5.plx y= 1.5
```

Shift the axis by $0.2 \mu\text{m}$ and write a 1D .plx file:

```
set PlxShift 0.2
WritePlx test.plx
```

See Also

[SetPlxList on page 619](#)

APPENDIX B Resistance, Capacitance, and Thermal Resistance

This appendix describes the resistance, capacitance, and thermal resistance analyses of Sentaurus Interconnect.

Overview

There are different simulation categories in Sentaurus Interconnect depending on the type of analysis:

- Electrical current analysis extracts the resistance matrix (see [Chapter 6 on page 131](#)).
- Electrostatic analysis extracts the capacitance matrix (see [Chapter 7 on page 143](#)).
- Thermal analysis extracts the thermal resistance matrix (see [Chapter 5 on page 121](#)).
- Thermal RC analysis calculates the thermal RC networks (see [Chapter 5 on page 121](#)).

Resistance, capacitance, and thermal resistance analyses are mutually exclusive, that is, only one type of analysis can be performed at a time. One exception is that resistance and capacitance analyses can be performed together for a distributed RC extraction simulation.

Electrical Current Analysis: Resistance Matrix

Sentaurus Interconnect calculates the resistance matrix for the listed contacts by solving the Laplace equation ([Eq. 83, p. 131](#)) without the displacement currents. If no contact information is given with the `mode` command, all user-defined contacts are used in the analysis. The resistances are calculated using:

$$I = G\Delta V \quad (128)$$

where:

- I is the total current.
- G is the conductance.
- ΔV is the voltage difference at the contact with respect to the reference point.

Since ΔV is set to 1 V, the absolute values of total current correspond to the conductance values. The inverse value of conductance corresponds to resistance. The absolute value of the

B: Resistance, Capacitance, and Thermal Resistance

Electrical Current Analysis: Resistance Matrix

diagonal terms in the matrix always represents the total conductance of the corresponding contact with respect to all other contacts (as if only that contact is biased and all others are grounded).

The absolute value of the off-diagonal terms represents the conductances between both contacts (for currents, the signs change). [Figure 104](#) represents a typical structure where resistances may be required.

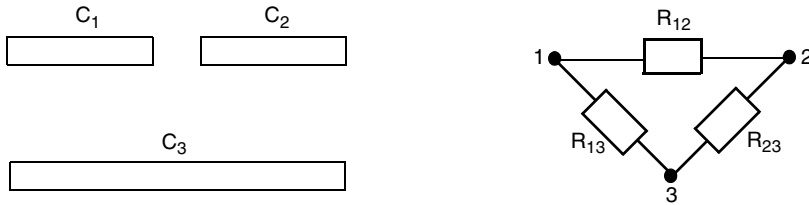


Figure 104 (Left) Structure formed by two conductors parallel to a plane and (right) corresponding schematic for the resistance matrix simulation

The mathematical relationship between the resistances expressed in the schematic and the corresponding short-circuit resistances calculated by Sentaurus Interconnect is expressed as:

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \begin{bmatrix} G_{11} & G_{12} & G_{13} \\ G_{21} & G_{22} & G_{23} \\ G_{31} & G_{32} & G_{33} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \begin{bmatrix} |I_{11}| & |I_{12}| & |I_{13}| \\ |I_{21}| & |I_{22}| & |I_{23}| \\ |I_{31}| & |I_{32}| & |I_{33}| \end{bmatrix} \quad (129)$$

$$\begin{aligned} |I_{13}| &= |I_{11} + I_{12}| & I_{13} &= I_{31} & G_{11} &= |I_{11}| & G_{12} = G_{21} &= |I_{12}| \\ |I_{23}| &= |I_{22} + I_{21}| & I_{23} &= I_{32} & G_{22} &= |I_{22}| & G_{23} = G_{32} &= |I_{23}| \\ |I_{12}| &= |I_{13} + I_{11}| & I_{12} &= I_{21} & G_{33} &= |I_{33}| & G_{31} = G_{13} &= |I_{31}| \end{aligned} \quad (130)$$

$$\begin{bmatrix} C_1 \\ C_1 \\ C_1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_1 \\ C_1 \end{bmatrix} \begin{bmatrix} G_{11}^{-1} & G_{12}^{-1} & G_{13}^{-1} \\ G_{21}^{-1} & G_{22}^{-1} & G_{23}^{-1} \\ G_{31}^{-1} & G_{32}^{-1} & G_{33}^{-1} \end{bmatrix} \quad (131)$$

$$\begin{aligned} R_{11} &= G_{11}^{-1} & R_{12} &= R_{21} = G_{12}^{-1} \\ R_{22} &= G_{22}^{-1} & R_{23} &= R_{32} = G_{23}^{-1} \\ R_{33} &= G_{33}^{-1} & R_{31} &= R_{13} = G_{31}^{-1} \end{aligned} \quad (132)$$

The matrix should be symmetric. Due to regriding or the solver tolerance, the final results may not be exactly symmetric.

To calculate the resistance matrix, use the `mode` command (see [mode on page 518](#)), for example:

```
mode resistance
mode resistance= { C1 C2 }
```

The first command calculates the resistance matrix between user-defined contacts in the structure. The second command calculates the resistance matrix between the contacts C1 and C2, which are user defined. All other unused contacts are grounded and lumped into one contact called `GROUND_RCX`. When the simulation is completed, the resistance matrix is printed in the log file. To print the resistance matrix, use the `printResistanceMatrix` command (see [printResistanceMatrix on page 577](#)).

Electrostatic Analysis: Capacitance Matrix

Sentaurus Interconnect calculates the capacitance matrix for the listed contacts by solving the Poisson equation ([Eq. 94, p. 143](#)) with zero charge density in the regions. If no contact information is given, conductors are treated as contacts. The short-circuit capacitances are calculated using:

$$Q = C\Delta V \quad (133)$$

where:

- Q is the total charge.
- C is the capacitance.
- ΔV is the voltage difference at the contact with respect to the reference point.

NOTE The Poisson equation ([Eq. 94, p. 143](#)) is not solved in the gas mesh.

Since ΔV is set to 1 V, the absolute values of total charges correspond to the capacitance values. The absolute value of the diagonal terms in the matrix always represents the total capacitance of the corresponding contact with respect to all other contacts (as if only that contact is biased and all others are grounded).

B: Resistance, Capacitance, and Thermal Resistance

Electrostatic Analysis: Capacitance Matrix

The absolute value of the off-diagonal terms represents the capacitances between both contacts (for charges, the signs change). [Figure 105](#) represents a typical structure where capacitances may be required.

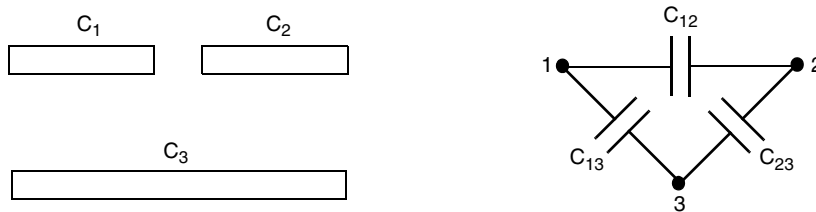


Figure 105 (Left) Structure formed by two conductors parallel to a plane and (right) corresponding schematic for the capacitance matrix simulation

The mathematical relationship between the capacitances expressed in the schematic and the corresponding short-circuit capacitances calculated by Sentaurus Interconnect is expressed as:

$$\begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \begin{bmatrix} |Q_{11}| & |Q_{12}| & |Q_{13}| \\ |Q_{21}| & |Q_{22}| & |Q_{23}| \\ |Q_{31}| & |Q_{32}| & |Q_{33}| \end{bmatrix} \quad (134)$$

$$\begin{aligned} |Q_{13}| &= |Q_{11} + Q_{12}| & Q_{13} &= Q_{31} & C_{11} &= |Q_{11}| & C_{12} &= C_{21} = |Q_{12}| \\ |Q_{23}| &= |Q_{22} + Q_{21}| & Q_{23} &= Q_{32} & C_{22} &= |Q_{22}| & C_{23} &= C_{32} = |Q_{23}| \\ |Q_{12}| &= |Q_{13} + Q_{33}| & Q_{12} &= Q_{21} & C_{33} &= |Q_{33}| & C_{31} &= C_{13} = |Q_{31}| \end{aligned} \quad (135)$$

The matrix should be symmetric. Due to regridding or the solver tolerance, the final results may not be exactly symmetric.

To calculate the capacitance matrix, use the `mode` command (see [mode on page 518](#)), for example:

```
mode capacitance
mode capacitance= { C1 C2 }
```

The first command calculates the capacitance matrix between conducting layers in the structure. User-defined contacts are ignored. The second command calculates the capacitance matrix between the contacts C1 and C2, which are user defined. All other unused contacts are grounded and lumped into one contact called `GROUND_RCX`. When the simulation is completed, the capacitance matrix is printed in the log file. To print the capacitance matrix, use the `printCapacitanceMatrix` command (see [printCapacitanceMatrix on page 573](#)).

Floating Conductors

Floating contacts or conductors refer to contacts or conductors with either charge boundary conditions or charge density-type boundary conditions specified on them (see [Applying Boundary Conditions to Electrodes on page 145](#)). When performing electrostatic analysis, floating conductors are omitted from the capacitance matrix, and the analysis is performed assuming 0 charge supply on the conductor.

For example, in the previous example, if the charge supply was specified on C2:

```
supply contact.name = C2 charge=1e-18<Coulomb>
```

the capacitance matrix would be:

$$\begin{bmatrix} C_{11} & C_{13} \\ C_{31} & C_{33} \end{bmatrix} = \begin{bmatrix} |Q_{11}| & |Q_{13}| \\ |Q_{31}| & |Q_{33}| \end{bmatrix} \quad (136)$$

with the magnitude of charge on C2 ignored and set to zero during the analysis.

Merging Contacts

Contacts, when defined, can be merged before performing an electrostatic or a capacitance calculation. Merged contacts behave as one and, therefore, `supply` should be defined on only one of them.

When merging contacts, either conducting regions or user-defined contacts should be merged, but they must not be mixed. For example, if contacts are defined as:

```
contact region=AA name=EA  
contact region=BB name=EB  
contact region=CC name=EC
```

the allowed merge syntaxes are:

```
contact merge= {AA CC}  
contact merge= {EA EC}
```

If only `contact merge` is specified, regions that are physically in contact with each other are merged.

B: Resistance, Capacitance, and Thermal Resistance

Distributed RC Extraction

NOTE When `mode capacitance` is used with `contact merge`, ensure that they both specify either regions or user-defined contacts. For example:

```
mode capacitance = {AA BB CC}
contact merge = {AA BB}
```

and:

```
mode capacitance = {EA EB EC}
contact merge = {EA EB}
```

are allowed. In addition, the default capacitance mode can only be used when merging conducting regions:

```
mode capacitance
contact merge = {AA BB}
```

NOTE Multiple `contact merge` commands can be used within a given simulation, that is:

```
contact merge = {AA BB}
contact merge = {CC DD}
```

are the same as:

```
contact merge = {"AA BB" "CC DD"}
```

To clear previous merge information before a `contact merge` command, you must call `contact merge clear`.

Distributed RC Extraction

Sentaurus Interconnect structures can be segmented automatically for distributed resistance analysis. Structures are segmented automatically, placing contacts at conductor–conductor interfaces. Distributed resistance analysis also takes into account user-defined contacts. To switch on automatic structure segmentation and contact placement, specify the option `auto.segment` with the option `resistance` in the `mode` command, for example:

```
mode resistance auto.segment
```

You can also create contacts automatically at the outer boundary of conductor regions using the option `boundary.contacts` of the `option` command (see [Figure 106 on page 689](#)), for example:

```
option boundary.contacts
```

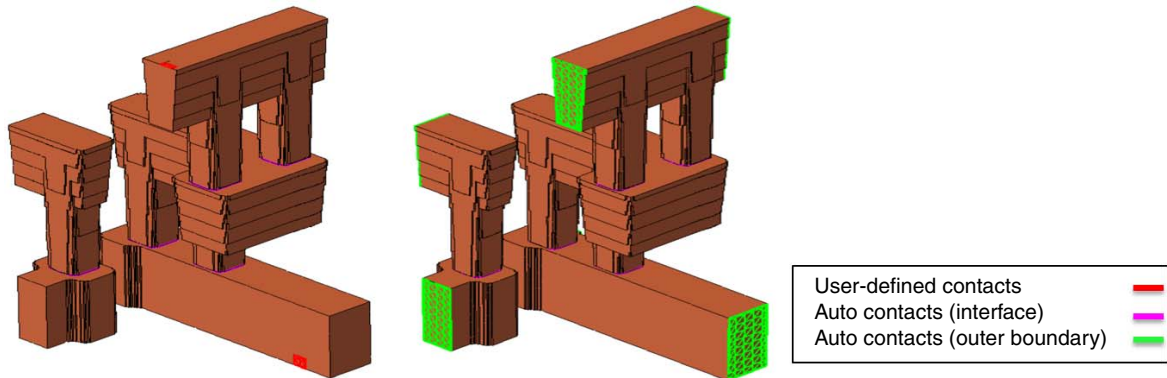


Figure 106 (Left) Contact placement with user-defined contacts and (right) automatic contact placement with outer boundary contacts

In many cases, it is useful to lump several materials, such as a metal with its liner or filler materials, into one material to simplify the distributed RC extraction. For this reason, you can use the `lumped.materials` argument of the `option` command to lump materials (see Figure 107), for example:

```
option lumped.materials= {"Cu1 TaN1" "Cu2 TaN2" "Cu3 TaN3"}
```

In each list, all materials will be lumped into the first material. In this example, `TaN1` will be lumped into `Cu1`, `TaN2` will be lumped into `Cu2`, and `TaN3` will be lumped into `Cu3`.

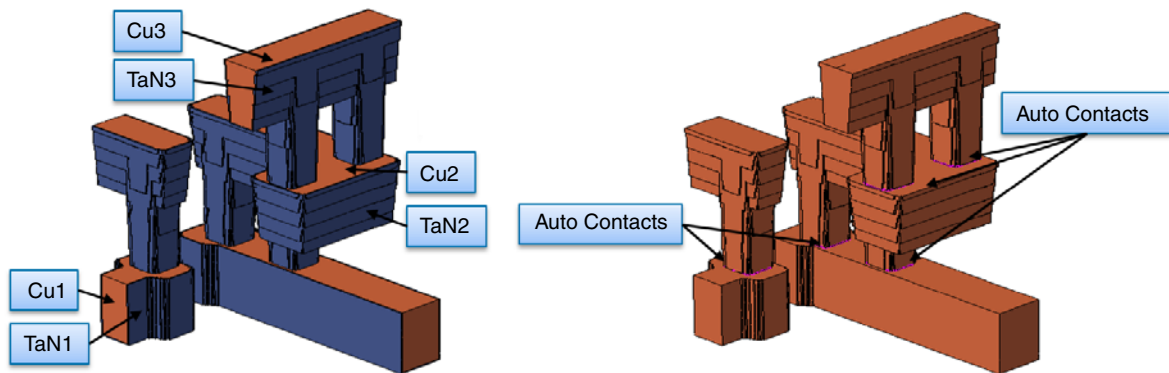


Figure 107 (Left) Before automatic segmentation and contact placement and (right) after automatic segmentation and contact placement

When `auto.segment` is specified, capacitance analysis can be performed with resistance analysis as follows:

```
mode resistance capacitance auto.segment
```

B: Resistance, Capacitance, and Thermal Resistance

Distributed RC Extraction

With or without automatic structure segmentation, capacitance analysis is distributed to the active conductor regions. Sentaurus Interconnect calculates capacitance using the total surface charge on each active conductor region.

At the end of the simulation, the resistance matrix and capacitance matrix are calculated independently. You can print a linked resistance and capacitance netlist with the `printRC` command (see [printRC on page 575](#)), for example:

```
printRC
printRC filename.spi
```

In the linked RC netlist, resistance for open circuits and capacitance for short circuits are discarded. Furthermore, any resistance values greater than $1e5 \Omega$ are excluded. You can set and change this cut-off value with:

```
pdbSet Solve R.Max.Limit <value>
```

NOTE When you set `R.Max.Limit` explicitly, `printResistanceMatrix` also uses the value to cut off resistance values during printing.

In addition, you can use `exclude.C` to exclude certain capacitances specified by material or region pairs, for example:

```
printRC exclude.C= {"Cu1 Cu3" "Region_1 Region_3"}
printRC exclude.C= {"Cu1"}
```

Capacitance between materials `Cu1` and `Cu3` and between regions `Region_1` and `Region_3` are excluded in the first command, and capacitance between material `Cu1` and any other materials are excluded in the second command.

To link capacitance and resistance netlists, the capacitance associated with a conductor region is further distributed to the contacts that are attached to the conductor region, using one of the following models (default is `model1`):

- `model1`: Capacitance fully divided per contact pairs
- `model2`: Capacitance partially divided per region with more contacts
- `model3`: Capacitance partially divided per region with fewer contacts
- `model4`: Capacitance lumped into one contact pair

For example:

```
printRC filename.spi model3
```

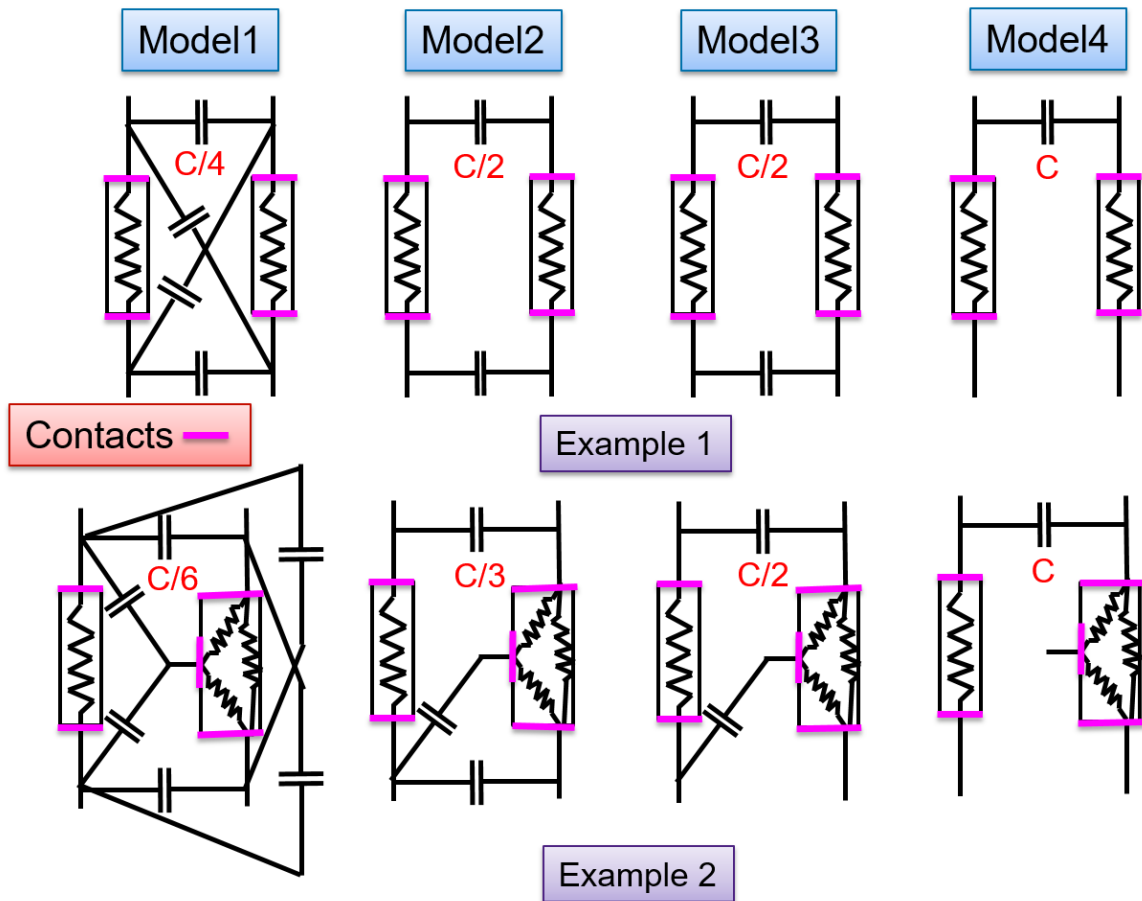



Figure 108 Examples of using the models to distribute capacitance and to link to the resistance netlist

Thermal Analysis: Thermal Resistance Matrix

Sentaurus Interconnect calculates the thermal resistance matrix for the listed contacts by solving the heat equation (Eq. 73, p. 121) with no time dependency and no distributed heat sources at ambient temperature, which is defined by the Parameter Database (PDB) parameter `Env.Temp` and is set to 26.844°C (300 K) by default. If no contact information is given with the mode command, all user-defined contacts are used in the analysis. The resistances are calculated using:

$$H = G\Delta T \quad (137)$$

B: Resistance, Capacitance, and Thermal Resistance

Thermal Analysis: Thermal Resistance Matrix

where:

- H is the total heat flow.
- G is the thermal conductance.
- ΔT is the temperature difference at the contact with respect to the reference point.

Since ΔT is set to 1 K, the absolute values of total heat correspond to the thermal conductance values. The inverse value of thermal conductance corresponds to thermal resistance. The absolute value of the diagonal terms in the matrix always represents the total thermal conductance of the corresponding contact with respect to all other contacts (as if only that contact is thermally biased and all others are thermally grounded).

The absolute value of the off-diagonal terms represents the thermal conductances between both contacts (for heat flows, the signs change). [Figure 109](#) represents a typical structure where resistances may be required.

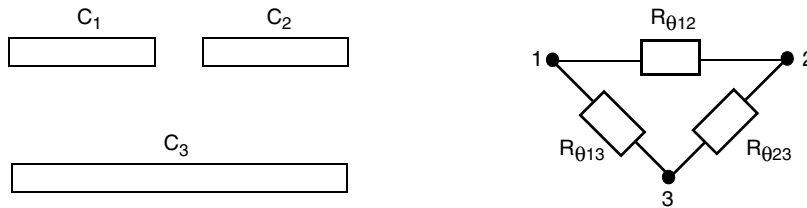


Figure 109 (Left) Structure formed by two conductors parallel to a plane and (right) corresponding schematic for the thermal resistance matrix simulation

The mathematical relationship between the thermal resistances expressed in the schematic and the corresponding short-circuit thermal resistances calculated by Sentaurus Interconnect is expressed as:

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \begin{bmatrix} G_{11} & G_{12} & G_{13} \\ G_{21} & G_{22} & G_{23} \\ G_{31} & G_{32} & G_{33} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \begin{bmatrix} |H_{11}| & |H_{12}| & |H_{13}| \\ |H_{21}| & |H_{22}| & |H_{23}| \\ |H_{31}| & |H_{32}| & |H_{33}| \end{bmatrix} \quad (138)$$

$$\begin{aligned} |H_{13}| &= |H_{11} + H_{12}| & H_{13} &= H_{31} & G_{11} &= |H_{11}| & G_{12} &= G_{21} &= |H_{12}| \\ |H_{23}| &= |H_{22} + H_{21}| & H_{23} &= H_{32} & G_{22} &= |H_{22}| & G_{23} &= G_{32} &= |H_{23}| \\ |H_{12}| &= |H_{13} + H_{11}| & H_{12} &= H_{21} & G_{33} &= |H_{33}| & G_{31} &= G_{13} &= |H_{31}| \end{aligned} \quad (139)$$

$$\begin{bmatrix} C_1 \\ C_1 \\ C_1 \end{bmatrix} \begin{bmatrix} R_{\theta 11} & R_{\theta 12} & R_{\theta 13} \\ R_{\theta 21} & R_{\theta 22} & R_{\theta 23} \\ R_{\theta 31} & R_{\theta 32} & R_{\theta 33} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_1 \\ C_1 \end{bmatrix} \begin{bmatrix} G_{11}^{-1} & G_{12}^{-1} & G_{13}^{-1} \\ G_{21}^{-1} & G_{22}^{-1} & G_{23}^{-1} \\ G_{31}^{-1} & G_{32}^{-1} & G_{33}^{-1} \end{bmatrix} \quad (140)$$

$$\begin{aligned} R_{\theta 11} &= G_{11}^{-1} & R_{\theta 12} &= R_{\theta 21} = G_{12}^{-1} \\ R_{\theta 22} &= G_{22}^{-1} & R_{\theta 23} &= R_{\theta 32} = G_{23}^{-1} \\ R_{\theta 33} &= G_{33}^{-1} & R_{\theta 31} &= R_{\theta 13} = G_{31}^{-1} \end{aligned} \quad (141)$$

The matrix should be symmetric. Due to regriding or the solver tolerance, the final results may not be exactly symmetric.

To calculate the thermal resistance matrix, use the `mode` command (see [mode on page 518](#)), for example:

```
mode thermal.resistance
mode thermal.resistance= { C1 C2 }
```

The first command calculates the thermal resistance matrix between user-defined contacts in the structure. The second command calculates the thermal resistance matrix between the contacts C1 and C2, which are user defined. All other unused contacts are grounded and lumped into one contact called `GROUND_RCX`. When the simulation is completed, the thermal resistance matrix is printed in the log file. To print the thermal resistance matrix, use the `printThermalResistanceMatrix` command (see [printThermalResistanceMatrix on page 581](#)).

Thermal Analysis: Thermal RC Network

Thermal RC networks are widely used for SPICE and mixed-mode device simulations to account for self-heating and interactive heating of electronic devices on a chip or in a package. Unlike thermal networks, which consist of thermal resistances only, thermal RC networks allow you to simulate the dynamic and transient thermal behavior of electronic devices, in particular, fast-switching power devices in a package.

A methodology for the automated extraction of such thermal RC networks has been developed and presented in publications [\[1\]\[2\]\[3\]](#).

Sentaurus Interconnect extracts thermal RC network elements using transient thermal analysis. Initially, the entire structure is set to the ambient temperature. Step-like power excitation of

B: Resistance, Capacitance, and Thermal Resistance

Thermal Analysis: Thermal RC Network

value $P_{\text{excitation}}$ (in W) is applied to a specified excitation contact. Some contacts (at least one) in the structure are maintained at a constant ambient temperature, and the remaining contacts are floating (that is, the heat flux through them is zero). The time-dependent thermal impedance of the response contact $\langle c_j \rangle Z_j(t)$ is computed using:

$$Z_j(t) = \frac{(T_j(t) - T_{\text{ambient}})}{P_{\text{excitation}}} \quad (142)$$

where $T_j(t)$ is the averaged temperature of contact $\langle c_j \rangle$ at time t , and T_{ambient} is the ambient contact temperature. The value of the excitation power influences the results when the system is nonlinear (for example, its materials have nonconstant thermal conductivities or specific heats).

The parameters of the thermal RC network are extracted for the Foster model using the built-in optimizer. The following model function is used to fit the thermal impedances:

$$Z_j(t) = \sum_{i=1}^n R_i \cdot \left(1 - \exp\left(\frac{-t}{R_i \cdot C_i}\right)\right) \quad (143)$$

where n is the rank (size) of the thermal RC network, and R_i and C_i are the values of the thermal resistance and the thermal capacitance, respectively.

To calculate the values of the elements of thermal RC networks, use the `mode` command (see [mode on page 518](#)), for example:

```
mode thermal.RC.network= {"C1" "C2"}
mode thermal.RC.network= {"C1=1.0e-4" "C2" "C3" "C4" }
```

The first command calculates the parameters of the thermal RC network between the excitation contact C1 and the only response contact C1 (only self-heating is considered), with the ambient contact C2 and the default excitation power of 1.0e-3 W.

The second command includes the computation of the floating contacts C3 and C4, which will be used as additional response contacts (interactive heating); therefore, three thermal RC networks will be computed at the same time. In addition, the excitation power has been set to 1.0e-4 W for this computation.

The thermal impedances are computed during the solution of the transient heat equation (Eq. 73, p. 121) with no distributed heat sources. To perform this, use the `solve` command (see [solve on page 629](#)), for example:

```
solve
solve time=1e-3<s> init=1e-9<s>
```

The first command solves the transient heat equation for the default time from 0 to 1 s, with the default time-step settings.

The second command changes the final time to 1e-3 s and the initial time step to 1e-9 s.

NOTE The final time and the time-stepping often require adjustments to better resolve the time-dependent thermal impedances in a particular structure.

The ambient temperature is taken from the PDB parameter `Env.Temp`, which is set to 26.844°C (that is, 300 K) by default.

To extract the parameters of the thermal RC network from the precomputed thermal impedances and to print the extracted parameters to the screen and to specified files, use the `printThermalRC` command (see [printThermalRC on page 579](#)). This command also can print the previously computed thermal impedances and the thermal RC network model to a specified `.plx` file for control purposes.

To change the thermal RC network size or the optimization parameters, such as the initial and bounding values of the thermal RC network parameters or the maximum number of iterations for optimization loops, use the `option` command before the `printThermalRC` command (see [option on page 526](#)). This might be necessary to achieve good optimization results.

Example: Electrostatic Analysis With Floating Conductor

The following self-contained example demonstrates electrostatic analysis (calculating the capacitance matrix) with a floating conductor on a simple structure containing a straight copper wire embedded in oxide between two metal planes (see [Figure 110](#)). The wire is the floating conductor with zero charge, and the planes are used to calculate the capacitance matrix.

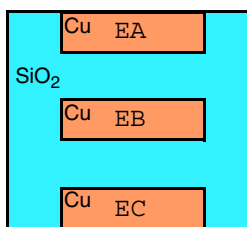


Figure 110 Copper wire embedded in oxide between two copper planes

```
# Define the structure
line clear
line x location= 0.0      spacing= 1.0<nm> tag= Bottom
line x location= 2.0<nm> spacing= 1.0<nm> tag= Mid1
line x location= 4.0<nm> spacing= 1.0<nm> tag= Mid2
line x location= 6.0<nm> spacing= 1.0<nm> tag= Mid3
```

B: Resistance, Capacitance, and Thermal Resistance

Example: Electrostatic Analysis With Floating Conductor

```
line x location= 8.0<nm> spacing= 1.0<nm> tag= Mid4
line x location= 10.0<nm> spacing= 1.0<nm> tag= Top

line y location= 0.0 spacing= 1.0<nm> tag= Left
line y location= 2.0<nm> spacing= 1.0<nm> tag= YMid1
line y location= 8.0<nm> spacing= 1.0<nm> tag= YMid2
line y location= 10.0<nm> spacing= 1.0<nm> tag= Right

region Copper name=AA xlo=Bottom xhi=Mid1 ylo=YMid1 yhi=YMid2
region Copper name=BB xlo=Mid2 xhi=Mid3 ylo=YMid1 yhi=YMid2
region Copper name=CC xlo=Mid4 xhi=Top ylo=YMid1 yhi=YMid2

region Oxide name=O1 xlo=Bottom xhi=Top ylo=Left yhi=YMid1
region Oxide name=O2 xlo=Mid1 xhi=Mid2 ylo=YMid1 yhi=YMid2
region Oxide name=O3 xlo=Mid3 xhi=Mid4 ylo=YMid1 yhi=YMid2
region Oxide name=O4 xlo=Bottom xhi=Top ylo=YMid2 yhi=Right

init !DelayFullD

# Define contacts
contact region=AA name=EA !replace
contact region=BB name=EB !replace
contact region=CC name=EC !replace

# Define the floating contact with 0 charge
supply contact.name=EB charge=0

# Specify capacitance matrix calculation
mode capacitance

# Start simulation
solve info=2

# Print the capacitance matrix and save results
printCapacitanceMatrix
struct tdr=cap-auto
```

The capacitance matrix output generated for this example is shown here (the floating wire conductor is not included in the output):

```
+-----+-----+-----+
| Capacitance (Farad) |         |         |
| Calculation (C) (V)=(Q) | AA      | CC      |
| AA                    | +7.5052e-17 | -7.5052e-17 |
| CC                    | -7.5052e-17 | +7.5052e-17 |
+-----+-----+-----+

+-----+-----+-----+
| Capacitance Matrix |         |         |
| (Farad)            | AA      | CC      |
| AA                  | 7.5052e-17 | 7.5052e-17 |
| CC                  | 7.5052e-17 | 7.5052e-17 |
+-----+-----+-----+
```

SPICE Models for Total Capacitance (F)			
C_0_0	AA	others	7.5052e-17
C_1_1	CC	others	7.5052e-17

SPICE Models for Entire Capacitance Matrix (F)			
C_0_1	AA	CC	7.5052e-17

To verify this result, the same simulation is performed without floating the wire. [Figure 111](#) shows the resulting circuit model with three capacitors. After floating the wire, these three capacitors can be reduced to a single capacitor from contact EA to contact EC, and the value of this equivalent capacitor is 0.07505 fF.

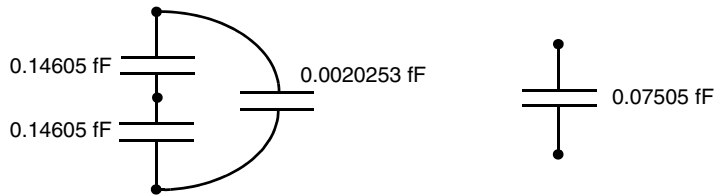


Figure 111 Circuit model of [Figure 110 on page 695](#) (left) before floating the wire and (right) after floating the wire

References

- [1] Y. Yu, T.-Y. T. Lee, and V. A. Chiriatic, “Compact Thermal Resistor-Capacitor-Network Approach to Predicting Transient Junction Temperatures of a Power Amplifier Module,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 2, no. 7, pp.1172–1181, 2012.
- [2] G. Greco *et al.*, “Generation of Electro-Thermal Models of Integrated Power Electronics Modules Using a Novel Synthesis Technique,” in *19th International Workshop on Thermal Investigations of ICs and Systems*, Berlin, Germany, pp. 216–221, September 2013.
- [3] D. Schweitzer, “Generation of Multisource Dynamic Compact Thermal Models by RC-Network Optimization,” in *29th Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, San Jose, CA, USA, pp. 116–121, March 2013.

B: Resistance, Capacitance, and Thermal Resistance
References