

Sentaurus™ Structure Editor User Guide

Version T-2022.03, March 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2022 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.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

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.

www.synopsys.com

Contents

Conventions	32
Customer Support	33
Acknowledgments	34
<hr/>	
1. Introduction to Sentaurus Structure Editor	35
Functionality of Sentaurus Structure Editor	35
ACIS Geometry Kernel.	37
Starting Sentaurus Structure Editor	38
Batch Mode	39
Heap Size.	39
Scheme Script Syntax-Checking	40
Literal Evaluation of Scheme Script	40
Interactive Mode.	41
Loading Boundary and Mesh Command Files at Startup	41
Loading an ACIS File at Startup	41
Exiting Sentaurus Structure Editor.	41
TCAD Sentaurus Tutorial: Simulation Projects	42
<hr/>	
2. Graphical User Interface	43
Introduction to Graphical User Interface	43
GUI Modes.	43
Menu Bar	44
Toolbars	45
Restoring GUI Settings From Previous Session	49
Manipulating the View	50
Lists	51
View Window	51
Command-Line Window.	52
Opening and Saving Operations	53
Opening Models.	53
Saving Models	53

Contents

Saving Boundaries	54
Importing Files	54
Recording Actions to a File	55
Undoing and Redoing Actions	56
Customization	57
Configuring the Command-Line Window	57
Changing GUI Attributes	58
Background Color	58
GUI Style	59
Font Size of Menu Bar of Main Window	59
Size of Main Window	59
Position of Main Window	59
Restoring GUI Settings	59
Selecting Entities	60
Snapping Modes	60
Printing	61
Defining Parameters	61
Parameterizing Dialog Boxes	63
<hr/>	
3. Controlling Views	64
Zooming, Panning, and Orbiting With Mouse Operations	64
Interactive Cutting Plane Viewer	65
Perspective and Orthographic Views in Three Dimensions	66
Selecting the DATEX Color Scheme	67
Selecting the Rendering Mode	67
Showing and Hiding Coordinate Axes	70
View Orientation	70
Displaying Grid Lines	71
Displaying Rulers	71
Scaling the View	71
Visualizing Selected Geometric Objects	72
Quick Access to Placements, Refinements, and Doping Profiles	73
Visualizing the Internal Entity Representation	74

Contents

4. Generating Geometric Structures	76
Modeling Unit and Modeling Range	76
Creating a New Structure	77
Setting Interactive Work Preferences	77
Exact Coordinates	77
Snapping	78
Active Material	78
Naming Regions	79
Overlap Behavior	79
Drawing Basic 2D Shapes	81
Drawing Rectangles	81
Drawing Regular Polygons	82
Drawing Polygons	82
Drawing Circles	82
Drawing Ellipses	83
Drawing Ruled Regions	83
Drawing Other Basic 2D Shapes	85
Editing 2D Shapes	85
Adding a Vertex	85
Moving a Vertex	86
Moving Region Edges	87
Moving Regions	87
Deleting Vertices	88
Rounding	89
Chamfering	90
Cutting	91
Creating a Convex Hull	91
Splitting Structures	92
Simplifying 2D Structures	93
Aligning Vertices	93
Merging Collinear Edges	95
Breaking Nearly Axis-Aligned Edges	99
Two-Dimensional Boundary Smoothing	101
Edge Length Queries	102
Drawing Basic 3D Shapes	104
Drawing Cuboids	104
Drawing Cylinders	105

Contents

Drawing Spheres	106
Drawing Ellipsoids	106
Drawing Other Basic 3D Shapes	107
Import Capability of User-Defined Data	107
Creating a Layered Lens Structure	107
Creating a Solid Body From Faces	109
Editing 3D Shapes	109
Chamfering Edges	109
Rounding Edges.	111
Tapering	112
Creating 3D Objects From 1D and 2D Objects	118
Wires	118
Extruding	119
Sweeping a 2D Object	121
Sweep Distance	122
Sweep Along a Vector	124
Sweep Around an Axis (Regular and Corner Sweeps).	126
Sweep Along a Wire	129
Sweep Options	131
Sweeping Examples.	133
Skinning	136
Default Skinning	136
Skinning With Normals	138
Skinning With Vectors	139
Skinning With Guides	139
Skinning Example.	140
Editing 2D and 3D Structures	141
Explicit Boolean Operations	141
Changing the Material of a Region	143
Changing the Name of a Region	144
Deleting Regions	144
Separating Lumps	145
Two-Dimensional Cuts From a 3D Structure	146
Split-Insert Stretching of a Device	147
Extending a 2D Device.	149
Trimming a 3D Structure	150
Coordinate Systems and Transformations	151
Work Planes	152
Local Coordinate Systems	154

Contents

Device Scaling	155
Entity Transformations: Scaling, Translation, Rotation, and Reflection	156
Local Scaling of Entities	157
Difference Between <code>sdegeo:scale</code> and <code>sdegeo:scale-selected</code>	158
Translation	160
Rotation	161
Reflection	163
Scheme Functions for Transformations	164
5. Structure Generation Using Etching and Deposition	165
Overview	165
Base Coordinate System	166
Unified Coordinate System	166
DF–ISE Coordinate System	167
Selecting the Coordinate System	167
Mask Layout Input and Mask Generation	167
Mask Generation	169
Offsetting (Biasing) Mask	170
TCAD Layout Reader of Sentaurus Structure Editor	170
Loading TCAD Layouts	171
Finding Layer Names and Layer IDs	171
Applying a Stretch	172
Selecting the Simulation Domain	172
Finding Coordinates of Bounding Box	173
Creating and Using Masks	173
Coordinate System Support	173
Layout-Driven Contact Assignment	175
Layout-Driven Meshing	175
Process Emulation Commands	178
Syntax and Keywords	178
Process Emulation Algorithms	186
The <code>lop-move</code> Algorithm	187
The <code>lopx</code> Algorithm	187
The <code>PT</code> Algorithm	187
The <code>sweep</code> Algorithm	188
Restrictions on Handling Mixed Models	189
Process Emulation Steps	190

Contents

Defining Domain	190
Generating Substrate	190
Patterning	190
Patterning Keywords and Keyword Combinations	190
Anisotropic Patterning	191
Isotropic Patterning	192
Photo Operation	192
Photo Keywords and Keyword Combinations	192
Example	192
Deposition	193
Deposition Keywords and Keyword Combinations	193
Anisotropic Deposition	197
Isotropic Deposition	198
Selective Deposition	200
Directional Deposition	201
Shadowing	203
Rounding and Advanced Rounding	204
Etching	205
Etching Keywords and Keyword Combinations	205
Example: Etching Operation Using Different Etch Options	206
Example: Multimaterial Etching	207
Some Notes About Multimaterial Etching	208
Some Notes About Shadowing	210
Fill	210
Polishing	210
Interconnect Layer Generation	211
Example: Generating an Interconnect Structure	212
Shape Library	213
PolygonSTI	214
PolygonWaferMask	214
PolyHedronCylinder	215
PolyHedronEllipticCylinder	215
PolyHedronEpiDiamond	217
PolyHedronSTI	217
PolyHedronSTIacc	219
PolyHedronSTIaccv	220
Removing Material	221
Doping and Implantation	221
Process Emulation Example	223

Contents

6. Electrical and Thermal Contacts	227
Overview	227
Defining and Activating a Contact	228
Deleting a Contact	229
Assigning Edges or Faces to a Contact	229
Assigning a Region Boundary to a Contact	230
Removing Edges or Faces From a Contact	231
Creating Edges or Faces for Use in Contacts	232
Protecting Contacts	233
Examples of Contact Assignments	234
Creating Different 2D Contacts	234
Creating Different 3D Contacts	235
 7. Generating Mesh and Doping Profiles	 238
Overview of Meshing Operations	238
Defining Areas for Mesh Refinement or Doping	239
Line-Segment Ref/Eval Windows	239
Rectangular and Polygonal Ref/Eval Windows	239
Cuboidal Ref/Eval Windows	241
Extracting Ref/Eval Window From Face	241
Extracting Ref/Eval Window From Body Interface	242
Extracting Ref/Eval Window From 3D Solid Body	242
Extracting Ref/Eval Windows From 3D Solid Body Interfaces	243
Deleting Ref/Eval Windows	244
Defining Mesh Refinements	245
Regular Mesh Refinement Boxes	245
Multibox Refinements	250
Offsetting Refinements	252
Defining Doping Profiles	254
Defining Constant Doping Profiles	254
Defining Analytic Doping Profiles	256
Including External 2D and 3D Doping Profiles	261
Examples	264
Defining Particle Doping Profiles	268

Contents

Controlling the Boundary Tessellation	270
Global Tessellation Settings	270
Available Faceters	270
Examples	271
Tessellating Spherical Faces	274
Boundary Tessellation in Three Dimensions	275
Defining Command File Sections Specific to Sentaurus Mesh	276
IOControls Section and Interpolate Section	276
AxisAligned Section	277
Delaunizer Section	279
Tensor Section	281
Boundary Section	281
Boundary Section	282
Tools Section	284
QualityReport Section	284
Building the Mesh	286
8. Creating Planar Layer Stacks	287
Using the sdeepi:create-layerstack Scheme Extension	287
Command File	288
Global Section	289
Size and Position of Structure	292
Global Refinement Strategy	294
User-Defined Extension Columns	294
Layers Section	295
Region Column	297
Material Column	297
SourceParFile Column	297
Thickness Column	297
Doping Column	297
MoleFraction Column	301
Refinement Column	304
Extension Columns	306
Processing the Layer Stack	307
References	308
9. Working With Scheme and Scheme Extensions	309
Scheme Data Types	309

Contents

Basic Scheme Programming	310
Basic Scheme Syntax	311
Defining Simple Variables and Data Types	311
Printing Text to Standard Output	312
String Operations	312
Lists	313
Arithmetic Expressions	314
Boolean Operations	315
If Blocks	316
Simple Do Loops	317
For Each Loops	317
Procedures	318
System Calls	319
Error Signaling to Sentaurus Workbench	319
10. Geometric Query Functions	320
Entity IDs and Attributes	320
Topological Entity Types	321
Selecting Geometric Objects	323
Graphic-Supported Object Selection	323
Script-Based Object Selection	323
Finding Region Names and Material Properties	325
Automatic Region-Naming	326
List of Supported Materials	327
Finding Vertex Positions	327
Vertex–Vertex Distance	328
Debugging Topological Entities	328
Finding Edges, Faces, and Other Elements of a Body	329
Bounding Box Query	330
Scheme Functions for Entity Queries	331
11. Miscellaneous Utilities	333
Background Image Loader	333
User-Defined Dialog Boxes	334
Example: Defining a Dialog Box	335

Contents

Step 1: Define a Scheme Function That the Dialog Box Executes	335
Step 2: Define and Configure the Dialog Box.....	335
Step 3: Launch the Dialog Box.....	336
Starting Sentaurus Structure Editor With User-Defined Variables.....	336
User-Defined GUI Interactions.....	337
Dialog Boxes for Obtaining Values.....	337
GUI Actions for Obtaining Positions.....	337
Message Boxes	338
<hr/>	
A. Commands	339
Presentation of Commands	339
Renamed Commands and Obsolete Commands	339
afm-smooth-layers	342
bbox	343
bbox-exact	344
body?.....	345
build-csv-lens.....	346
color:rgb	347
complete-edge-list	348
convert-to-degree.....	349
convert-to-radian	349
edge?.....	350
edge:circular	351
edge:circular?	352
edge:elliptical?.....	352
edge:end	353
edge:length	353
edge:linear	354
edge:linear?	355
edge:mid-point	356
edge:start	356
edge:type	357

Contents

entity:box	357
entity:copy	358
entity:debug	359
entity:deep-copy	360
entity:delete	360
entity:dist	361
entity:edges	362
entity:erase	362
entity:faces	363
entity:loops	364
entity:lumps	365
entity:set-color	366
entity:shells	367
entity:vertices	368
env:set-tolerance	369
env:tolerance	370
erf	370
erfc	371
exists-empty-mask-name	371
exists-mask-name	372
extract-interface-normal-offset-refwindow	373
extract-interface-offset-refwindow	374
extract-refpolyhedron	375
extract-refwindow	376
face:area	377
face:conical?	378
face:cylindrical?	378
face:planar?	379
face:plane-normal	379
face:spherical?	380
face:spline?	380

Contents

face:toroidal?	381
filter:type	381
find-body-id	382
find-body-id-drs	383
find-drs-id	383
find-edge-id	384
find-edge-id-drs	385
find-face-id	385
find-face-id-drs	386
find-material-id	386
find-region	387
find-region-id	388
find-vertex-id	388
find-vertex-id-drs	389
get-body-list	389
get-drs-list	390
get-empty-mask-list	390
get-mask-list	390
gvector	391
gvector?	392
gvector:+	392
gvector:-	393
gvector:copy	393
gvector:cross	394
gvector:dot	394
gvector:from-to	395
gvector:length	395
gvector:parallel?	396
gvector:perpendicular?	396
gvector:reverse	397
gvector:scale	397

Contents

gvector:set!	398
gvector:set-x!	399
gvector:set-y!	400
gvector:set-z!	401
gvector:transform.	402
gvector:unitize	402
gvector:x	403
gvector:y	403
gvector:z	404
journal:abort.	404
journal:append.	405
journal:clean.	406
journal:load	407
journal:off	407
journal:on	408
journal:pause	409
journal:resume	409
journal:save	409
journal:step	410
loop?	411
loop:external?	411
lump?	412
mask-refevalwin-extract-2d	413
mask-refevalwin-extract-3d	414
member?	415
merge-collinear-edges-2d	416
part:entities	417
part:load.	418
part:save	419
part:save-selection.	420
part:set-name.	421

Contents

position.	422
position?.	423
position:+.	424
position:-.	424
position:distance.	425
position:set!.	426
position:set-x!.	427
position:set-y!.	427
position:set-z!.	428
position:x.	428
position:y.	429
position:z.	429
protect-all-contacts.	430
random-sd.	431
remove-body-ABA.	432
remove-body-BAB.	433
render:rebuild.	434
roll.	434
sde:add-material.	435
sde:back-coord.	436
sde:bg-image-transparency.	436
sde:bool-regularise.	437
sde:bottom-coord.	437
sde:build-mesh.	438
sde:change-datex-color-scheme.	439
sde:check-3d-license-status.	439
sde:check-model.	440
sde:checkout-3d-license.	440
sde:clear.	441
sde:create-bg-image.	442
sde:create-dialog.	443

Contents

sde:define-parameter	444
sde:delay-graphics-update	445
sde:delay-graphics-update?	445
sde:delete-bg-image	446
sde:delete-materials.	446
sde:delete-parameter	447
sde:dialog-add-input	448
sde:dialog-add-pixmap.	449
sde:dialog-delete	450
sde:dialog-ok-command.	451
sde:dialog-show	452
sde:display	452
sde:display-err	453
sde:display-std	454
sde:draw-ruler	455
sde:dump-non-default-options	457
sde:exact-coords?	457
sde:extract-tdr-boundary	458
sde:fix-imprint.	458
sde:fix-orientation.	459
sde:front-coord.	459
sde:ft_scalar.	460
sde:get-backwards-compatibility	461
sde:get-datex-color-scheme	461
sde:get-default-material	462
sde:get-view-params	462
sde:gui-get-integer.	463
sde:gui-get-real	464
sde:gui-get-string	465
sde:hide	466
sde:hide-bg-image	466

Contents

sde:hide-contact	467
sde:hide-interface	467
sde:hide-mask	468
sde:hide-material	469
sde:hide-region	470
sde:hide-ruler	471
sde:info	471
sde:left-coord	472
sde:load-sat	472
sde:material-type	473
sde:max-x	473
sde:max-y	474
sde:max-z	474
sde:merge-materials	475
sde:min-x	476
sde:min-y	476
sde:min-z	477
sde:new-region-name	477
sde:off-lights	477
sde:offset-mask	478
sde:on-lights	478
sde:open-model	479
sde:part-load	480
sde:pick-point-on-wp	481
sde:pick-two-points-on-wp	481
sde:post-message	482
sde:project-name	482
sde:refresh	483
sde:rename-regions	484
sde:restore-cursor	484
sde:right-coord	485

Contents

sde:save-model	486
sde:save-parameters	487
sde:save-tcl-parameters	488
sde:scale-scene	490
sde:scmwin-get-font-families	490
sde:scmwin-get-font-family	491
sde:scmwin-get-font-size	491
sde:scmwin-get-font-style	491
sde:scmwin-get-window-height	492
sde:scmwin-select-font	492
sde:scmwin-set-font-family	492
sde:scmwin-set-font-size	493
sde:scmwin-set-prefs	493
sde:scmwin-set-window-height	494
sde:scmwin-suppress-output	494
sde:selected-entities	495
sde:selected-refeval-windows	495
sde:separate-lumps	496
sde:set-background-color	498
sde:set-backwards-compatibility	499
sde:set-default-material	500
sde:set-menubar-font-size	500
sde:set-process-up-direction	501
sde:set-project-name	502
sde:set-rendering-mode	502
sde:set-selection-level	503
sde:set-translucency	504
sde:set-view-mode	505
sde:set-view-operator	505
sde:set-view-params	506
sde:set-window-position	507

Contents

sde:set-window-size	507
sde:set-window-style	508
sde:setrefprops	509
sde:setup-grid	510
sde:show	511
sde:showattribs	512
sde:show-bg-image	512
sde:show-contact	513
sde:show-grid.	513
sde:show-interface.	514
sde:show-mask	514
sde:show-material	515
sde:show-pcurves	516
sde:show-region.	517
sde:split-solid.	518
sde:stripextension	519
sde:substring	519
sde:test-entity.	520
sde:toggle-lights.	520
sde:top-coord.	521
sde:tr-get	521
sde:view-filter-reset	522
sde:view-set-light-intensity.	522
sde:view-set-visible-area	523
sde:wait-cursor.	523
sde>window-select-2d	524
sde>window-select-3d	525
sde:xshow	526
sde:xshow-contact.	527
sde:xshow-interface.	528
sde:xshow-mask	529

Contents

sde:xshow-material	530
sde:xshow-region.	531
sde:zoom-all.	532
sdedr:append-cmd-file	532
sdedr:clear.	533
sdedr:clear-multibox-definitions	533
sdedr:clear-multibox-placements	533
sdedr:clear-profile-definitions.	534
sdedr:clear-profile-placements.	534
sdedr:clear-ref-windows.	534
sdedr:clear-refinement-definitions	535
sdedr:clear-refinement-placements	535
sdedr:clear-submesh-placement-transform	536
sdedr:convert-mask-to-drs-body	536
sdedr:define-1d-external-profile.	537
sdedr:define-analytical-profile	539
sdedr:define-analytical-profile-placement	540
sdedr:define-body-interface-refwin	542
sdedr:define-constant-profile	543
sdedr:define-constant-profile-material	544
sdedr:define-constant-profile-placement	545
sdedr:define-constant-profile-region	546
sdedr:define-erf-profile.	547
sdedr:define-gaussian-profile.	548
sdedr:define-multibox-placement.	550
sdedr:define-multibox-size	551
sdedr:define-particle-profile	552
sdedr:define-particle-profile-placement	553
sdedr:define-refeval-window	554
sdedr:define-refinement-function	557
sdedr:define-refinement-material.	558

Contents

sdedr:define-refinement-placement	559
sdedr:define-refinement-region	560
sdedr:define-refinement-size	561
sdedr:define-submesh	562
sdedr:define-submesh-placement	563
sdedr:del-selected-drentity	564
sdedr:delete-multibox-placement	564
sdedr:delete-profile-placement	565
sdedr:delete-refeval-window	565
sdedr:delete-refinement-placement	566
sdedr:delete-submesh-placement	567
sdedr:get-cmdprecision	567
sdedr:get-definition-list	567
sdedr:get-placement-list	568
sdedr:hide-mbox	568
sdedr:hide-profile	569
sdedr:hide-refinement	569
sdedr:hide-rewin	570
sdedr:offset-block	571
sdedr:offset-interface	572
sdedr:read-cmd-file	573
sdedr:redefine-refeval-window	574
sdedr:refine-box	575
sdedr:refine-doping	576
sdedr:refine-interface	577
sdedr:set-cmdprecision	578
sdedr:set-title	579
sdedr:show-mbox	579
sdedr:show-profile	580
sdedr:show-refinement	580
sdedr:show-rewin	581

Contents

sdedr:transform-submesh-placement	582
sdedr:write-cmd-file	583
sdedr:write-scaled-cmd-file	583
sdeepi:create-layerstack	584
sdeepi:publish-global-vars	585
sdeepi:scm	586
sdeepi:tcl	587
sdegeo:2d-cut	588
sdegeo:3d-cut	589
sdegeo:align-horizontal	591
sdegeo:align-horizontal-aut	592
sdegeo:align-to-line	593
sdegeo:align-vertical	594
sdegeo:align-vertical-aut	595
sdegeo:average-edge-length	596
sdegeo:body-trim	597
sdegeo:bool-intersect	598
sdegeo:bool-subtract	599
sdegeo:bool-unite	600
sdegeo:break-nearly-axis-aligned-edges	601
sdegeo:chamfer	602
sdegeo:chamfer-2d	604
sdegeo:check-overlap	605
sdegeo:chop-domain	606
sdegeo:chull2d	607
sdegeo:contact-sets	607
sdegeo:create-circle	608
sdegeo:create-cone	609
sdegeo:create-cuboid	611
sdegeo:create-cylinder	612
sdegeo:create-ellipse	614

Contents

sdegeo:create-ellipsoid	615
sdegeo:create-ellipsoid-d	616
sdegeo:create-linear-edge	617
sdegeo:create-ot-ellipsoid	617
sdegeo:create-ot-sphere	619
sdegeo:create-polygon	620
sdegeo:create-polyline-wire	621
sdegeo:create-prism	622
sdegeo:create-pyramid	623
sdegeo:create-rectangle	625
sdegeo:create-reg-polygon	626
sdegeo:create-ruled-region	627
sdegeo:create-sphere	628
sdegeo:create-spline-wire	629
sdegeo:create-torus	630
sdegeo:create-triangle	632
sdegeo:curve-intersect	633
sdegeo:define-3d-contact-by-polygon	634
sdegeo:define-contact-set	635
sdegeo:define-coord-sys	636
sdegeo:define-work-plane	637
sdegeo:del-short-edges	638
sdegeo:delete-collinear-edges	639
sdegeo:delete-contact-boundary-edges	639
sdegeo:delete-contact-boundary-faces	640
sdegeo:delete-contact-edges	641
sdegeo:delete-contact-faces	641
sdegeo:delete-contact-set	642
sdegeo:delete-coord-sys	642
sdegeo:delete-edges	643
sdegeo:delete-nearly-collinear-edges	644

Contents

sdegeo:delete-region	645
sdegeo:delete-short-edges	645
sdegeo:delete-vertices	646
sdegeo:delete-work-plane	646
sdegeo:distance	647
sdegeo:dnce	647
sdegeo:extend-device	648
sdegeo:extrude	650
sdegeo:face-find-interior-point	651
sdegeo:fillet	652
sdegeo:fillet-2d	654
sdegeo:find-closest-edge	655
sdegeo:find-closest-face	655
sdegeo:find-closest-vertex	656
sdegeo:find-touching-faces	657
sdegeo:find-touching-faces-global	658
sdegeo:get-active-work-plane	658
sdegeo:get-auto-region-naming	659
sdegeo:get-contact-edgelist	659
sdegeo:get-contact-facelist	660
sdegeo:get-current-contact-set	660
sdegeo:get-default-boolean	661
sdegeo:get-region-counter	661
sdegeo:imprint-circular-wire	662
sdegeo:imprint-contact	663
sdegeo:imprint-polygonal-wire	664
sdegeo:imprint-rectangular-wire	665
sdegeo:imprint-triangular-wire	666
sdegeo:insert-vertex	666
sdegeo:max-edge-length	667
sdegeo:min-edge-length	668

Contents

sdegeo:mirror-selected	669
sdegeo:move-2d-regions	670
sdegeo:move-edge	671
sdegeo:move-vertex	672
sdegeo:point-entity-relationship	673
sdegeo:polygonal-split	674
sdegeo:prune-vertices	675
sdegeo:ray-test	676
sdegeo:reflect	677
sdegeo:rename-contact	678
sdegeo:revolve	679
sdegeo:rotate-selected	680
sdegeo:scale	682
sdegeo:scale-selected	683
sdegeo:set-active-coord-sys	684
sdegeo:set-active-work-plane	684
sdegeo:set-auto-region-naming	685
sdegeo:set-contact	686
sdegeo:set-contact-boundary-edges	687
sdegeo:set-contact-boundary-faces	688
sdegeo:set-contact-edges	689
sdegeo:set-contact-faces	690
sdegeo:set-contact-faces-by-polygon	691
sdegeo:set-current-contact-set	692
sdegeo:set-default-boolean	693
sdegeo:set-region-counter	694
sdegeo:set-region-counter-aut	695
sdegeo:skin-wires	696
sdegeo:skin-wires-guides	697
sdegeo:skin-wires-normal	698
sdegeo:skin-wires-vectors	699

Contents

sdegeo:split-insert-device	700
sdegeo:sweep	701
sdegeo:sweep-corner	703
sdegeo:taper-faces	704
sdegeo:translate	705
sdegeo:translate-selected	706
sdegeo:vsmooth.	707
sdeicwb:clear	707
sdeicwb:contact	708
sdeicwb:create-boxes-from-layer	713
sdeicwb:define-refinement-from-layer	714
sdeicwb:down	717
sdeicwb:gds2mac	718
sdeicwb:generate-mask-by-layer-name	721
sdeicwb:get-back	722
sdeicwb:get-dimension	722
sdeicwb:get-domains	723
sdeicwb:get-front	723
sdeicwb:get-global-bot	724
sdeicwb:get-global-top	724
sdeicwb:get-label	725
sdeicwb:get-label-for-layer	725
sdeicwb:get-labels	726
sdeicwb:get-layer-ids	726
sdeicwb:get-layer-names	726
sdeicwb:get-layer-polygon-midpoints.	727
sdeicwb:get-left	728
sdeicwb:get-polygon-bounding-boxes-by-layer-name	728
sdeicwb:get-polygon-by-name.	729
sdeicwb:get-polygon-names-by-layer-name	729
sdeicwb:get-region-bot.	730

Contents

sdeicwb:get-region-top	731
sdeicwb:get-right	732
sdeicwb:get-xmax	732
sdeicwb:get-xmin	733
sdeicwb:get-ymax	733
sdeicwb:get-ymin	734
sdeicwb:get-zmax	734
sdeicwb:get-zmin	735
sdeicwb:load-file	736
sdeicwb:mapreader	737
sdeicwb:set-domain	738
sdeicwb:stretch	738
sdeicwb:up	739
sdeio:read-dfise-mask	740
sdeio:read-tdr	741
sdeio:read-tdr-bnd	742
sdeio:save-1d-tdr-bnd	743
sdeio:save-tdr-bnd	745
sdepe:add-substrate	748
sdepe:clean	749
sdepe:define-pe-domain	750
sdepe:depo	752
sdepe:doping-constant-placement	754
sdepe:etch-material	755
sdepe:extend-masks	756
sdepe:fill-device	757
sdepe:generate-domainboundary	758
sdepe:generate-empty-mask	759
sdepe:generate-mask	760
sdepe:icon_layer	762
sdepe:implant	764

Contents

sdepe:pattern	766
sdepe:photo	767
sdepe:polish-device	768
sdepe:remove	769
sdepe:trim-masks	770
sdesnmesh:axisaligned	771
sdesnmesh:boundary	774
sdesnmesh:boundary-clear	775
sdesnmesh:delaunizer	776
sdesnmesh:delaunizer-tolerance	778
sdesnmesh:interpolate	779
sdesnmesh:iocontrols	780
sdesnmesh:offsetting	781
sdesnmesh:qualityreport	782
sdesnmesh:tensor	784
sdesnmesh:tools	785
sdesp:begin	786
sdesp:define-step	787
sdesp:finalize	787
sdesp:restore-state	788
sdestr:capitalize	789
sdestr:casefold	790
sdestr:center	791
sdestr:count	792
sdestr:endswith	793
sdestr:expandtabs	794
sdestr:find	795
sdestr:format	796
sdestr:index	798
sdestr:isalnum	799
sdestr:isalpha	800

Contents

sdestr:isdecimal	801
sdestr:isdigit	802
sdestr:islower	803
sdestr:isnumeric	804
sdestr:isspace	805
sdestr:istitle	806
sdestr:isupper	807
sdestr:join	808
sdestr:length	809
sdestr:ljust	810
sdestr:lower	811
sdestr:lstrip	812
sdestr:partition	813
sdestr:replace	814
sdestr:rfind	815
sdestr:rindex	816
sdestr:rjust	817
sdestr:rpartition	818
sdestr:rsplit	819
sdestr:rstrip	820
sdestr:split	821
sdestr:splitlines	822
sdestr:startswith	824
sdestr:swapcase	825
sdestr:upper	826
sdestr:zfill	827
set-interface-contact	828
shell?	829
skin:options	830
solid?	833
solid:area	833

Contents

solid:massprop	834
sort	835
string:head	836
string:tail	837
sweep:law	838
sweep:options	840
system:command	842
system:getenv	842
timer:end	843
timer:get-time	843
timer:show-time	844
timer:start	844
transform:reflection	845
transform:rotation	846
transform:scaling	847
transform:translation	848
util:make-bot-contact	849
util:make-top-contact	850
vertex?	851
view:set-point-size	851
wire?	852
wire-body?	852
wire:planar?	853
<hr/>	
Index of Scheme Extensions	854

About This Guide

The Synopsys® Sentaurus™ Structure Editor tool is a two-dimensional (2D) and three-dimensional (3D) device structure editor, and a 3D process emulator. The three operational modes – 2D structure editing, 3D structure editing, and 3D process emulation – share a common data representation. Geometric and process emulation operations can be mixed freely, adding more flexibility to the generation of 3D structures.

Two-dimensional models can be used to create 3D structures or 3D structures can be defined directly. When a 3D model is created, three-dimensional device editing operations and process emulation operations can be applied interchangeably to the same model. The 2D and 3D structure editing modes include geometric model generation, doping and refinement definition, and submesh inclusion (to generate the mesh command file).

For additional information, see:

- The TCAD Sentaurus release notes, available on the Synopsys SolvNetPlus support site (see [Accessing SolvNetPlus on page 33](#))
- Documentation available on the SolvNetPlus support site

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
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.
<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.
Key+Key	Indicates keyboard actions, for example, Ctrl+I (press the I key while pressing the Control key).
Menu > Command	Indicates a menu command, for example, File > New (from the File menu, choose New).

Customer Support

Customer support is available through the Synopsys SolvNetPlus support site and by contacting the Synopsys support center.

Accessing SolvNetPlus

The SolvNetPlus 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 SolvNetPlus site:

1. Go to <https://solvnetplus.synopsys.com>.
2. Enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

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 www.synopsys.com. There you can find email addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNetPlus site or the Synopsys Global Support Centers site and open a case (Synopsys user name and password required).

Contacting Your Local TCAD Support Team Directly

Send an email 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

About This Guide

Acknowledgments

Acknowledgments

Portions of this software are owned by Spatial Corp. © 1986–2022. All rights reserved.

1

Introduction to Sentaurus Structure Editor

This chapter introduces Sentaurus Structure Editor and describes its basic operations.

Functionality of Sentaurus Structure Editor

Sentaurus Structure Editor can be used as a two-dimensional (2D) or three-dimensional (3D) structure editor, and a 3D process emulator to create TCAD devices.

In Sentaurus Structure Editor, structures are generated or edited interactively using the graphical user interface (GUI). Doping profiles and meshing strategies can also be defined interactively. Sentaurus Structure Editor features an interface to configure and call the Synopsys meshing engines. In addition, it generates the necessary input files (the TDR boundary file and mesh command file) for the meshing engines, which generate the TDR grid and data file for the device structure.

Alternatively, devices can be generated in batch mode using scripts. Scripting is based on the Scheme scripting language. This option is useful, for example, for creating parameterized device structures. Sentaurus Structure Editor records interactive actions in the form of script commands (journaling). Therefore, it is easy to generate a script from recorded interactive operations. These scripts can be parameterized afterwards.

[Command-Line Window on page 52](#) provides information regarding the Scheme scripting language. When a GUI action is performed, Sentaurus Structure Editor prints the corresponding Scheme command in the command-line window. This facilitates another convenient way to generate and work with scripts. Open a text editor and use cut-and-paste operations to transfer the Scheme commands, which Sentaurus Structure Editor prints in the command-line window, to a script file. In the text editor, the commands can be edited as needed and pasted back into the command window for execution.

In addition, Scheme resources are listed in [Basic Scheme Programming on page 310](#).

Device structures are defined by using geometric operations such as:

- Generation of 2D and 3D primitives (rectangles, circles, cuboids, cylinders)
- Filleting, chamfering, 3D vertex and edge blending, face-tapering operations

Chapter 1: Introduction to Sentaurus Structure Editor

Functionality of Sentaurus Structure Editor

- Boolean operations between bodies
- General extrusion, sweep, and skinning and lofting operations

In addition, device structures can be defined using emulated process steps such as:

- Import or define mask layouts
- Substrate generation
- Patterning
- Isotropic and anisotropic etching and deposition, with or without shadowing and directional effects
- Polishing and fill operations
- Implantation

Sentaurus Structure Editor can be used in three different ways or modes:

- Two-dimensional device editor
- Three-dimensional device editor
- Three-dimensional process emulator (Procem)

The common characteristics of these modes are:

- The use of a state-of-the-art geometry modeling kernel (ACIS, from Dassault Systèmes S.A.) that provides a robust and reliable base for model generation
- A high-quality rendering engine and GUI
- A scripting interface, which is based on the Scheme scripting language
- All modes share a common kernel that provides TDR file input and output, conformal model tessellation for the 2D and 3D Synopsys meshing engines, a link to the meshing engines (with an appropriate user interface to the selected engine), refinement control for tessellated curved boundaries, and other support functionalities

The 2D and 3D device editors provide a GUI and scripting support to:

- Generate the model geometry
- Define contact regions
- Add constant, analytic, and externally generated doping profiles to the model
- Define local refinements
- Include external submeshes
- Interface to the Synopsys meshing engines

Chapter 1: Introduction to Sentaurus Structure Editor

Functionality of Sentaurus Structure Editor

The process emulator provides additional scripting functions to emulate TCAD process steps.

All three modes share the same software infrastructure and internal data representation, and can be combined freely. A 2D model can be extruded or swept along a curve to generate a 3D model. Afterwards, process steps (for example, a deposition step) can be performed on the generated 3D model. Similarly, a 2D slice can be generated from a 3D model and can be saved in a 2D boundary file.

ACIS Geometry Kernel

The geometry operations in Sentaurus Structure Editor are based on the ACIS geometry kernel. The ACIS 3D geometric modeler (ACIS) is an object-oriented 3D geometric modeling engine from Spatial Corp.

ACIS is based on boundary representation. An ACIS boundary representation (B-rep) of a model is a hierarchical decomposition of the topology of the model into lower-level topological objects. A typical body contains lumps, shells, faces, loops, wires, coedges, edges, and vertices.

The Topology Browser in Sentaurus Structure Editor can be used to explore the relationship between these objects. When Sentaurus Structure Editor generates a new body or performs a Boolean operation or any other action that affects the geometry, the ACIS data representation always provides a valid model.

Since the geometry representation is always three dimensional, there is a seamless transition from 2D models to 3D models (using extrusion, sweep operations, and so on). When only 2D objects are present, the TDR output will be two dimensional. For 3D objects, the TDR file is three dimensional.

An entity is the most basic ACIS object. The selection tool and (`sde:selected-entities`) always return the ACIS entity IDs for the selected entities. These entity IDs are used to refer to specific entities.

Apart from the Sentaurus Structure Editor documentation, a useful resource to learn the basics of ACIS and Scheme is:

- J. Corney and T. Lim, *3D Modeling with ACIS*, Stirling, UK: Saxe-Coburg Publications, 2001.

Starting Sentaurus Structure Editor

You can start Sentaurus Structure Editor by typing at the command prompt:

```
sde [<options>]
```

where [<options>] are the command-line options listed in [Table 1](#).

Note:

The geometry engine of Sentaurus Structure Editor is three dimensional. Even if the operating mode is 2D, the underlying geometry representation of the model is three dimensional. The 2D operating mode can be switched to 3D at any time. The Scheme interface is not affected by the 2D GUI mode or 3D GUI mode.

Table 1 Command-line options

Option	Description
-2D	Starts the graphical user interface (GUI) in a simplified 2D mode (see GUI Modes on page 43). Choose View > GUI Mode > 3D Mode to switch to the default 3D mode. In 2D mode, the menu bar is simplified and only 2D-related operations and commands are available. In 3D mode, the menu bar is extended and all features are available.
-action <i>option</i>	Sets the default GUI action to the specified action from the following options: <ul style="list-style-type: none">• draw-polygon• draw-rectangle• orbit (default)• pan• select• zoom
-defaultGUI	Resets GUI parameters to their defaults. The settings include toolbar positions, DATEX color scheme, and parameters for the command-line window. Does not restore the GUI settings from previous session.
-e	Runs in batch mode (see Batch Mode on page 39), that is, without the GUI. Use with -l to run a script in batch mode.
-h <i>heapsize</i>	Heap size in kilobytes. By default, Sentaurus Structure Editor uses 1600 MB of heap space. See Heap Size on page 39 .
-help	Prints the help message.
-l <i>scriptname</i>	Loads and executes the script file called <i>scriptname</i> .

Chapter 1: Introduction to Sentaurus Structure Editor

Starting Sentaurus Structure Editor

Table 1 *Command-line options (Continued)*

Option	Description
-noloadCmd	When loading a boundary file, this option suppresses the loading of the corresponding mesh command file (see Loading Boundary and Mesh Command Files at Startup on page 41).
-noLogFile	Do not create a .log file during the execution of a Scheme script.
-noOpenGL	Explicitly suppresses the use of OpenGL. Note that Sentaurus Structure Editor automatically switches to the noOpenGL mode if OpenGL is not available.
-noSyntaxCheck	Do not run the Scheme syntax checker.
-nowait	Do not wait for license to become available.
-r	Loads and executes script commands from standard input. Use Ctrl+D to revert to the default GUI mode, or Ctrl+\ to quit Sentaurus Structure Editor.
-S <i>scriptname</i>	Tests the Scheme syntax only; it implies -e, that is, no GUI. See Scheme Script Syntax-Checking on page 40 .
-Sl <i>scriptname</i>	Tests the Scheme syntax, and then executes as -l if it passes the syntax check. See Scheme Script Syntax-Checking on page 40 .
-v	Prints information about the tool version.
-var var=value	Defines and loads additional Scheme variables. These variables can be used to parameterize a Scheme script.

Batch Mode

To run a Scheme script file, for example `MyScript.scm`, in batch mode, start Sentaurus Structure Editor with the `-e` option (this disables the graphical display), and use the `-l` option to give the script to be run:

```
sde -e -l MyScript.scm
```

If the `-e` option is not used, the GUI is launched after the specified script `-l MyScript.scm` is executed.

Heap Size

By default, Sentaurus Structure Editor uses 1600 MB of heap space. For most applications, the allocated heap space should be sufficient. If Sentaurus Structure Editor does not have

Chapter 1: Introduction to Sentaurus Structure Editor

Starting Sentaurus Structure Editor

enough heap space during script execution, then the Scheme error file will contain the following error message: Out of heap space.

In this case, Sentaurus Structure Editor exits, and the script must be executed again with increased heap space. To increase the heap space, use the -h argument. For example:

```
(sde -h 2400000 -e -l myscript.scm)
```

Scheme Script Syntax-Checking

The `-S scriptname` and `-S1 scriptname` options invoke the Scheme syntax-checking feature.

The `-S` option runs the syntax-checker on the Scheme script *scriptname* and reports on the results without executing the script.

In contrast, after first running the syntax-checker on the Scheme script *scriptname*, the `-S1` option will subsequently run the script if it passes the syntax check; otherwise, it will not run the script.

Note:

Due to the complexity of the Scheme scripting language, it is possible, in some cases, for the syntax-checker to report a false positive. This means a syntax error might be flagged in complex code even though no syntax error exists. In the case of the `-S1` option, the script is not run if there is an inaccurate flagging of a nonexistent syntax error. The option `-l` can be used instead to bypass the syntax check and to run the script if it is indeed free of syntax errors. You are encouraged to report such cases, so that the number of false positives will be reduced in future versions (contact the TCAD Support Team).

Literal Evaluation of Scheme Script

When a Scheme script is executed in batch mode, a log file is created that contains the executed Scheme commands. The log file does not contain the literal evaluation and substitution of the Scheme variables. Therefore, for example, the log file will contain `(position xp yp zp)`, not the actual numeric values of the `xp`, `yp`, and `zp` variables that were used during the script evaluation.

In some cases, you might want to see the actual numeric values that were used during command execution. To facilitate this, an `.eval` file also can be created during script evaluation. The global Scheme variable `evaluate-log-file` can be used to trigger the logging of the evaluated Scheme commands. By default, the log file evaluation is disabled and the value of the `evaluate-log-file` global Scheme variable is set to `#f`. If you want to create the evaluated Scheme file, you must add the `(set! evaluate-log-file #t)` command to the Scheme script.

Chapter 1: Introduction to Sentaurus Structure Editor

Exiting Sentaurus Structure Editor

If the input Scheme script is called `test.scm`, the log file is saved as `test.log` and the evaluated log file is saved as `test.log.eval`. If a script fails for some reason, the evaluated log file can be used to check the failing command, where all user-defined variables contain the actual values that were used during command execution.

The evaluated log file shortens and simplifies debugging and bug reporting, since the variables do not need to be evaluated separately.

Interactive Mode

Sentaurus Structure Editor can be used interactively by either using the GUI menu bar and toolbars, or entering the Scheme commands in the command-line window.

To run Sentaurus Structure Editor in interactive mode, type in a command prompt:

```
sde
```

Loading Boundary and Mesh Command Files at Startup

To load a boundary (`*_bnd.tdr`) file and a mesh command (`*.cmd`) file when starting Sentaurus Structure Editor, supply the common file name stem as a command-line option. For example:

```
sde MyDevice
```

This command starts Sentaurus Structure Editor and loads the `MyDevice.cmd` and `MyDevice_bnd.tdr` files.

Loading an ACIS File at Startup

To load a file in the native ACIS format (`*.sat`) when starting Sentaurus Structure Editor, enter the file name as a command-line option. For example:

```
sde MyDevice.sat
```

This command starts Sentaurus Structure Editor and loads the `MyDevice.sat` file.

Exiting Sentaurus Structure Editor

To exit Sentaurus Structure Editor:

- Choose **File > Exit** or press **Ctrl+Q**.

Chapter 1: Introduction to Sentaurus Structure Editor

TCAD Sentaurus Tutorial: Simulation Projects

The corresponding Scheme command is:

```
(exit)
```

Note:

If the model has changed since the last save operation, then Sentaurus Structure Editor displays a warning in the interactive mode.

Press **Ctrl+** to terminate Sentaurus Structure Editor from the command prompt as **Ctrl+C** is not recognized.

TCAD Sentaurus Tutorial: Simulation Projects

The TCAD Sentaurus Tutorial provides various projects demonstrating the capabilities of Sentaurus Structure Editor.

To access the TCAD Sentaurus Tutorial:

1. Open Sentaurus Workbench by entering the following on the command line: `swb`
2. From the menu bar of Sentaurus Workbench, choose **Help > Training** or click  on the toolbar.

Alternatively, to access the TCAD Sentaurus Tutorial:

1. Go to the `$STROOT/tcad/current/Sentaurus_Training` directory.
The `STROOT` environment variable indicates where the Synopsys TCAD distribution has been installed.
2. Open the `index.html` file in your browser.

2

Graphical User Interface

This chapter describes the graphical user interface of Sentaurus Structure Editor.

Introduction to Graphical User Interface

The graphical user interface (GUI) of Sentaurus Structure Editor has three main areas (see [Figure 1 on page 44](#)). The menu bar, toolbars, and list boxes are located in the upper part of the main window, the view window is in the center, and the command-line window is in the lower part.

GUI Modes

Sentaurus Structure Editor offers different GUI modes: 3D (default mode) and 2D. To set the GUI mode:

- Choose **View > GUI Mode > 2D Mode**.
- Choose **View > GUI Mode > 3D Mode**.

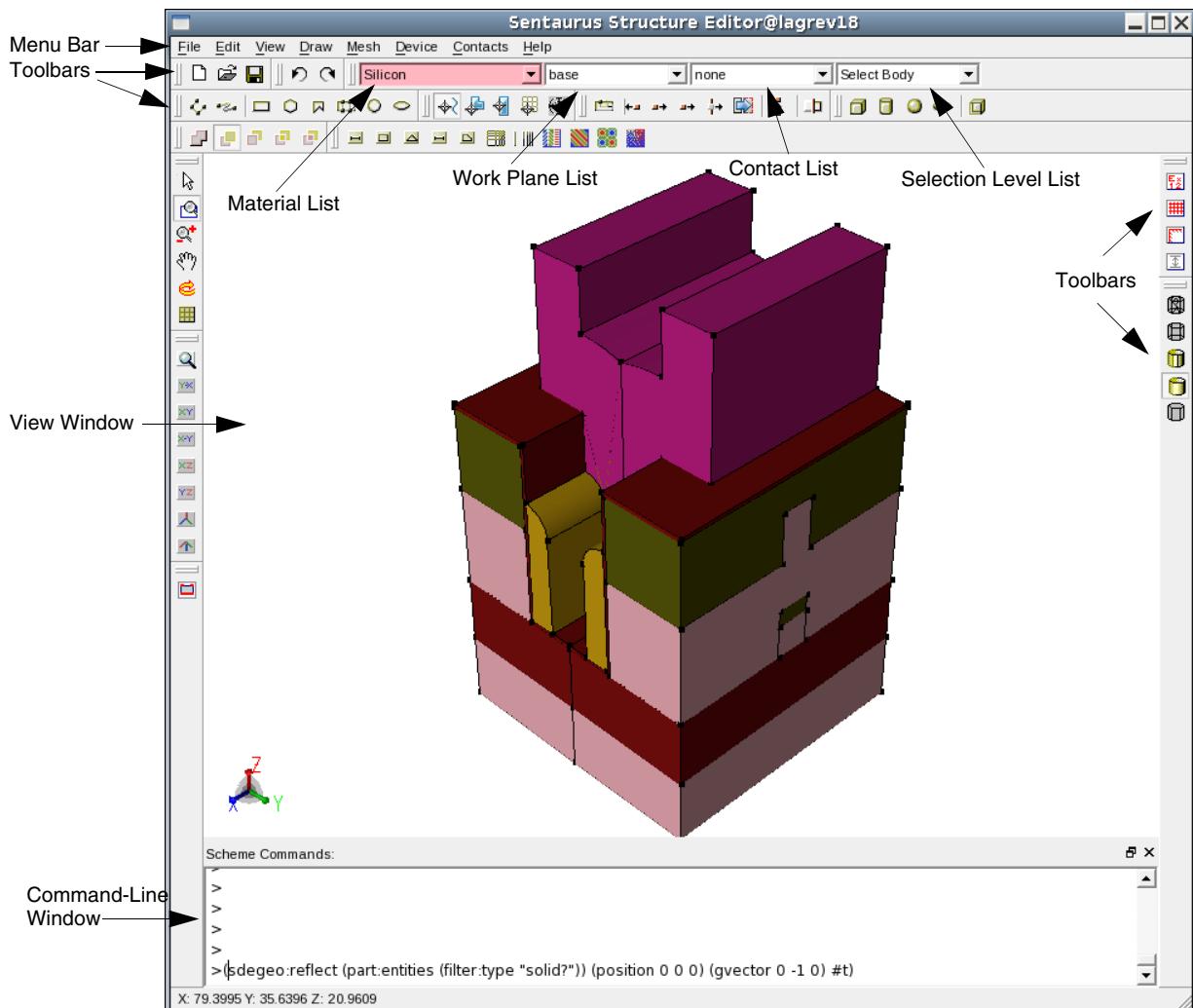
The only difference between the 3D GUI mode and the 2D GUI mode is that in the 2D GUI mode, some toolbars and GUI operations are unavailable. In this way, a more streamlined and simplified GUI is provided for 2D applications.

The dialog boxes related to doping, refinement, and external submeshes are the same in both the 2D GUI mode and 3D GUI mode. However, when defining refinement/evaluation (Ref/Eval) windows directly from these dialog boxes, in 2D GUI mode, rectangular Ref/Eval windows are created and, in 3D GUI mode, cuboidal Ref/Eval windows are created.

Chapter 2: Graphical User Interface

Introduction to Graphical User Interface

Figure 1 Main window of Sentaurus Structure Editor



Menu Bar

Table 2 lists the menus that are available from the GUI of Sentaurus Structure Editor.

Table 2 Menus

Menu	Description
File	Load, save, and print functions
Edit	Change existing geometric objects

Chapter 2: Graphical User Interface

Introduction to Graphical User Interface

Table 2 Menus (Continued)

Menu	Description
View	Visualization preferences and auxiliary views
Draw	Drawing and basic object creation, preferences
Mesh	Define a meshing strategy, call the meshing engine, visualize the generated mesh and data fields
Device	Define doping profiles
Contacts	Define and edit contacts and contact sets
Help	Version information

Toolbars

Each toolbar contains a set of predefined shortcuts and icons, which are shown in the following tables.

Table 3 File toolbar buttons

Button	Shortcut keys	Description	Button	Shortcut keys	Description
	Ctrl+N	Create new file		Ctrl+S	Save model
	Ctrl+O	Open model		Ctrl+P	Print

Table 4 Edit toolbar buttons

Button	Shortcut keys	Description
	Ctrl+Z	Undo last operation
	Ctrl+Y	Redo last operation

Chapter 2: Graphical User Interface

Introduction to Graphical User Interface

Table 5 Rendering mode toolbar buttons (for 3D)

Button	Description	Button	Description
	Facets		Gouraud Shaded
	Wireframe		Hidden Line
	Flat Shaded		

Table 6 Standard views toolbar buttons

Button	Description	Button	Description
	Isometric View		YZ Plane
	XY Plane		X-Y Plane (y-axis pointing downwards)
	XZ Plane		Zoom to Extents (reset zoom)
	Isometric View (Sentaurus Process up direction)		

Table 7 GUI actions toolbar buttons (zoom and move)

Button	Description	Button	Description
	Select (selects a single object or objects in a window drawn with mouse)		Zoom (zooming controlled by mouse)
	Zoom to Window (zooms to window drawn with mouse)		Pan (move device with mouse)
	Orbit (3D rotation)		Cut Plane (shows a cross section of the structure)

Chapter 2: Graphical User Interface

Introduction to Graphical User Interface

Table 8 GUI actions toolbar buttons (2D edit tools)

Button	Description	Button	Description
	Polygonal Region Split		Move Region
	Add Vertex		Vertex-Vertex Distance
	Move Vertex or Move Vertices		2D Cut
	Move Edge		

Table 9 GUI actions toolbar buttons (2D create tools)

Button	Description	Button	Description
	Create Polyline Wire		Create Spline Wire
	Create Rectangle		Create Regular Polygon (needs Exact Coordinates mode to set parameters)
	Create Polygon (end polygon with middle mouse button)		Create Ruled Region
	Create Circular Region		Create Elliptical Region

Table 10 Snapping actions toolbar buttons

Button	Description	Button	Description
	No Snapping		Snap to Edge
	Snap to Vertex		Snap to Grid

Chapter 2: Graphical User Interface

Introduction to Graphical User Interface

Table 10 Snapping actions toolbar buttons (Continued)

Button	Description	Button	Description
	Toggle gravity snapping		

Table 11 GUI actions toolbar buttons (3D create tools)

Button	Description	Button	Description
	Create Cuboid		Create Cylinder
	Create Sphere		Create Ellipsoid
	Create Cross Section		

Table 12 Default Boolean toolbar buttons

Button	Description	Button	Description
	Merge (new and existing objects are merged; new material and region names are assigned)		New object replaces old in overlapping regions, but the overlap becomes a separate region
	New object replaces old in overlapping regions		Old object replaces new in overlapping regions, but the overlap becomes a separate region
	Old object replaces new in overlapping regions		

Table 13 Shortcut toolbar buttons

Button	Description	Button	Description
	Switch between geometry editor and grid and data viewer		Switch on or off exact coordinates

Table 13 Shortcut toolbar buttons (Continued)

Button	Description	Button	Description
	Grid visualization switch		Ruler visualization switch
	Auto stretch scene		

Table 14 DRS (doping, refinement, submesh) toolbar buttons

Button	Description	Button	Description
	Reference line (Ref/Eval window)		Reference rectangle (Ref/Eval window)
	Reference polygon (Ref/Eval window)		Reference element extracted from a geometric face (Ref/Eval window)
	Refinement placement		Multibox placement
	Constant profile placement		Analytic profile placement
	External profile placement		Particle profile placement

Restoring GUI Settings From Previous Session

Some GUI information is saved automatically in your local home directory. On Linux operating systems, this information is stored in the file `~/.config/Synopsys/Sentaurus Structure Editor.conf`.

The stored parameters are toolbar positions, the setting of the DATEX color scheme, and the parameters for the command-line window.

To reset these GUI parameters to their defaults, begin an interactive session by using the `-defaultGUI` command-line option. This starts the session with the default GUI parameters and replaces any personalized parameters with these defaults when exiting.

Manipulating the View

Additional shortcut key combinations are available to manipulate the view.

Table 15 Shortcut keys for manipulating the view

Shortcut keys	Action
Up Arrow	View window: Zoom out Command-line window: Scroll up command history
Down Arrow	View window: Zoom in Command-line window: Scroll down command history
Left Arrow	View window: Pan to the left Command-line window: Position cursor for editing echoed Scheme command
Right Arrow	View window: Pan to the right Command-line window: Position cursor for editing echoed Scheme command
Ctrl+Left Arrow	View window: Pan downwards
Ctrl+Right Arrow	View window: Pan upwards
Ctrl+Z	(roll)
Ctrl+Y	(roll 1)
Shift+Orbit	Rotate about x-axis (horizontal view) or Pan (when (<code>sde:use-camera-manipulator #t</code>) is used)
Ctrl+Orbit	Rotate about y-axis (vertical view) or Zoom in and out (when (<code>sde:use-camera-manipulator #t</code>) is used)
Ctrl+Shift+Orbit	Rotate about z-axis (perpendicular to view)
Ctrl+N	Create new file
Ctrl+O	Open model
Ctrl+S	Save model
Ctrl+P	Print
Ctrl+Q	Exit

Lists

The main window of Sentaurus Structure Editor contains the following lists:

- Material list – Selects the material to be assigned to new objects.
 - Work Plane list – Sets the work plane for 3D editing.
 - Contact list – Selects the contact name to be used in the next set contact region, or face, or edge operations.
 - Selection Level list – Selects which type of object can be selected.
-

View Window

The current device is displayed in the view window. The result of all interactive operations is reflected immediately in the view window.

Right-click in this area to open the following shortcut menus:

- **Selection Level** controls which type of object can be selected by clicking (Select mode). Options are **Auto Select**, **Body**, **Face**, **Edge**, **Vertex**, and **Ref/Eval Window**.
- **Toggle Visibility** activates or deactivates a light source for shading 3D objects.

Placing the cross-hair cursor over an object and right-clicking selects the object corresponding to the current selection level and opens these shortcut menus.

The following additional menus and options become available, where appropriate, in the context of the selection made:

- **Contacts:** If the selection level is set to **Edge** or **Face**, contacts are assigned to the selected objects. It also allows for the creation of new contact sets.
- **Delete:** In many contexts, the object can be deleted by selected this option.
- **Properties:** Displays a window with information about the selected object. In some contexts, for example, Ref/Eval windows, properties of the object are editable.

For Ref/Eval windows, the bounding vertex coordinates are displayed for rectangular and cuboidal Ref/Eval windows. For these types, the vertex coordinates are editable and can be changed. In this case, all doping/refinement/submesh (DRS) objects that use the given Ref/Eval window are redefined using the new vertex coordinates.

- **Hide:** If the selection level is set to **Body** or **Ref/Eval Window**, the selected entities are hidden.
- **Show All:** All the previously hidden bodies and Ref/Eval windows are displayed again.

Command-Line Window

Most GUI operations have an associated Scheme command. After a GUI operation, the corresponding Scheme command is echoed in the command-line window.

Use the Up Arrow and Down Arrow keys to scroll through echoed Scheme commands. Echoed commands can be edited (use the Left Arrow and Right Arrow keys to position the cursor, and the Delete key or Backspace key to delete parts of the command, type in new parts of the command) and reexecuted by pressing the Enter key.

Use cut-and-paste operations to save echoed Scheme commands into a text editor to interactively build a Scheme script. Individual Scheme commands or groups of Scheme commands can be pasted back into the command-line window as needed, for example, to test a section of a Scheme script. Scheme commands can also be entered directly at the command-line prompt.

Activate journaling to automatically save the echoed Scheme command to a journal file, for later editing and replaying (see [Recording Actions to a File on page 55](#)).

The following basic rules apply to the Scheme scripting language (see [Chapter 9 on page 309](#)):

- Comment lines start with a semicolon. In each line, a comment proceeds from the first inserted semicolon.

New variables are defined using the keyword `define`. Defined variables can be reassigned a value by using the `set!` keyword. For example:

```
(define width 5) ; defines variable width and sets its value to 5
(define height 10)
(set! width 3)      ; now width is 3
```

- Scheme has all the conventional data types such as character, string, list, array, Boolean, number, function, and symbol.
- All data types are equal. Any variable can hold any type of data. Data initialization, memory allocation, and memory cleanup (garbage collection) are performed automatically.

Note:

A few keywords (such as `length`) are reserved keywords used by Scheme and they define a Scheme function or operator. These keywords must not be redefined by a user-initiated `define` command.

Opening and Saving Operations

The next sections describe basic opening and saving operations.

Opening Models

A model consists of a structure saved in the native ACIS format (*filename.sat*) and an auxiliary Scheme script file (*filename.scm*), which contain parameter settings, contact definitions, refinement/evaluation (Ref/Eval) windows, surface refinement settings, as well as work plane and view settings (some of these definitions are also part of the *.sat* file). Refinement-related and doping-related information is stored in a third file (*filename.cmd*).

To open a model:

- Choose **File > Open Model** or press Ctrl+O.

The corresponding Scheme command is:

```
(sde:open-model "filename")
```

Saving Models

To save a model, you can do one of the following:

- Choose **File > Save Model** or press Ctrl+S.
- Choose **File > Save Model As**.

The corresponding Scheme command is:

```
(sde:save-model "filename")
```

This command saves the following:

- Model geometry and Ref/Eval windows in native ACIS format (*filename.sat*)
- Scheme file (*filename.scm*) with various parameter settings
- Mesh command file (*filename_msh.cmd*) with refinement-related and doping-related information
- Boundary file in TDR format (*filename_bnd.tdr*)

Saving Boundaries

To save only the boundary (and not the entire model) in TDR format, you can do one of the following:

- Choose **File > Save Boundary**.
 - Choose **File > Save Boundary As**.
-

Importing Files

To import a file:

- Choose **File > Import**.

Table 16 Supported file formats for loading into Sentaurus Structure Editor and corresponding Scheme commands

File format	Scheme command	Description
ACIS SAB file (*.sab)	(part:load "filename.sab" #f)	Native ACIS format (binary) to store the complete model
ACIS SAT file (*.sat)	(part:load "filename.sat" #t)	Native ACIS format (ASCII) to store the complete model
Mesh doping and refinement file (*.cmd)	(sdedr:read-cmd-file "filename.cmd")	Command file for the Synopsys mesh generator
Layout file (*.lyt)	(sdeio:read-dfise-mask "filename.lyt")	Layout file in deprecated DF-ISE format
Scheme script file (*.scm, *.cmd)	(load "filename.scm")	Scheme command file that is loaded and executed
TDR file (*.tdr)	(sdeio:read-tdr-bnd "filename.tdr")	Boundary file in TDR format

Note:

When importing a .sat file, the Scheme command `part:load` is used. This command works well when importing a structure into an empty database (no geometric objects defined). In this case, the saved structure is restored correctly.

When importing a structure to a non-empty database (with preexisting geometric objects), overlapping regions might be generated since `part:load` does not respect the active Boolean settings for overlap handling.

Chapter 2: Graphical User Interface

Recording Actions to a File

You can use the `sde:load-sat` function (not accessible from the GUI) to load a native `.sat` file with correct overlap handling. The function is the same as `part:load`, except that it observes the active Boolean setting for overlap control (that is, overlapping regions will not be present in the model and the overlaps will be removed based on the active Boolean setting).

Recording Actions to a File

Operations performed using the GUI or command-line window can be recorded in a journal file for replaying or editing later.

To activate, suspend, and deactivate the journal feature:

1. Choose **File > Journal > On**.
2. Enter the name of the journal file in which all further actions should be recorded.
3. Choose **File > Journal > Pause** to suspend the recording of actions.
4. Choose **File > Journal > Resume** to resume recording actions.
5. Choose **File > Journal > Off** to end recording actions.

To replay a journal file:

1. Choose **File > Journal > Load**.
2. Enter the name of the journal file.

To execute the journal file step-by-step, open the journal file in a text editor and enter the Scheme command (`journal:step #t`) at the point in the script where the stepping mode should start.

Note:

In stepping mode, the Enter key must be pressed twice for each single line of the Scheme script, including blank lines and comments. To revert to continuous execution, enter the command (`journal:step #f`).

To save the current journal to a file:

1. Choose **File > Journal > Save**.
2. In the dialog box, save the file as required.

Chapter 2: Graphical User Interface

Undoing and Redoing Actions

Table 17 Corresponding Scheme commands for journal features

Scheme command	Description
(journal:clean "filename.jrl")	Cleans the specified journal file. (Removes all nonexecutable content from the file to simplify debugging and parameterization.)
(journal:load "filename.jrl")	Loads a journal file and runs each command contained in that file. Each line is journaled if journaling is switched on.
(journal:off)	Closes the current journal file and switches off journaling.
(journal:on "filename.jrl")	Switches on journal recording. All future commands are journaled to the file.
(journal:pause)	Pauses recording.
(journal:resume)	Resumes recording.
(journal:save "filename.jrl")	Saves the current journal to a file, but leaves the journal session open.

Undoing and Redoing Actions

To undo an action:

- Choose **Edit > Undo** or press Ctrl+Z.

The corresponding Scheme command is:

(roll)

To redo an action:

- Choose **Edit > Redo** or press Ctrl+Y.

The corresponding Scheme command is:

(roll+1)

Note:

Undo and redo operations work in multiple steps as well. For example, (roll -n) rolls back the modeler by n steps, and (roll n) rolls forward the history stream of the modeler by n steps. (roll) is a shortcut to (roll -1).

Chapter 2: Graphical User Interface

Customization

Undo and redo operations work with actions supported directly by the ACIS modeling engine, including operations that create and change Ref/Eval windows. Operations that involve creating or changing placements cannot be undone by using the (`roll`) feature. However, placements can be edited, deleted, and recreated.

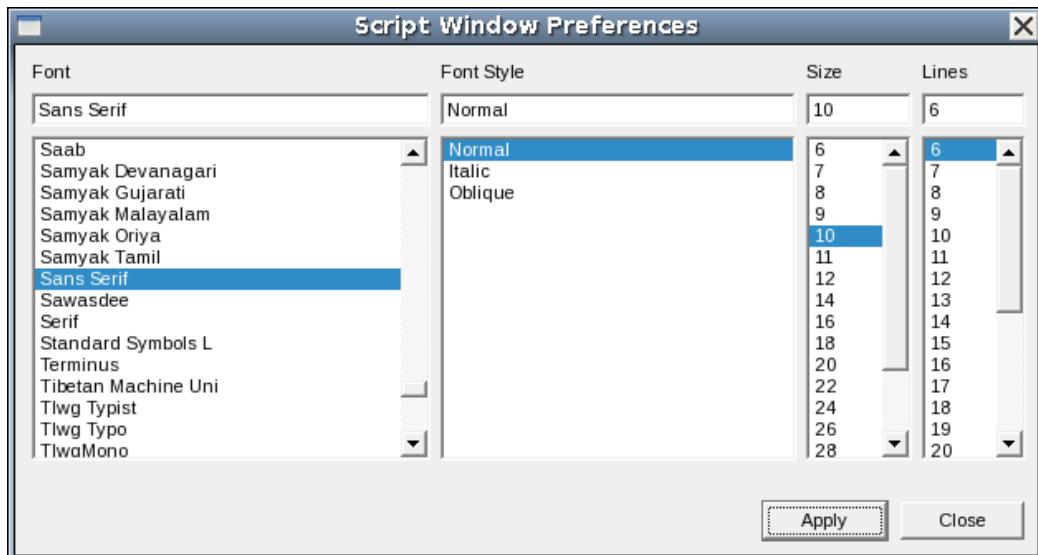
Customization

Some parts of the GUI can be customized.

Configuring the Command-Line Window

You can configure the command-line window by choosing **View > Script Win Prefs**, which displays the Script Window Preferences dialog box, where you select the font family, font style, font size, and height of the command-line window.

Figure 2 Script Window Preferences dialog box



Alternatively, you can use Scheme extensions (see [Table 18](#)).

Table 18 Scheme extensions for configuring the command-line window

Scheme extension	Description
(sde:scmwin-get-font-families)	Returns all available font families.
(sde:scmwin-get-font-family)	Returns the currently selected font family.

Chapter 2: Graphical User Interface

Customization

Table 18 Scheme extensions for configuring the command-line window (Continued)

Scheme extension	Description
(sde:scmwin-get-font-size)	Returns the active font size.
(sde:scmwin-get-font-style)	Returns the font style.
(sde:scmwin-get-window-height)	Returns the height of the command-line window.
(sde:scmwin-select-font)	Displays the Select Font dialog box.
(sde:scmwin-set-font-family font-type)	Sets the font family.
(sde:scmwin-set-font-size font-size)	Sets the font size.
(sde:scmwin-set-prefs)	Can be used to configure the command-line window, using one single function.
(sde:scmwin-set-window-height height)	Sets the height of the command-line window in terms of number of lines.

Note:

The height, font family, font size, and font style of the command-line window are stored in your local home directory when exiting Sentaurus Structure Editor, making these parameters persistent from session to session. For information about how to restore default values, see [Restoring GUI Settings From Previous Session on page 49](#).

Changing GUI Attributes

The following GUI attributes are saved by the `sde:save-model` command and restored by the `sde:open-model` command.

Background Color

To change the background color of the view window:

- Choose **View > Background Color**.

The corresponding Scheme command is:

```
(sde:set-background-color r_top g_top b_top r_bot g_bot b_bot)
```

Chapter 2: Graphical User Interface

Customization

RGB colors for the top and bottom of the view window must be specified as integers in the range 0–255. For example, to create a graded background with red on top and blue at the bottom, specify:

```
(sde:set-background-color 255 0 0 0 0 255)
```

GUI Style

To change the GUI style:

- Choose **View > GUI Style**.

The available GUI styles are Windows, Motif, Cleanlooks, Plastique, and CDE. See [sde:set-window-style on page 508](#).

Font Size of Menu Bar of Main Window

To change the font size of the menu bar of the main window, use:

```
(sde:set-menubar-font-size font-size)
```

For example:

```
(sde:set-menubar-font-size 10)
```

Size of Main Window

You can resize the main window at any time using the mouse or by using:

```
(sde:set-window-size x-size y-size)
```

For example:

```
(sde:set-window-size 640 480)
```

Position of Main Window

You can position the main window on the screen by moving it using the mouse or by using:

```
(sde:set-window-position x y)
```

For example, to place the main window in the upper-left corner of the screen, specify:

```
(sde:set-window-position 0 0)
```

Restoring GUI Settings

To capture figures for presentations or to compare models visually, the state of the GUI graphical view (translation, zoom, rotation) can be recorded and restored using Scheme

Chapter 2: Graphical User Interface

Selecting Entities

functions. The `sde:get-view-params` function returns the actual view parameters. The view parameters are restored using the `sde:set-view-params` function. For example:

```
(define myview (sde:get-view-params))
; The myview object stores the GUI settings, which can be restored
; later.
(sde:set-view-params myview)
```

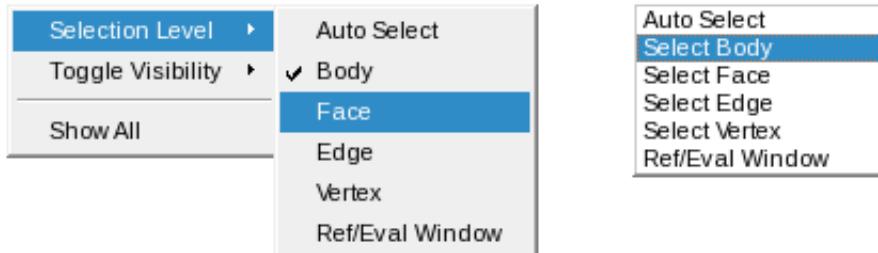
Selecting Entities

To select an entity:

1. Click the Select button (see [Table 7 on page 46](#)).
2. Select the required entity type from the **Selection Level** list.
3. Select an entity and hold the Shift or Ctrl key to select additional entities, or drag a box around a set of entities.
4. Click the (blank) background to clear the selected entity list.

Entity types are **Body**, **Face**, **Edge**, **Vertex**, or **Ref/Eval Window**.

Figure 3 Set entity selection type: (left) shortcut menu and (right) Selection Level list



Snapping Modes

The default drawing mode is freehand for creating 2D regions (rectangles, circles, ellipses, and so on). Optional features such as snap-to-grid, exact coordinates, and snap-to-existing vertices can also be used. These features can also be accessed interactively during model generation.

For example, during a 2D polygonal region generation when the E key is pressed, the pointer will snap to the closest existing edge. When the pointer is moved, it slides along the closest edge. When the V key is pressed, the pointer snaps automatically to the closest vertex. To move back to freehand drawing, press the N key.

Chapter 2: Graphical User Interface

Printing

Table 19 *Keyboard keys for 2D drawing operations*

Key	Action
Esc	Resets operator
V	Snap-to-vertex
E	Snap-to-edge (closest point on edge)
G	Snap-to-grid
N	Disables snapping

Printing

To print the current view of the structure:

1. Choose **File > Print** or press Ctrl+P.
2. Select the printer and set print options such as page orientation, and color or black-and-white mode if available.
3. Select the **Print to file** option, and enter a file name to export the view to a portable document format (PDF) file.
4. Click **Print**.

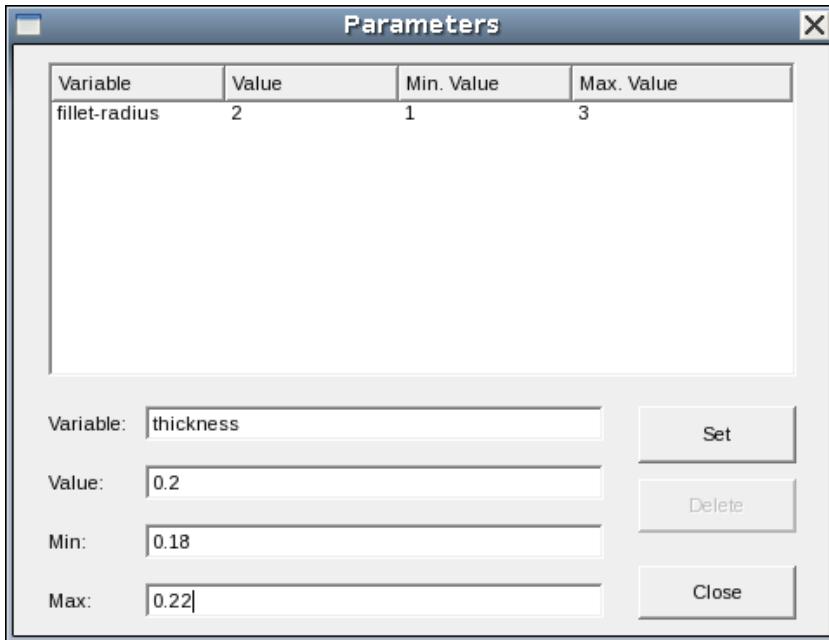
Defining Parameters

To define parameters:

1. Choose **Edit > Parameters**.
- The Parameters dialog box opens.
2. Enter the parameter names and values.

Chapter 2: Graphical User Interface

Defining Parameters



3. Click **Set**.

The corresponding Scheme command is:

```
(sde:define-parameter parameter-name value [min-value max-value])
```

See [sde:define-parameter on page 444](#). For example:

```
(sde:define-parameter "myvar1" "banana")  
(sde:define-parameter "myvar2" 10)  
(sde:define-parameter "myvar3" 10 0 20)
```

To delete parameters, click the **Delete** button of the Parameters dialog box or use the `sde:delete-parameter` function (see [sde:delete-parameter on page 447](#)).

Note:

Parameter names must be enclosed in double quotation marks when using `sde:define-parameter` or `sde:delete-parameter`.

Scheme variables function as parameters for various operations, such as extrusion and chamfering, and can be used for device parameterization. These variables can be assigned not only a value by choosing **Edit > Parameters**, but also minimum and maximum values. For example, when `myvar1` is defined with Value 10, Min. Value 0, and Max. Value 20, you can enter `myvar1` in the command-line window that will evaluate `myvar1` as 10. The minimum value is stored in a variable called `myvar1_min`, and the maximum value is stored in a variable called `myvar1_max`.

Chapter 2: Graphical User Interface

Parameterizing Dialog Boxes

Note:

Defined parameters can be used in all subsequent Scheme commands as well as in the fields of the dialog box that expect numeric values.

Parameters can also be defined by the basic Scheme command (`(define parameter-name value)`).

Parameterizing Dialog Boxes

Several dialog boxes can be parameterized. The dialog boxes can accept not only numeric inputs, but also Scheme variables and Scheme expressions. The Scheme variables must be defined and initialized *before* they can be used in a Sentaurus Structure Editor dialog box. For example, in Exact Coordinates mode, the Exact Coordinates dialog box is displayed after a rectangle is drawn. As input data, in addition to numeric values, Scheme variables and Scheme expressions can be used to define the vertex coordinates.

For example, after the two Scheme variables `width` and `height` are defined in the command-line window, using:

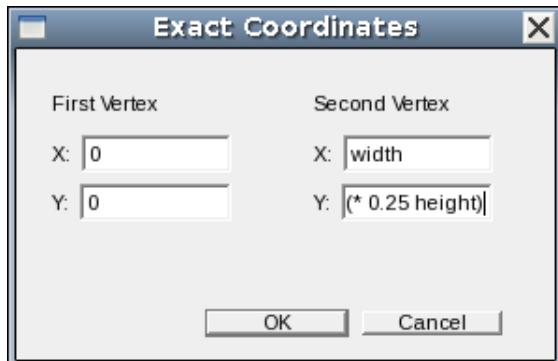
```
(define width 10)
(define height 5)
```

and the Exact Coordinates mode is switched on and a rectangle is drawn, the Exact Coordinates dialog box is displayed where you can use Scheme variables and expressions as input data.

This operation generates the following Scheme command:

```
(sdegeo:create-rectangle (position 0 0 0.0)
                          (position width (* 0.25 height) 0.0) "Silicon" "region_1")
```

Figure 4 Exact Coordinates dialog box using Scheme variables and expressions



3

Controlling Views

This chapter describes the various options to control views available in Sentaurus Structure Editor.

Zooming, Panning, and Orbiting With Mouse Operations

To access the different modes of mouse operations:

- Choose **View > Interactive Mode**, or click the corresponding toolbar button (see [Figure 5](#)).

Figure 5 Toolbar buttons for zooming, panning, and orbiting with mouse operations



The options are:

Zoom to Extents

Resets the zoom factor so that the entire structure is displayed.

Zoom to Window

Drag to define a zoom window. The view is updated such that the defined zoom window is displayed as large as possible. (If the aspect ratio of the selected zoom window differs from the current view window aspect ratio, the zooming might appear to be less than expected.)

Orbit

In this mode, dragging adjusts the camera position in a way that simulates orbiting around the drawn structure. Both the position and path of the pointer during the dragging operation provide an intuitive mechanism for rotation about different axes. Dragging left or right along the middle of the view window rotates the structure about the vertical axis. Similarly, dragging up or down along the center of the view window rotates the structure about the horizontal axis. Dragging up, down, left, or right near any edge of the view window rotates the structure about the axis perpendicular to the view window. Combinations of the movements are designed to give you control over the viewing angle of the drawn structure.

While the Orbit mode is active, different shortcut keys can be used to rotate the model about the imaginary coordinate axes of the screen (in the view window). Hold the key or keys when in Orbit mode:

Rotate about x-axis (horizontal view)	Shift+Orbit
Rotate about y-axis (vertical view)	Ctrl+Orbit
Rotate about z-axis (perpendicular to view)	Ctrl+Shift+Orbit

The `(sde:use-camera-manipulator #t)` function can be used to change these key functionalities to another type of behavior. (The `(sde:use-camera-manipulator #f)` command can be used to change it back.)

After the `(sde:use-camera-manipulator #t)` command is executed when in Orbit mode, the Ctrl key can be used to zoom and the Shift key can be used to pan (while pressing the left mouse button).

Zoom

In this mode, dragging upwards zooms in on the structure, and dragging downwards zooms out.

Pan

In this mode, dragging moves the structure around the view window.

Interactive Cutting Plane Viewer

To access the interactive cutting plane viewer:

1. Choose **View > Interactive Mode > Cutting Plane**, or click the corresponding toolbar button.
2. Select either **Rotate X, Drag Y, or Rotate Z** to control the location of the cut plane. Enter the rotation or drag amount in the respective field or operate the dials with the mouse.

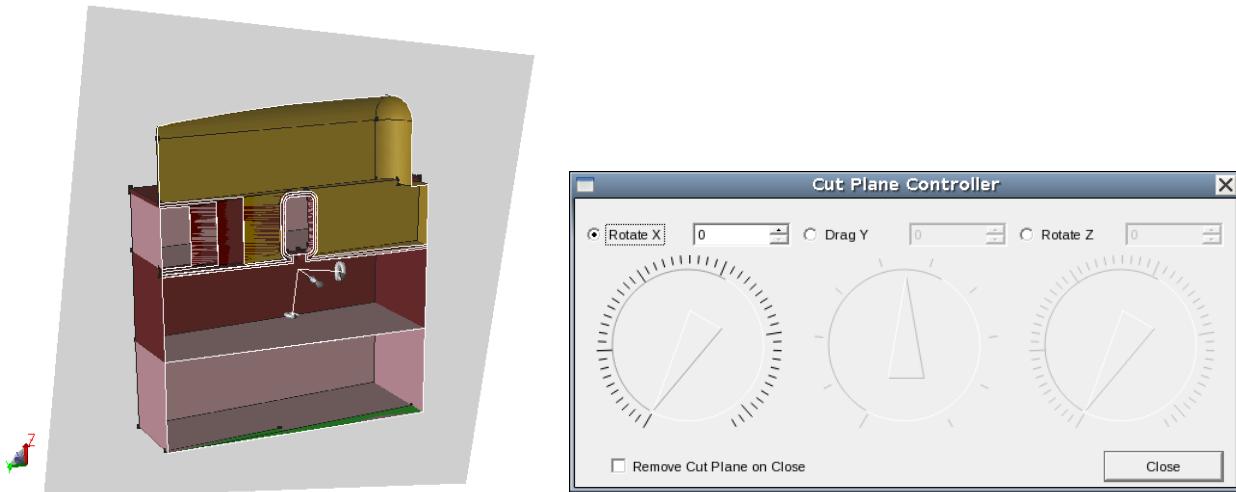
Alternatively, the cut plane can be manipulated directly in the view window by clicking the respective handles for rotating and dragging.

Chapter 3: Controlling Views

Perspective and Orthographic Views in Three Dimensions

3. To suppress the display of the cut plane, either select **Remove Cut Plane on Close** and click **Close**, or toggle the Cutting Plane toolbar button.

Figure 6 *Interactive cutting plane viewer: structure with cut plane (left) and Cut Plane Controller dialog box (right)*



Perspective and Orthographic Views in Three Dimensions

Three-dimensional objects are displayed using either orthographic projection (more distant objects are drawn with their true height and width as closer objects) or perspective (the height and width of more distant objects are reduced according to their distance from the viewer).

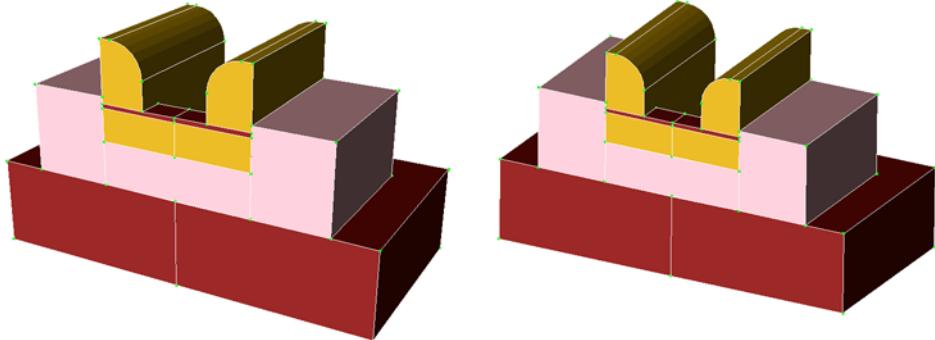
To switch between the perspective and orthographic views when displaying 3D objects:

- Choose **View > Perspective**.

Note:

Some GUI-supported operations such as 3D object creation (cube, cylinder, sphere) will change the view automatically to orthographic view.

Figure 7 (Left) Perspective view and (right) orthographic view



Selecting the DATEX Color Scheme

The `datexcodes.txt` file contains different color schemes for each DATEX material: Vivid and Classic.

By default, Sentaurus Structure Editor uses the Vivid color scheme to assign the color for each DATEX material for rendering. This means the **View > Vivid Colors (DATEX)** command is selected by default, indicated by a check mark.

To change to the Classic color scheme, choose **View > Vivid Colors (DATEX)**, so there is no check mark.

The selected color scheme is saved in the Sentaurus Structure Editor `.qt` configuration file. Therefore, when you exit and start a new session of Sentaurus Structure Editor, the color setting is synchronized automatically with the configuration file setting.

You can use the `-defaultGUI` command-line option to restore the DATEX color setting to the Vivid color scheme (see [Table 1 on page 38](#)).

In addition, you can use the `sde:change-datex-color-scheme` Scheme extension to select the DATEX color scheme and the `sde:get-datex-color-scheme` Scheme extension to retrieve the current setting of the color scheme.

Selecting the Rendering Mode

To access the different rendering modes:

1. Choose **View > Render**.
2. Select a rendering mode, or click the corresponding toolbar button (see [Table 5 on page 46](#)).

Chapter 3: Controlling Views

Selecting the Rendering Mode

The available rendering modes are illustrated here using the following example structure:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 0.8 1.0)
    "Silicon" "R.Substrate")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-cuboid (position 0.0 0.0 1.0) (position 0.5 0.8 1.02)
    "Oxide" "R.Gox")
(sdegeo:create-cuboid (position 0.0 0.0 1.02) (position 0.3 0.8 1.5)
    "PolySilicon" "R.Poly")
(sdegeo:set-default-boolean "BAB")
(sdegeo:create-cuboid (position 0.0 0.0 1.02) (position 0.5 0.8 1.4)
    "Nitride" "R.Spacer")
(sdegeo:fillet (find-edge-id (position 0.5 0.4 1.4)) 0.18)
(sde:setrefprops 0 30 0 0)
```

Facets

The faceted model shows a triangulated view in two dimensions and a triangulated surface tessellation in three dimensions. When exporting the boundary file, the 2D facetting algorithm extracts the boundary edges from the tessellated view. In three dimensions, the triangular elements become part of the polyhedral boundary representation. When changing the surface refinement properties, the faceted view always reflects the triangulation that will be used when exporting a boundary file.

Wireframe

This rendering mode shows the boundary wires only.

Flat Shaded

A simple shaded mode where the surface normals are taken from the tessellated (triangulated) mode. Each surface element has a uniform color.

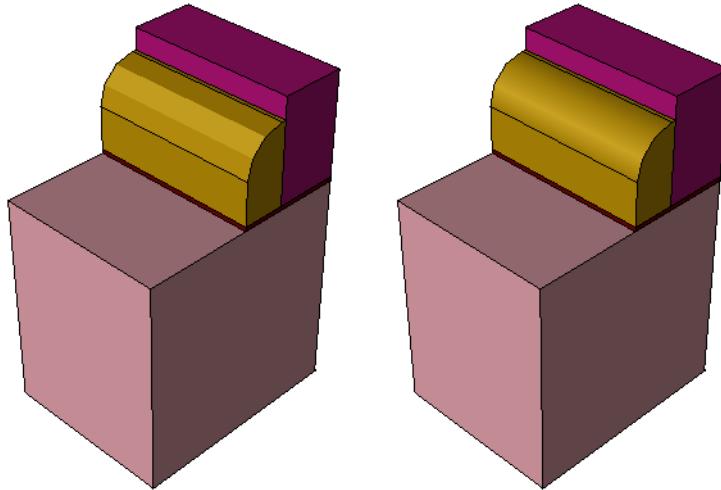
Gouraud Shaded

In this shading method, the surface normals are interpolated (using a simple linear interpolation) between the neighboring surface triangles. In this way, a continuously shaded view is generated.

Chapter 3: Controlling Views

Selecting the Rendering Mode

Figure 8 Rendering modes: (left) flat shaded and (right) Gouraud shaded



Hidden Line

This mode generates a 3D view in which nonvisible edges are hidden. Additional silhouette edges are added to the view for the correct visualization.

Wireframe or Hidden Line with Face Parameter Lines

To better visualize faces in Wireframe mode or Hidden Line mode, additional face parameter lines can be displayed as well. In this mode, each face is ‘decorated’ with a given number of parametric lines for each orthogonal direction. The default number is 3.

The Scheme commands for displaying face parameter lines are:

```
(view:display-param-lines #t)
; Switches on or off parameter lines for visualization
(option:set "u_param" 5) ; Number of parameter lines in u direction
(option:set "v_param" 5) ; Number of parameter lines in v direction
(render:rebuild)
```

This option is not available from the GUI.

Note:

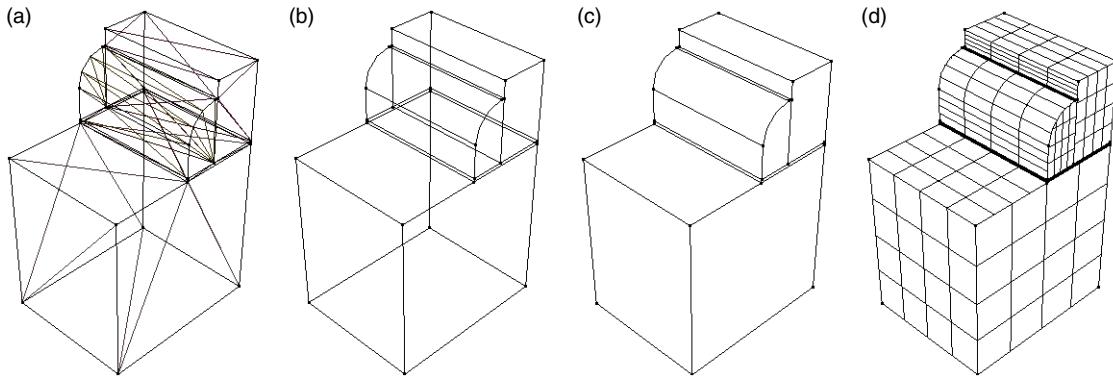
Apart from these rendering modes, other Scheme commands for changing the rendering are available:

```
(view:set-point-size 0) ; Switches on and off the vertex markers.
(render:rebuild) ; Rebuilds the graphics scene.
```

Chapter 3: Controlling Views

Showing and Hiding Coordinate Axes

Figure 9 Rendering modes: (a) facets, (b) wireframe, (c) hidden lines, and (d) hidden lines with face parameter lines



Showing and Hiding Coordinate Axes

To switch between the display of the coordinate axes in the lower-right corner of the main window:

- Choose **View > Show Axes**.

View Orientation

The camera position and orientation can be selected interactively. The resulting orientation of the coordinate axes displayed are given as follows:

Iso	3D isometric view
XY	X to the right, Y up
XZ	X to the right, Z up
YZ	Z to the left, Y up
X(-Y)	X to the right, Y down

To select the camera view:

- Choose **View > Camera Views**, or click the corresponding toolbar button.

Displaying Grid Lines

To display the grid lines:

1. Choose **View > Grid**, or click the corresponding toolbar button.
2. Enter the grid line spacing along the x axis (**Grid Width**) and along the y axis (**Grid Height**). Optionally, the width of the grid line and the line pattern can be selected.
3. Click **Apply** to activate the changed settings.
4. Click **Show** to display the grid lines, or click **Hide** to suppress the display of the grid lines.

Note:

The grid lines are always shown in the current work plane. Grid lines along the z-axis are not supported.

Displaying Rulers

To switch between the display of rulers in the xy coordinate directions:

- Choose **View > Show Ruler**, or click the corresponding toolbar button.

Note:

The rulers displayed in the GUI follow the dimensionality of the model. If only 2D bodies are defined, rulers are drawn in the xy coordinate directions. For 3D models, a ruler is also drawn in the z-axis coordinate direction. In addition to these rulers, you can create rulers using the `sde:draw-ruler` Scheme extension (see [sde:draw-ruler on page 455](#)). These user-defined rulers are displayed in the GUI automatically. You can hide user-defined rulers using the `sde:hide-ruler` Scheme extension (see [sde:hide-ruler on page 471](#)).

In two dimensions, rulers in the xy coordinate directions are placed such that they cross in the upper-left corner when using the xy view orientation. Using other view orientations might make it difficult to read the tick labels of the rulers (see [View Orientation on page 70](#)).

Scaling the View

You can scale the view (uniform or nonuniform). Scaling the view does not change the model coordinates; it is applied only to the graphical scene.

Chapter 3: Controlling Views

Visualizing Selected Geometric Objects

To scale the view:

1. Choose **View > View Scaling**.
2. Enter a view scaling factor for each of the coordinate axis.

The corresponding Scheme command is:

```
(sde:scale-scene x-factor y-factor z-factor)
```

For example, to scale the view by 2 in the y-direction, use `(sde:scale-scene 1 2 1)`. The command `(sde:scale-scene 1 1 1)` restores the original uniform unscaled model view.

Visualizing Selected Geometric Objects

To view geometric entities associated with a selected material, or selected regions, masks, contacts:

- Choose **View > Entity Viewer**.

The Entity Viewer is used to switch on and off the display of all different entity types (body, face, edge, vertex, and other), masks, contacts, regions, and materials. Each can be shown or hidden, individually or in groups, using the **Show** and **Hide** buttons. To restrict the display to only the selected entities, click **Exclusive**.

To see through the selected entities, click the **Translucent** button. To switch off translucency for the selected entities, click **Opaque**.

Any additions, changes, and deletions to entities made through GUI actions in the view window or through Scheme commands are reflected immediately in the Entity Viewer.

Two-dimensional or three-dimensional contacts can be visualized as separate objects. To see only a given contact, highlight the contact name in the Entity Viewer and click **Exclusive**.

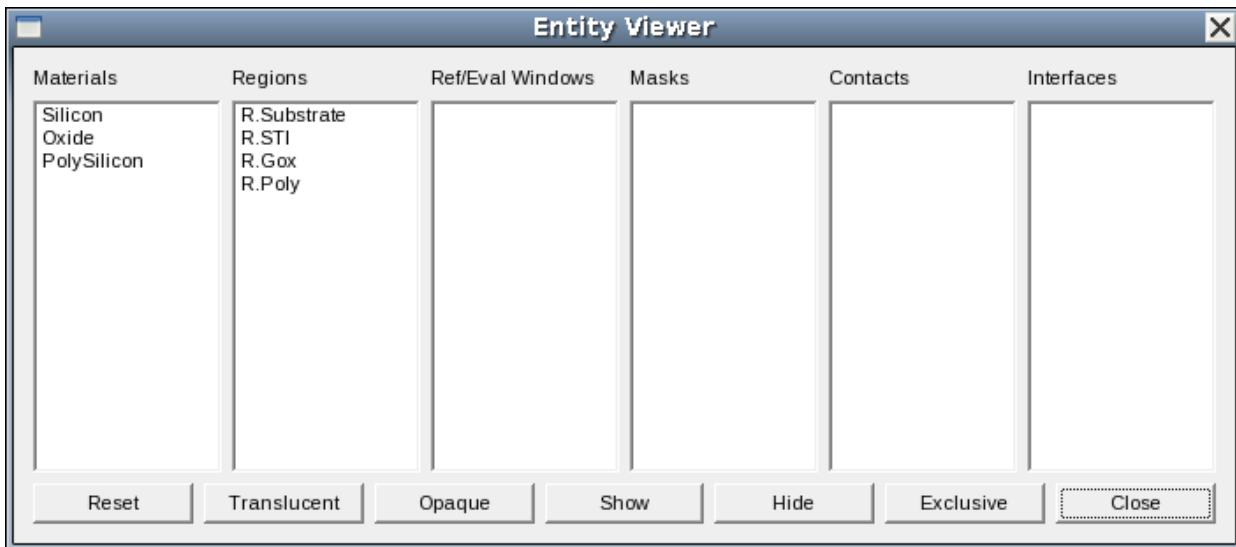
Related Scheme commands to display, hide, or show exclusively a given contact, respectively, are:

```
(sde:show-contact contactname)
(sde:hide-contact contactname)
(sde:xshow-contact contactname)
```

Chapter 3: Controlling Views

Quick Access to Placements, Refinements, and Doping Profiles

Figure 10 Entity Viewer



Quick Access to Placements, Refinements, and Doping Profiles

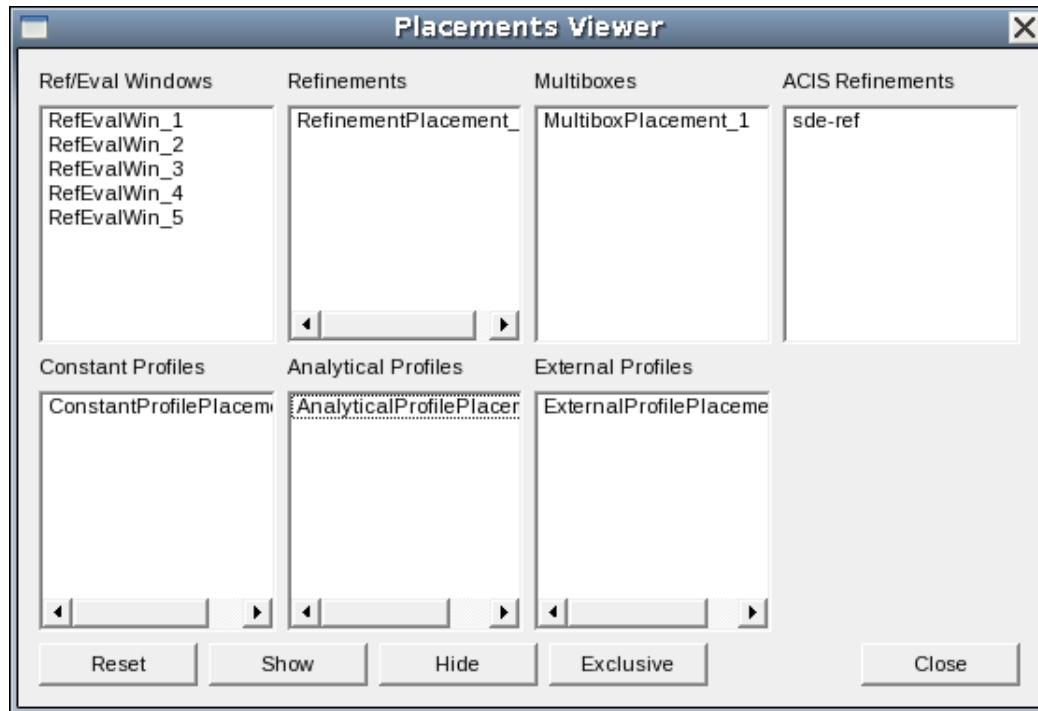
To list and access all currently defined Ref/Eval windows, refinements, and doping profiles:

1. Choose **View > Placements Viewer**.
2. Double-clicking any option opens the corresponding dialog box to view and edit its properties. Additions, changes, and deletions to placements made through GUI actions in the view window or through Scheme commands are reflected immediately in the Placements Viewer.
3. To visualize Ref/Eval windows, select one or more Ref/Eval window and click **Show**.
Click **Exclusive** to display the item while suppressing the display of all other items.
Click **Hide** to suppress the display of the selected item. (These two buttons only apply to Ref/Eval windows.)

Chapter 3: Controlling Views

Visualizing the Internal Entity Representation

Figure 11 Placements Viewer



Visualizing the Internal Entity Representation

To view the internal representation of entities, choose **View > Topology Browser**.

Sentaurus Structure Editor represents the topology of a structure as a hierarchy of entities. On the top of the hierarchy are the bodies, which consist of lumps, shells, faces, loops, coedges, edges, and finally vertices. When working with Sentaurus Structure Editor, mainly bodies, faces, edges, and vertices are important.

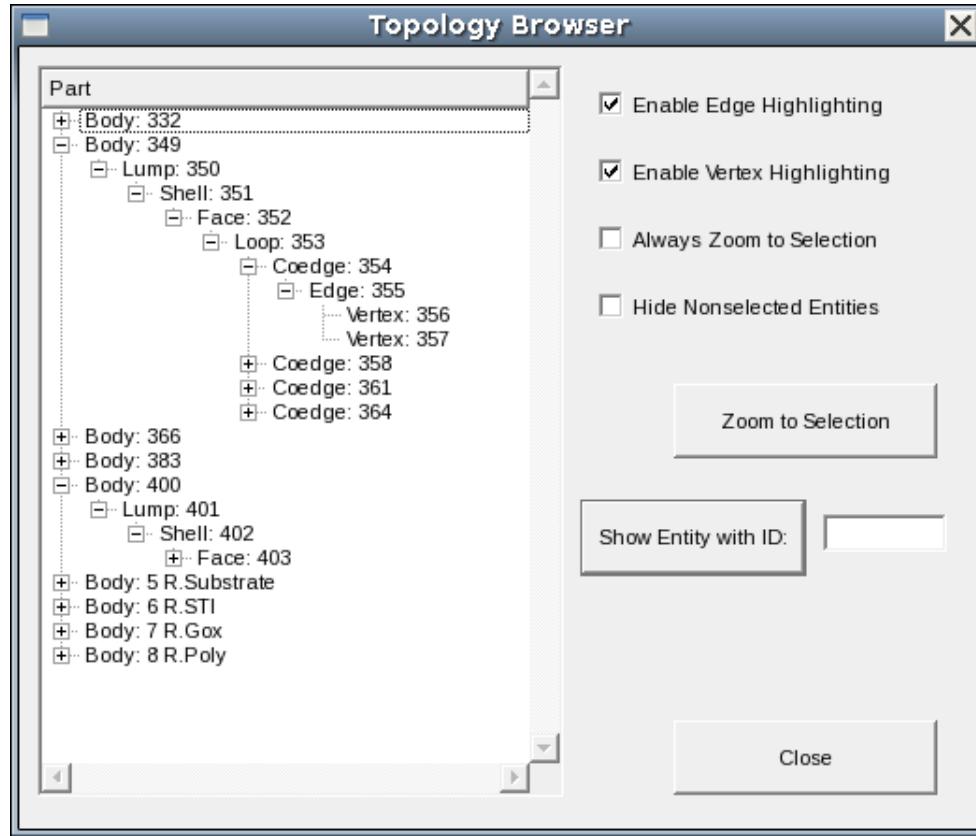
The Topology Browser displays a tree of entities and their relationships, as used internally by Sentaurus Structure Editor. Use this tool to determine entity numbers and relationships. The graphical selection of entities in the main window of Sentaurus Structure Editor results in the highlighting of the corresponding entity in the Topology Browser. Similarly, the selection of an entity in the Topology Browser results in the highlighting of the entity in the main window.

The Topology Browser also displays the doping/refinement/submesh (DRS)–related Ref/Eval windows as geometric objects.

Chapter 3: Controlling Views

Visualizing the Internal Entity Representation

Figure 12 Topology Browser



4

Generating Geometric Structures

This chapter provides details about creating a variety of structures and presents examples that illustrate various commands.

Modeling Unit and Modeling Range

The geometry operations in Sentaurus Structure Editor are based on the ACIS geometry kernel. Most geometric calculations are performed using finite-precision arithmetic and tolerances. You must be aware of two tolerance values because they determine the suggested modeling unit:

- `resabs` (default: 1e-6) is the tolerance used when comparing two positions in the 3D model space. If two positions are within `resabs` of each other, they can be considered in the same position.
- `resnor` (default: 1e-10) is the tolerance used when comparing the directions of two vectors. If the directions of two vectors are within `resnor` of each other, the vectors can be considered to be parallel.

Basically, the modeling range is the ratio of these two values (`resabs/resnor`).

Geometry modeling in Sentaurus Structure Editor is unitless. This means that you can choose any unit for model representation. However, the modeling range and the smallest representable feature will depend on the selected modeling unit.

For example, if you want to use micrometer as the modeling unit and a modeling range that is 100 µm, then the smallest representable feature will be $\text{resnor} \times 100 = 10^{-8}$ µm, which is 10^{-5} nm. If the modeling unit and the modeling range are selected to be 1 m, then the smallest representable feature will be 10^{-10} m, which is 10 nm. From this example, it is clear that it would not be practical to select meter as the modeling unit, since the TCAD devices could not be represented in Sentaurus Structure Editor based on the tolerance values that are used for the geometric calculations. Selecting micrometer as the modeling unit would be a better choice for TCAD devices. This means that, in this case, the smallest representable geometric feature is approximately 10^{-5} nm.

Chapter 4: Generating Geometric Structures

Creating a New Structure

Based on experience in nanodevice modeling, it is suggested that you use nanometer as the modeling unit when such devices are being modeled. This will result in a smaller modeling range but a larger precision, which might be preferable. It is best to select the modeling unit in such a way that it will provide the largest precision within the necessary modeling range.

If there is a modeling unit mismatch between the different TCAD tools, then an additional scaling can be applied to the tessellated boundary file created by Sentaurus Structure Editor and to the mesh command file. This operation does not introduce any modeling error, since scaling is performed directly on the files, and no additional geometric operations are applied to the model.

Creating a New Structure

Before creating a new structure, it is good practice to reset Sentaurus Structure Editor.

To reset Sentaurus Structure Editor, choose **File > New** or press **Ctrl+N**.

The corresponding Scheme command is:

```
(sde:clear)
```

Note:

This command restores the default settings of the graphical user interface (GUI), clears and reinitializes the geometry database, and clears all doping-related and refinement-related data. It is recommended as good practice to start each Scheme script with this command.

Setting Interactive Work Preferences

This section discusses the interactive work preferences that you can set.

Exact Coordinates

In Sentaurus Structure Editor, geometric objects can be drawn free hand. However, for most applications, it is convenient (and sometimes necessary) to specify explicitly the coordinates of the object, for example, to allow for precise alignment of different regions.

To activate exact coordinates, choose **Draw > Exact Coordinates**, or click the corresponding toolbar button (see [Table 7 on page 46](#)).

Chapter 4: Generating Geometric Structures

Setting Interactive Work Preferences

In Exact Coordinates mode, after drawing any 2D shape, or after adding or moving vertices, a dialog box is displayed that shows the vertex coordinates of the new shape. The dialog box allows you to edit these values.

Snapping

Snapping can be used as an alternative or a convenient augmentation of the Exact Coordinates mode for creating 2D regions (rectangles, circles, ellipses, and so on). The different snapping modes are activated interactively when drawing shapes by pressing the following keys (or click the corresponding toolbar buttons):

- **Grid Snapping:** To activate grid snapping, press the G key or click the corresponding toolbar button (see [Table 10 on page 47](#)). The pointer jumps from grid point to grid point. (To visualize the grid, choose **View > Grid**.)
- **Vertex Snapping:** To snap the pointer to the nearest vertex of an existing shape, press the V key or click the corresponding toolbar button (see [Table 10](#)). When the pointer is moved, it jumps from vertex to vertex. A click will insert the new vertex to be exactly on a vertex.
- **Edge Snapping:** To snap the pointer to the nearest edge of an existing shape (it will slide along this edge), press the E key or click the corresponding toolbar button (see [Table 10](#)). A click will insert the new vertex to be exactly on the edge.
- **Free Drawing:** To return to the default free-drawing mode, press the F key or click the corresponding toolbar button (see [Table 10](#)).

Note:

You can use the Exact Coordinates mode concurrently with all snapping options.

Active Material

To select the active material to be used for subsequent actions, use the Material list.

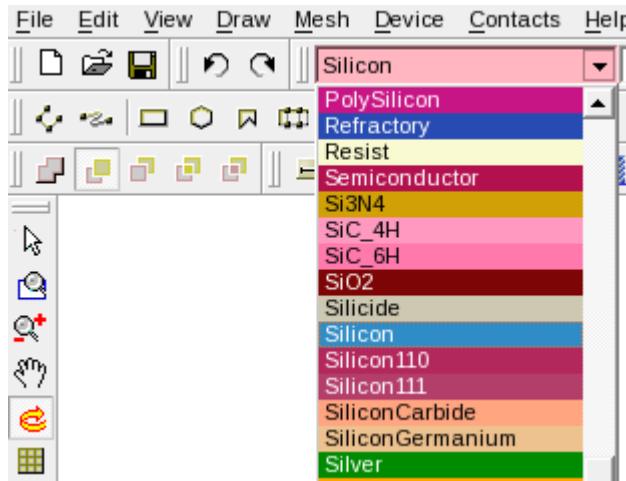
Note:

The `datexcodes.txt` file controls the material list. The default location is `$STROOT/tcad/$STRELEASE/lib/datexcodes.txt`. New materials can be defined by creating a local version of this file in the current working directory.

Chapter 4: Generating Geometric Structures

Setting Interactive Work Preferences

Figure 13 Material list: select active material for all subsequent actions



Naming Regions

When creating a new region, Sentaurus Structure Editor automatically assigns a default region name. The default region names follow the sequence `region_1`, `region_2`, `region_3`, and so on. To be prompted for a custom region name after creating a new region interactively, deactivate the automatic region-naming feature.

To switch automatic region-naming on or off, choose **Draw > Auto Region Naming**.

Region names can be changed later (see [Changing the Name of a Region on page 144](#)). Within Sentaurus Structure Editor, multiple geometric objects can share the same region name. However, when saving the final boundary, disjoint regions must have different region names. If the region-naming does not satisfy this criterion, Sentaurus Structure Editor automatically modifies the region names by adding the suffix `_lump%N` to each disjoint region (where `N` is a lump counter). Choose **Edit > Separate Lumps** to explicitly force this region-renaming step.

Overlap Behavior

In most applications, a device consists of multiple regions. When creating a device with multiple regions in Sentaurus Structure Editor, later-added regions might intersect existing regions. If this occurs, a predefined scheme is used to resolve the overlapping regions.

To select the overlap resolution behavior, choose **Draw > Overlap Behavior**, or click the corresponding toolbar button (see [Table 12 on page 48](#)).

Chapter 4: Generating Geometric Structures

Setting Interactive Work Preferences

The types of overlap resolution behavior available in Sentaurus Structure Editor are:

Merge

Old and new structures merge, and all boundary lines between them are dissolved. The resulting merged structure takes on the region name and material of the last-drawn structure. The corresponding Scheme command is:

```
(sdegeo:set-default-boolean "AB")
```

New Replaces Old

The new structure is created as drawn, and the old structure is adapted. The part of the old structure that overlaps with the new one is deleted, and its boundary is replaced by the boundary of the new structure. Two separate regions remain. The corresponding Scheme command is:

```
(sdegeo:set-default-boolean "ABA")
```

Old Replaces New

The new structure is created, but it is adapted so that the old one remains unchanged. The part of the new structure that overlaps with the old one is deleted, and its boundary is replaced by the boundary of the old structure. Two separate regions remain. The corresponding Scheme command is:

```
(sdegeo:set-default-boolean "BAB")
```

New Overlaps Old

The new structure is created as drawn, and the old structure is adapted. The part of the old structure that overlaps with the new structure is deleted. However, the boundary edges of the old structure do not dissolve. Three regions result: original, new, and overlap, where the overlap region is named `region_1region_2`, that is, the concatenation of the original and new region names. The corresponding Scheme command is:

```
(sdegeo:set-default-boolean "ABiA")
```

Old Overlaps New

The new structure is created, but it is adapted so that the old one remains unchanged. The part of the new structure that overlaps the old one is deleted. However, the boundary edges of the new structure do not dissolve. Three regions result: original, new, and overlap, where the overlap region is named `region1region2`, that is, the concatenation of the original and new region names. The corresponding Scheme command is:

```
(sdegeo:set-default-boolean "ABiB")
```

Unresolved Overlaps

An additional option "xx" is available. The `(sdegeo:set-default-boolean "xx")` command switches off the automatic overlap region handling, thereby allowing the

Chapter 4: Generating Geometric Structures

Drawing Basic 2D Shapes

creation of overlapping regions. This option should be used only by experienced users. The overlaps must be resolved manually later; otherwise, the exported final boundary will contain an invalid topology. This option is not available from the GUI.

Current Overlap Behavior

The Scheme command (`sdegeo:get-default-boolean`) returns information about the currently active overlap resolution behavior.

Removing Overlaps

If the model contains overlapping regions, the overlaps can be removed by using either explicit Boolean operations (`sdegeo:bool-intersect`, `sdegeo:bool-subtract`, `sdegeo:bool-unite`), or the `remove-body-ABA` or `remove-body-BAB` command.

Note:

Some geometric operations might create overlaps, which can be difficult to detect, especially for larger 3D models. The overlaps must be removed from the model before the tessellated boundary file is saved for meshing. The `sdegeo:check-overlap` function can be used to find the possible overlaps automatically.

Drawing Basic 2D Shapes

This section discusses how to draw basic 2D shapes.

Drawing Rectangles

To draw a rectangle:

1. Choose **Draw > 2D Create Tools > Rectangular Region**, or click the corresponding toolbar button (see [Table 9 on page 47](#)).
2. Drag to draw the diagonal of the rectangle in the view window.
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the first vertex and second vertex of the diagonal.

The corresponding Scheme command is [`sdegeo:create-rectangle` on page 625](#). For example:

```
(sdegeo:create-rectangle (position 0 0 0) (position 2 1 0) "Silicon"  
"R.Substrate")
```

Drawing Regular Polygons

To draw a regular polygon:

1. Choose **Draw > 2D Create Tools > Regular Polygonal Region**, or click the corresponding toolbar button (see [Table 9 on page 47](#)).
2. Drag to draw the radius of the regular polygon in the view window. By default, a ten-sided polygon is created.
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the center point, the radius, the number of sides, as well as the starting angle.

The corresponding Scheme command is [sdegeo:create-reg-polygon on page 626](#). For example:

```
(sdegeo:create-reg-polygon (position 0 0 0) 1.0 3 30 "Nitride"  
"R.TriangleSpacer")
```

Drawing Polygons

To draw a general polygon:

1. Choose **Draw > 2D Create Tools > Polygonal Region**, or click the corresponding toolbar button (see [Table 9 on page 47](#)).
2. Click at each vertex of the polygon in the view window. To finish drawing the polygon, click the middle mouse button. (Press both buttons when using a two-button mouse.) The polygon closes automatically.
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates for each vertex.

The corresponding Scheme command is [sdegeo:create-polygon on page 620](#). For example:

```
(sdegeo:create-polygon (list (position 0 0 0) (position 0.5 -1 0)  
(position 1 -1 0) (position 1.5 0 0) (position 0 0 0))  
"PolySilicon" "R.TaperedPoly")
```

Drawing Circles

To draw a circle:

1. Choose **Draw > 2D Create Tools > Circular Region**, or click the corresponding toolbar button (see [Table 9 on page 47](#)).
2. Drag to draw the radius of the circle in the view window.

Chapter 4: Generating Geometric Structures

Drawing Basic 2D Shapes

3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the center point and the radius.

The corresponding Scheme command is [sdegeo:create-circle on page 608](#). For example:

```
(sdegeo:create-circle (position 0 0 0) 0.01 "Silicon" "R.Nanowire")
```

Drawing Ellipses

To draw an ellipse:

1. Choose **Draw > 2D Create Tools > Elliptical Region**, or click the corresponding toolbar button (see [Table 9 on page 47](#)).
2. Drag to draw the major axis in the view window. Then, release the mouse button and move the pointer to select the width of the ellipse.
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the center point, the endpoint of the major axis, as well as the ratio between the major and the minor axes.

The corresponding Scheme command is [sdegeo:create-ellipse on page 614](#). For example:

```
(sdegeo:create-ellipse (position 0 0 0) (position 1 0 0) 0.5  
"Silicon" "R.Ellipse")
```

Note:

The ratio can be greater than 1, in which case, the minor axis will actually be larger than the other axis (which was referred to above as the *major axis*).

Drawing Ruled Regions

A ruled region is a shape that is defined by two edges. The region is created by connecting the endpoints of the two edges.

To draw a ruled region:

1. From the Selection Level list, select **Select Edge**.
2. Choose **Draw > 2D Create Tools > Ruled Region**.
3. Select the edge and hold the Shift key to select the second edge.

The corresponding Scheme command is [sdegeo:create-ruled-region on page 627](#). For example:

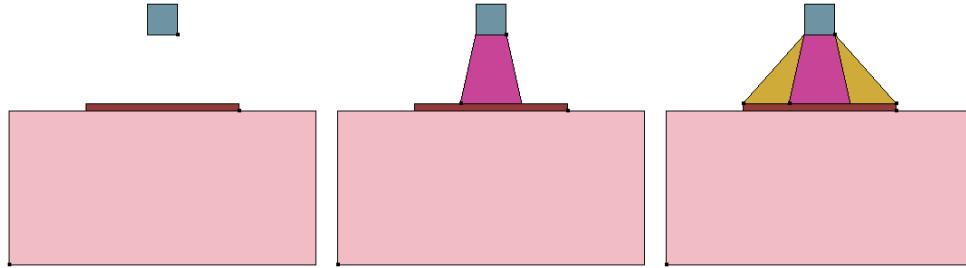
```
(sdegeo:create-rectangle (position -1.0 0.0 0) (position 1.0 1.0 0)  
"Silicon" "R.Substrate")  
(sdegeo:create-rectangle (position -0.5 0.0 0) (position 0.5 -0.05 0)
```

Chapter 4: Generating Geometric Structures

Drawing Basic 2D Shapes

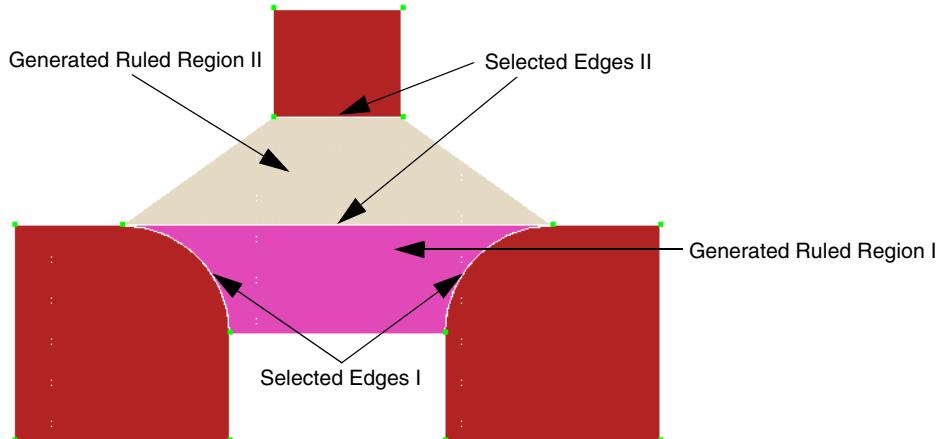
```
"Oxide" "R.Gox")
(sdegeo:create-rectangle (position -0.1 -0.5 0) (position 0.1 -0.7 0)
  "Aluminum" "R.GateCont")
(sdegeo:insert-vertex (position -0.2 -0.05 0))
(sdegeo:insert-vertex (position 0.2 -0.05 0))
(sdegeo:create-ruled-region (list
  (car (find-edge-id (position 0.0 -0.5 0)))
  (car (find-edge-id (position 0.0 -0.05 0)))) "PolySilicon"
  "R.PolyGate")
(sdegeo:create-ruled-region (list
  (car (find-edge-id (position -0.15 -0.275 0))))
  (car (find-edge-id (position -0.35 -0.05 0)))) "Nitride" "R.SpacerL")
(sdegeo:create-ruled-region (list
  (car (find-edge-id (position 0.15 -0.275 0))))
  (car (find-edge-id (position 0.35 -0.05 0)))) "Nitride" "R.SpacerR")
```

Figure 14 Generating ruled regions using linear edges: (left) original structure, (middle) after adding the first ruled region, and (right) after adding the remaining two ruled regions



Ruled regions can be created only for 2D models. The edges that are used for creating the ruled region do not have to be linear. Figure 15 shows such a case. The inserted ruled region (ruled region I) was created between two circular edges, which were created by a filleting operation.

Figure 15 Generating a ruled region using curved edges



Drawing Other Basic 2D Shapes

Other basic 2D shapes are available using Scheme extensions such as `sdegeo:create-triangle`.

See the [Index of Scheme Extensions](#) for details.

Editing 2D Shapes

This section describes various editing operations on 2D shapes.

Adding a Vertex

To draw a vertex:

1. Choose **Edit > 2D Edit Tools > Add Vertex**, or click the corresponding toolbar button (see [Table 8 on page 47](#)).
2. Click a position on an edge where the vertex is to be placed.

The corresponding Scheme command is:

```
(sdegeo:insert-vertex position)
```

For example:

```
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
  "Silicon" "region_1")
(sdegeo:insert-vertex (position 0.5 1.0 0.0))
```

Vertices are added to existing 2D regions in such a way that model conformity is maintained. For regions sharing a common edge or an edge segment, the vertex is inserted onto both edges, splitting the edges. Added vertices always are inserted onto the closest edge, by projecting the inserted vertex to the nearest edge.

Note:

This operation is available only for 2D regions. Use the Wireframe mode to visualize the new vertex (see [Table 5 on page 46](#)). The Exact Coordinates mode can be used to precisely control the new location; however, snapping is not supported.

Moving a Vertex

To move a vertex:

1. Choose **Edit > 2D Edit Tools > Move Vertex**, or click the corresponding toolbar button (see [Table 8 on page 47](#)).
2. Drag the vertex to the required position.

The corresponding Scheme command is:

```
(sdegeo:move-vertex vertex position)
```

For example:

```
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silicon"  
"region_1")  
(sdegeo:move-vertex (car (find-vertex-id (position 0 0 0)))  
 (position 0.25 0.25 0))
```

To move several vertices:

1. Choose **Edit > 2D Edit Tools > Move Vertices**, or click the corresponding toolbar button (see [Table 8 on page 47](#)).
2. Select the vertices.
3. When finished, drag any of the selected vertices to move all of them to the required position.

The corresponding Scheme command is:

```
(sdegeo:move-vertex vertex-list gvector)
```

For example:

```
(define BODY (sdegeo:create-reg-polygon (position 0 0 0) 1 10 0  
"Silicon")  
(sdegeo:move-vertex (sde:window-select-2d 0 -1.1 1.1 1.1 BODY  
"vertex") (gvector 1 0 0))
```

Note:

This operation is available only for 2D regions. Only vertices that connect linear line segments can be moved. Vertices on curved edges cannot be moved. The Exact Coordinates mode can be used to precisely control the new location; however, snapping is not supported.

The function can be used to move the vertices of some 2D DRS Ref/Eval windows (line, rectangle, polygon).

Moving a vertex is allowed only to such an extent that the expected operation results in a topologically correct 2D region, that is, crossing edges are not allowed. If a vertex belongs to multiple regions, all regions are modified accordingly.

Moving Region Edges

To move a region edge:

1. Choose **Edit > 2D Edit Tools > Move Edge**, or click the corresponding toolbar button (see [Table 8 on page 47](#)).
2. Select a linear edge, and drag the edge to the required position.

The corresponding Scheme command is:

```
(sdegeo:move-edge edgeid gvector)
```

For example:

```
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silicon"
  "region_1")
(sdegeo:move-edge (car (find-edge-id (position 1 0.5 0)))
  (gvector 0.5 0.5 0))
```

Note:

This operation is available only for 2D regions. Only linear edges can be moved and only when all connecting edges are linear as well. The edges are moved like rigid bodies (that is, all connecting edges are moved as well). The Exact Coordinates mode can be used to precisely control the new location.

If the new location of the moved region is such that it overlaps other regions, the overlaps are removed automatically following the active Boolean overlap control rule.

Moving Regions

To move one or more regions:

1. Choose **Edit > 2D Edit Tools > Move Region**, or click the corresponding toolbar button (see [Table 8 on page 47](#)).
2. Select one region (or more regions by holding the Ctrl key). Release the Ctrl key, and drag the regions to the required position.

The corresponding Scheme command is:

```
(sdegeo:move-2d-regions body | body-list gvector)
```

Chapter 4: Generating Geometric Structures

Editing 2D Shapes

For example:

```
(define mb1 (sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
                                         "Silicon" "region_1"))
(sdegeo:move-2d-regions mb1 (gvector 1 1 0))
```

Note:

This operation is available only for 2D regions. The Exact Coordinates mode can be used to precisely control the new location.

If the new locations of the moved regions are such that they overlap other regions, then the overlaps are removed automatically following the active Boolean overlap control rule.

Deleting Vertices

To delete one or more vertices:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Vertex**.
3. Select the vertex and hold the Shift key to select additional vertices, or drag a box around a set of vertices.
4. Choose **Edit > 2D Edit Tools > Delete Vertex**, or press the Delete key.

The corresponding Scheme command is:

```
(sdegeo:delete-vertices vertex-list)
```

Example 1 with an explicit vertex selection:

```
(sdegeo:create-reg-polygon (position 0 0 0) 1 6 0 "Silicon"
                             "region_1")
(sdegeo:delete-vertices (list
                         (car (find-vertex-id (position -1.0 0 0)))
                         (car (find-vertex-id (position 1.0 0 0)))))
```

Example 2 with a window vertex selection:

```
(sdegeo:create-polygon (list
                         (position 0.0 0.0 0) (position 1.0 0.0 0) (position 1.0 1.0 0)
                         (position 0.8 1.1 0) (position 0.6 0.9 0) (position 0.4 1.1 0)
                         (position 0.2 0.9 0) (position 0.0 1.0 0) (position 0.0 0.0 0))
                         "Silicon" "region_1")
(define VERTICES (sde:window-select-2d 0.1 0.8 0.9 1.2 "all" "vertex"))
(sdegeo:delete-vertices VERTICES)
```

Note:

This operation is available only for 2D regions.

Rounding

To round one or more corners of a 2D region:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Vertex**.
3. Select the vertex and hold the Shift key to select additional vertices, or drag a box around a set of vertices.
4. Choose **Edit > 2D Edit Tools > Fillet**.
5. Enter a fillet radius in the dialog box.

If the parameter `fillet-radius` was set before performing this operation (choosing **Edit > Parameters**), then this dialog box is suppressed, and the given value of `fillet-radius` is used instead.

The corresponding Scheme command is:

```
(sdegeo:fillet-2d vertex-list radius)
```

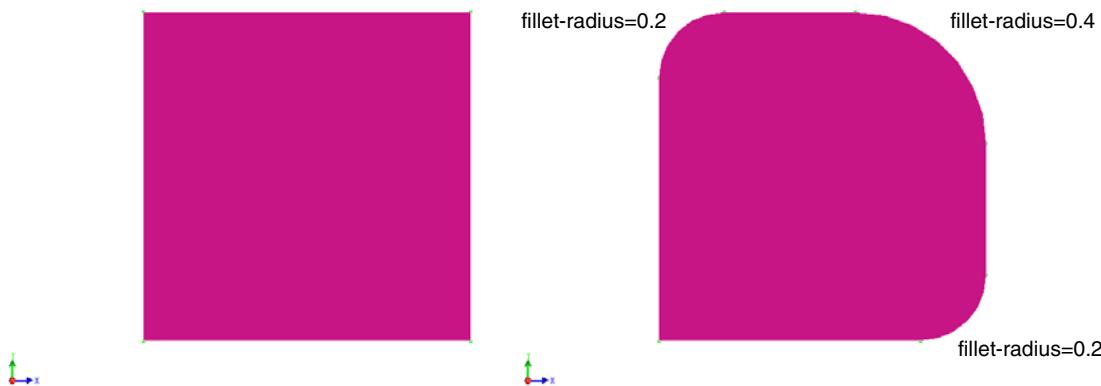
For example:

```
(sdegeo:create-rectangle (position 0 0 0.0) (position 1 1 0.0)
  "Silicon" "region_1")
(sdegeo:fillet-2d (list
  (car (find-vertex-id (position 1 0 0)))
  (car (find-vertex-id (position 0 0 0)))) 0.4)
```

Note:

If `fillet-radius` is too large such that two adjacent fillets overlap, the operation will fail.

Figure 16 Two-dimensional vertex rounding: (left) rectangle (1x1) and (right) rounded rectangle



Chamfering

To chamfer one or more corners of a 2D region:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Vertex**.
3. Select the vertex and hold the Shift key to select additional vertices, or drag a box around a set of vertices.
4. Choose **Edit > 2D Edit Tools > Chamfer**.
5. Enter a chamfer distance in the dialog box.

If the parameter `chamfer-dist` was set before performing this operation (choosing **Edit > Parameters**), then this dialog box is suppressed, and the given value of `chamfer-dist` is used instead.

Note:

The `chamfer-dist` cannot be larger than the smallest edge length of the adjacent edges of the selected vertices. For multiple vertices sharing the same edge, the distance must be smaller than half the length of the smallest such edge. Otherwise, the operation will fail.

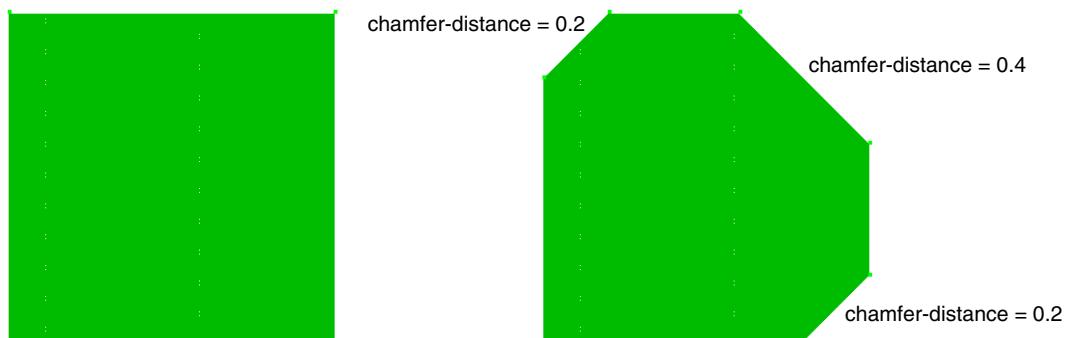
The corresponding Scheme command is:

```
(sdegeo:chamfer-2d vertex-list chamfer-dist)
```

For example:

```
(sdegeo:create-rectangle (position 0 0 0.0) (position 1 1 0.0)
  "Silicon" "region_1")
(sdegeo:chamfer-2d (list
  (car (find-vertex-id (position 1 0 0)))
  (car (find-vertex-id (position 0 0 0)))) 0.4)
```

Figure 17 Two-dimensional vertex chamfering: (left) rectangle (1x1) and (right) chamfered rectangle



Cutting

To cut out a rectangular part from a 2D structure:

1. Choose **Edit > 2D Edit Tools > 2D Cut**.
2. Drag to draw the diagonal of the rectangle in the view window.
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the first vertex and second vertex of the diagonal.

The corresponding Scheme command is:

```
(sdegeo:2d-cut position position)
```

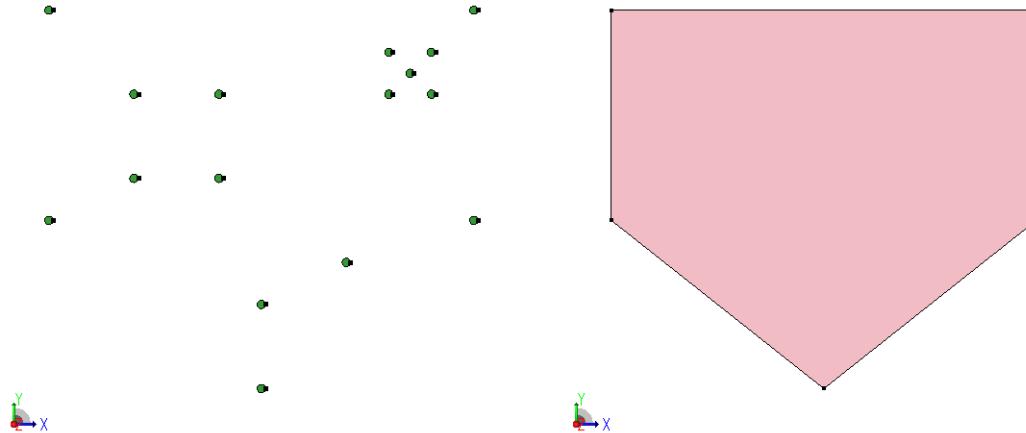
The argument list contains the two corner positions of the rectangular area. All regions outside of this rectangle are removed from the model. For example:

```
(sdegeo:create-circle (position 0 0 0) 1 "Silicon" "region_1")
(sdegeo:2D-cut (position 0 -1.1 0.0) (position 0.8 1.1 0.0))
```

Creating a Convex Hull

The Scheme extension sdegeo:chull2d can be used to compute the 2D convex hull of the specified position list. The position list of the computed convex hull is returned.

Figure 18 Creating a 2D convex hull: (left) original point set and (right) created convex hull



Splitting Structures

To split one or more 2D regions by a polygonal line:

1. Choose **Edit > 2D Edit Tools > Polygonal Region Split**, or click the corresponding toolbar button (see [Table 8 on page 47](#)).
2. Click at each vertex of the polygonal line. To finish drawing the polygonal line, click the middle mouse button. (Press both buttons when using a two-button mouse.)
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates for each vertex.

The first vertex and last vertex of the defined polygonal line snap to the closest edge or vertex of the existing model. Unique region names are given to the split-off parts of the original bodies.

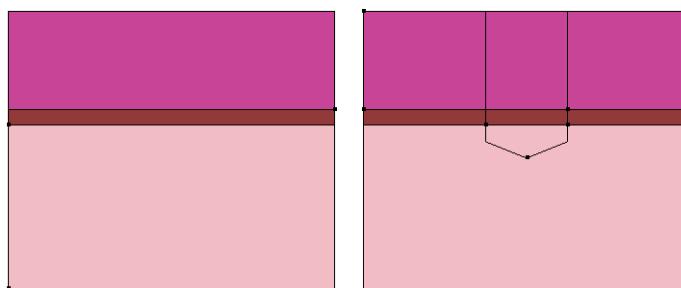
The corresponding Scheme command is:

```
(sdegeo:polygonal-split position-list)
```

For example:

```
(sdegeo:create-rectangle (position -1.0 0.0 0.0)
  (position 1.0 1.0 0.0) "Silicon" "region_1")
(sdegeo:create-rectangle (position -1.0 -0.1 0.0)
  (position 1.0 0.0 0.0) "SiO2" "region_2")
(sdegeo:create-rectangle (position -1.0 -0.1 0.0)
  (position 1.0 -0.7 0.0) "PolySi" "region_3")
(sdegeo:polygonal-split (list
  (position -0.25 -1 0) (position -0.25 0.1 0) (position 0 0.2 0)
  (position 0.25 0.1 0) (position 0.25 -1 0)))
```

Figure 19 Splitting a device using a polygonal line segment: (left) original structure and (right) after the split operation



Simplifying 2D Structures

Sentaurus Structure Editor has a set of functions to simplify 2D structures. Such boundary simplification is sometimes advantageous for structures generated by other tools such as Sentaurus Process. Examples of such simplifications are aligning vertices, removing collinear edges, and breaking nonaxis-aligned edges into axis-aligned *staircases*. In general, the subsequent meshing tool will generate grids with fewer nodes if the boundary has fewer points and if most edges are axis aligned.

Aligning Vertices

To align vertices to an axis horizontally (or vertically):

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Vertex**.
3. Select the vertex and hold the Shift key to select additional vertices, or drag a box around a set of vertices.
4. Choose **Edit > 2D Edit Tools > Align Vertices to Horizontal** (or **Align Vertices to Vertical**).
5. Click to select the horizontal (or vertical) alignment level.

The corresponding Scheme commands are:

```
(sdegeo:align-horizontal vertex-list yalign)
(sdegeo:align-vertical   vertex-list xalign)
```

For example:

```
(sdegeo:create-polygon (list
  (position 0.0 1.0 0) (position 1.0 1.0 0) (position 1.0 0.0 0)
  (position 0.8 0.1 0) (position 0.6 -0.1 0) (position 0.4 0.1 0)
  (position 0.2 -0.1 0) (position 0.0 0.0 0) (position 0.0 1.0 0))
  "Silicon" "region_1")
(define VERTICES (sde>window-select-2d -0.1 -0.2 1.1 0.2 "all"
  "vertex"))
(sdegeo:align-horizontal VERTICES 0.0)
```

To automatically align the selected vertices to the average y-position (or x-position), choose **Edit > 2D Edit Tools > Aut. Align Vertices to Horizontal** (or **Aut. Align Vertices to Vertical**).

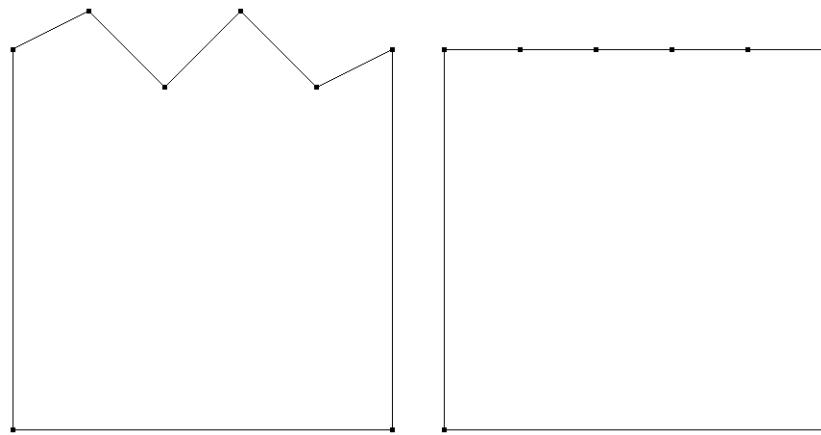
The corresponding Scheme commands are:

```
(sdegeo:align-horizontal-aut vertex-list)
(sdegeo:align-vertical-aut   vertex-list)
```

Chapter 4: Generating Geometric Structures

Editing 2D Shapes

Figure 20 Aligning vertices horizontally: (left) original structure and (right) after alignment. For better viewing, the structure is shown as a wireframe.



To align the selected vertices to an arbitrary line, choose Choose **Edit > 2D Edit Tools > Align Vertices to a Line**.

To define the line, click at the starting point and drag to the endpoint.

The corresponding Scheme command is:

```
(sdegeo:align-to-line vertex-list starting-position end-position)
```

For example:

```
(sdegeo:create-polygon (list
  (position 0.0 0.0 0) (position 1.0 0.0 0) (position 1.0 1.0 0)
  (position 0.8 1.1 0) (position 0.6 0.9 0) (position 0.4 1.1 0)
  (position 0.2 0.9 0) (position 0.0 1.0 0) (position 0.0 0.0 0))
  "Silicon" "region_1")
(define VERTICES (sde>window-select-2d -0.1 0.8 1.1 1.2 "all"
  "vertex"))
(sdegeo:align-to-line VERTICES (position 0.0 0.7 0)
  (position 1.0 1.3 0))
```

Note:

The command moves the vertices along the normal to the specified line.

Therefore, both the x-coordinates and y-coordinates can be altered by this operation.

When the vertices have been aligned, the now redundant vertices can be removed by using either vertex deletion (see [Deleting Vertices on page 88](#)) or edge removal (see [Merging Collinear Edges on page 95](#)).

Figure 21 Two-dimensional boundary simplification, vertical alignment of vertices, and removal of short edges

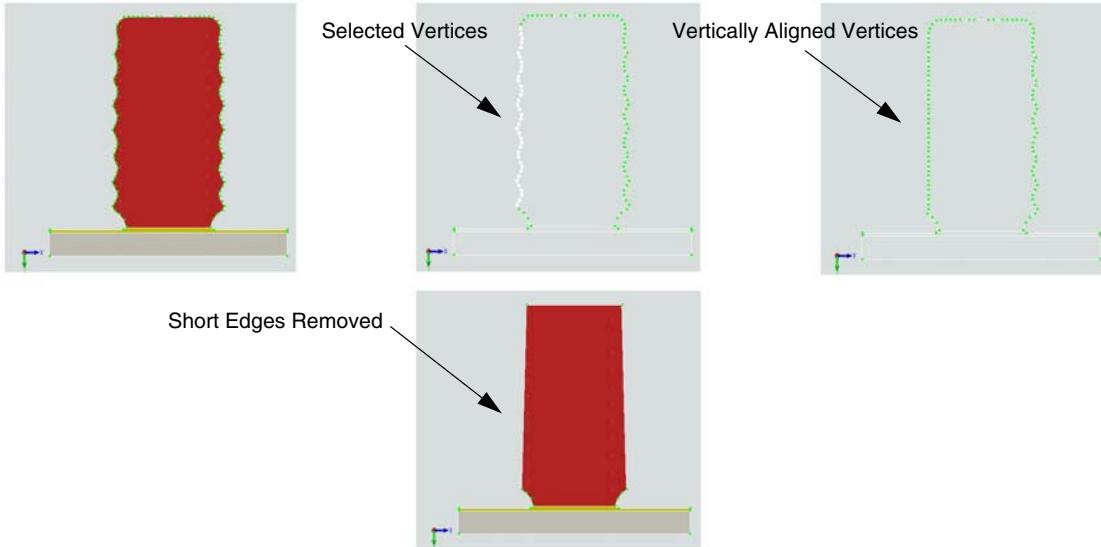
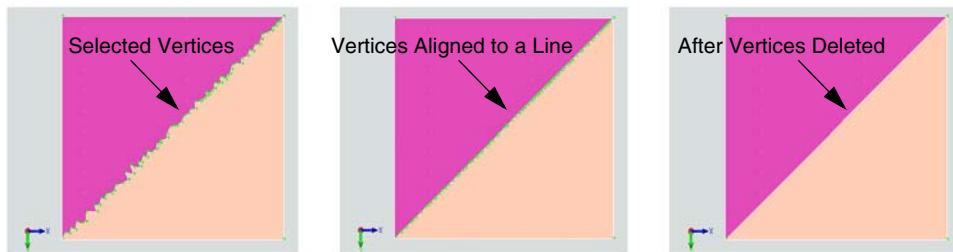


Figure 22 Two-dimensional boundary simplification, align to line, and vertex removal



Merging Collinear Edges

To merge collinear edges:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Edge**.
3. Select the edge and hold the Shift key to select additional edges, or drag a box around a set of edges.
4. Choose **Edit > 2D Edit Tools > Delete Collinear Edges**.

The corresponding Scheme command is:

```
(sdegeo:delete-collinear-edges edge-list)
```

Chapter 4: Generating Geometric Structures

Editing 2D Shapes

For example:

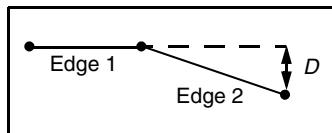
```
(sdegeo:create-rectangle (position 0.0 0.0 0) (position 1.0 1.0 0)
    "Silicon" "region_1")
(sdegeo:insert-vertex (position 0.3 0.0 0))
(sdegeo:insert-vertex (position 0.0 0.3 0))
(sdegeo:insert-vertex (position 0.0 0.6 0))
(define EDGES (sde>window-select-2d -0.1 -0.1 1.1 0.8 "all" "edge"))
(sdegeo:delete-collinear-edges EDGES)
```

This command removes edges only if they are exactly collinear. For ‘noisy’ boundaries:

1. Choose **Edit > 2D Edit Tools > Delete Nearly Collinear Edges**.
2. In the dialog box, enter a threshold distance.

Two edges (Edge 1 and Edge 2) are considered to be collinear if the distance D , defined as shown in [Figure 23](#), is less than the specified distance.

Figure 23 Distance definition for nearly collinear edges



The corresponding Scheme command is:

```
(sdegeo:delete-nearly-collinear-edges edge-list distance)
```

For example:

```
(sdegeo:create-rectangle (position 0.0 0.0 0) (position 1.0 1.0 0)
    "Silicon" "region_1")
(sdegeo:insert-vertex (position 0.25 0.0 0))
(sdegeo:insert-vertex (position 0.50 0.0 0))
(sdegeo:insert-vertex (position 0.75 0.0 0))
(sdegeo:move-vertex (find-vertex-id (position 0.5 0.0 0))
    (position 0.5 0.1 0))
(define EDGES (sde>window-select-2d -0.1 -0.1 1.1 0.2 "all" "edge"))
(sdegeo:delete-nearly-collinear-edges EDGES 0.2)
```

To remove all edges shorter than a given length by merging them with adjacent edges:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Edge**.
3. Select the edge and hold the Shift key to select additional edges, or drag a box around a set of edges.

Chapter 4: Generating Geometric Structures

Editing 2D Shapes

4. Choose **Edit > 2D Edit Tools > Delete Short Linear Edges.**

5. Enter the minimal-allowed edge length in the dialog box.

The corresponding Scheme command is:

```
(sdegeo:delete-short-edges edge-list minimal-edge-length)
```

For example:

```
(sdegeo:create-rectangle (position 0.0 0.0 0) (position 1.0 1.0 0)
    "Silicon" "region_1")
(sdegeo:insert-vertex (position 0.1 0.0 0))
(sdegeo:insert-vertex (position 0.0 0.1 0))
(sdegeo:insert-vertex (position 0.0 0.2 0))
(define EDGES (sde>window-select-2d -0.1 -0.1 1.1 0.3 "all" "edge"))
(sdegeo:delete-short-edges EDGES 0.2)
```

Note:

This command preserves vertices at which the angle between two edges is 90° or more (ridges). At rounded corners, this command might result in the unintended removal of the entire corner.

To remove edges based on an angular cut-off:

1. Choose **Edit > 2D Edit Tools > Delete Linear Edges.**

2. Enter the cut-off angle (angular tolerance) in the dialog box.

If the angle between two edges at a vertex is larger than the cut-off angle, the edges are removed.

Note:

The angle between collinear edges is 180°. To protect corners, use an angle greater than 90°.

The corresponding Scheme command is:

```
(sdegeo:delete-short-edges edge-list [edge-angular-cut-off])
```

The default value for edge-angular-cut-off is 90°.

For example:

```
(sdegeo:create-polygon (list
    (position 0.0 0.0 0) (position 1.0 0.0 0) (position 1.0 1.0 0)
    (position 0.8 1.02 0) (position 0.6 0.98 0) (position 0.4 1.02 0)
    (position 0.2 0.98 0) (position 0.0 1.0 0) (position 0.0 0.0 0))
    "Silicon" "region_1")
(define EDGES (sde>window-select-2d -0.1 0.9 1.1 1.1 "all" "edge"))
(sdegeo:delete-short-edges EDGES 150)
```

Chapter 4: Generating Geometric Structures

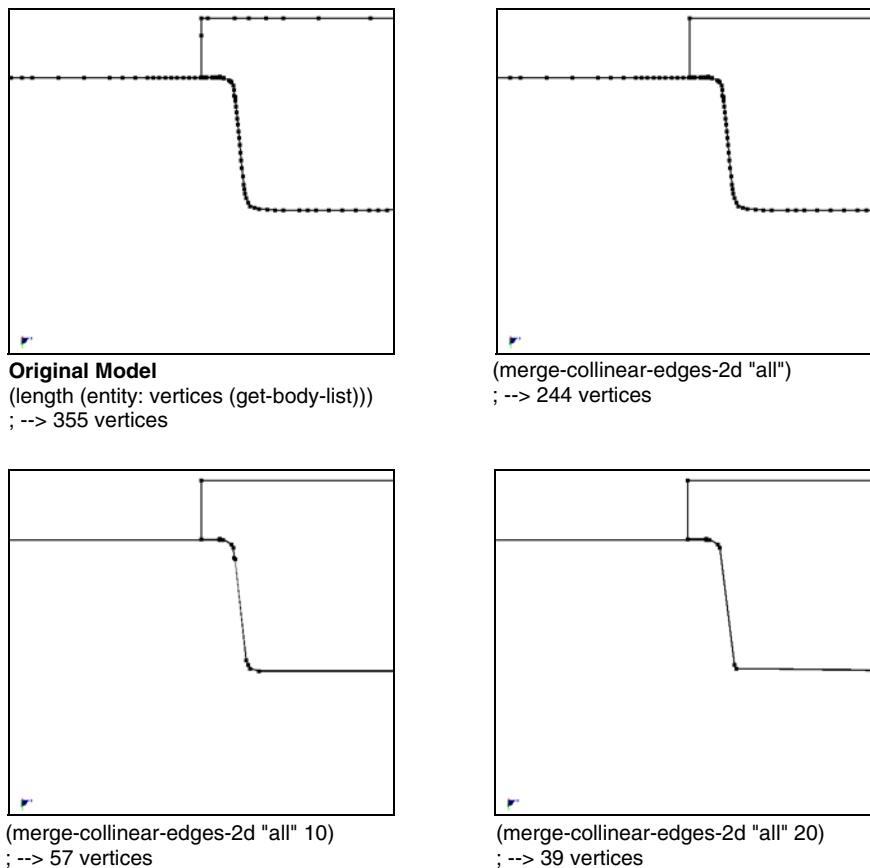
Editing 2D Shapes

The alternative command:

```
(merge-collinear-edges-2d body-list [normal-angular-cut-off])
```

works similarly to `sdegeo:delete-short-edges`, but it automatically selects all edges from all bodies selected, and it uses a different definition for the cut-off angle. Here, the edge is removed if the angle between the face normals is smaller than the given cut-off angle. For example, an `edge-angular-cut-off` of 120° corresponds to a `normal-angular-cut-off` of 30° .

Figure 24 Merging collinear edges: (upper left) original model and examples of different merges of collinear edges



An additional variance:

```
(sdegeo:prune-vertices body-list normal-angular-cut-off)
```

works with vertices instead of edges, but it uses the same algorithm for the selected edges to be removed. The command:

```
(sdegeo:dnce normal-angular-cut-off)
```

automatically calls `sdegeo:prune-vertices` for all bodies.

A further variant of the edge removal command allows you to specify an edge length as well as a cut-off angle. This command operates on a list of bodies, instead of an edge list, and is not available from the GUI.

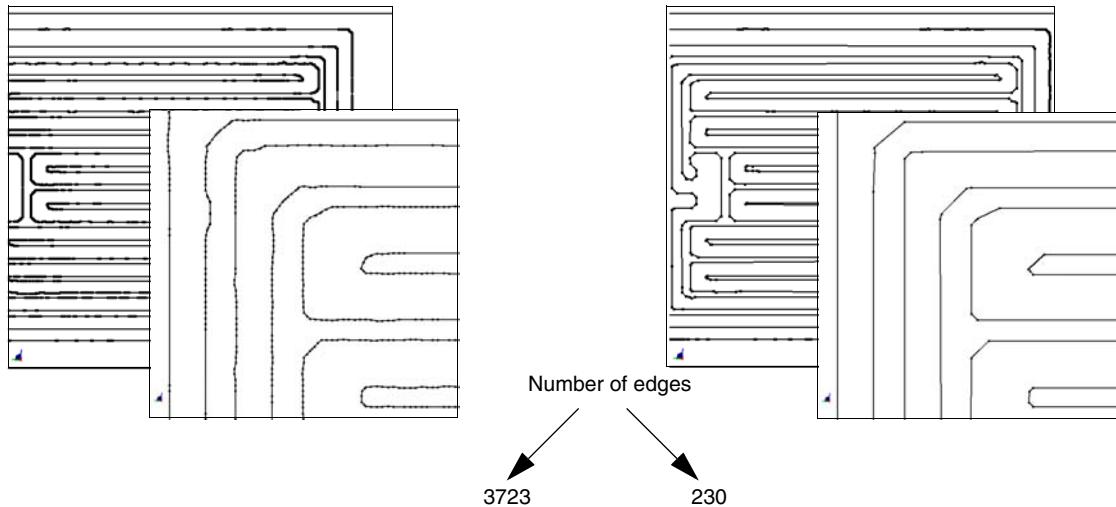
The Scheme command is:

```
(sdegeo:del-short-edges body-list edge-list minimal-edge-length  
normal-angular-cut-off)
```

For example:

```
(define BODY (sdegeo:create-rectangle (position 0.0 0.0 0)  
                                      (position 1.0 1.0 0) "Silicon" "R.Substrate"))  
(sdegeo:insert-vertex (position 0.1 0.0 0))  
(sdegeo:insert-vertex (position 0.0 0.1 0))  
(sdegeo:insert-vertex (position 0.0 0.2 0))  
(sdegeo:move-vertex (find-vertex-id (position 0 0 0))  
                     (position 0.03 0.03 0))  
(sdegeo:del-short-edges BODY 0.2 10)
```

Figure 25 Mask layout: (left) before deleting nearly collinear edges and (right) after by deleting nearly collinear edges



Breaking Nearly Axis-Aligned Edges

To break up nearly axis-aligned edges into a staircase of axis-aligned edges:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Edge**.

Chapter 4: Generating Geometric Structures

Editing 2D Shapes

3. Select the edge and hold the Shift key to select additional edges, or drag a box around a set of edges.
4. Choose **Edit > 2D Edit Tools > Break Nearly Axis-Aligned Edges**.
5. Enter a cut-off angle in the dialog box.

All edges that form an angle smaller than the cut-off angle with either the x-axis or y-axis are converted to a staircase-like set of edges.

The corresponding Scheme command is:

```
(sdegeo:create-polygon (list
```

```
  (position 0 0 0) (position 0.7 0.1 -0) (position 0.8 0.6 -0)
  (position 0.1 0.7 -0) (position -0.2 0.4 -0) (position 0 0 0))
  "Silicon" "region_1")
(define EDGES (sde:window-select-2d -0.5 -0.5 1.5 1.5 "all" "edge"))
(sdegeo:break-nearly-axis-aligned-edges EDGES 15)
```

Note:

Previously, slightly off-axis edges resulted in very small and unwanted mesh elements near interfaces. This is no longer the case for Sentaurus Mesh, which has improved support for nonaxis-aligned boundaries. This reduces the need for this kind of boundary simplification.

Figure 26 Breaking nearly axis-aligned edges into staircases: (left) original structure and (right) after the edge-breaking operation

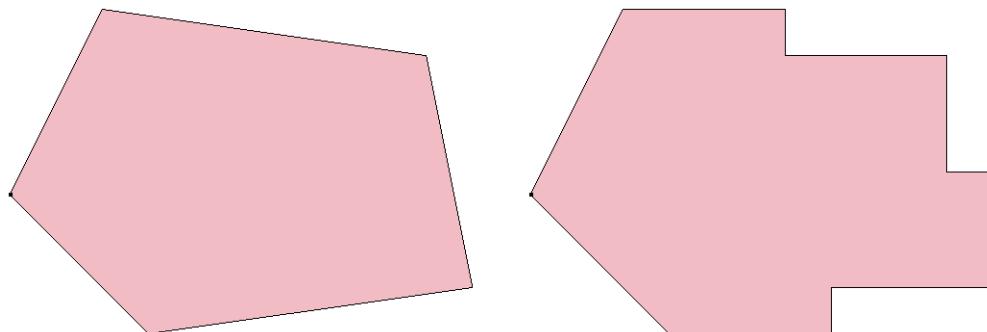
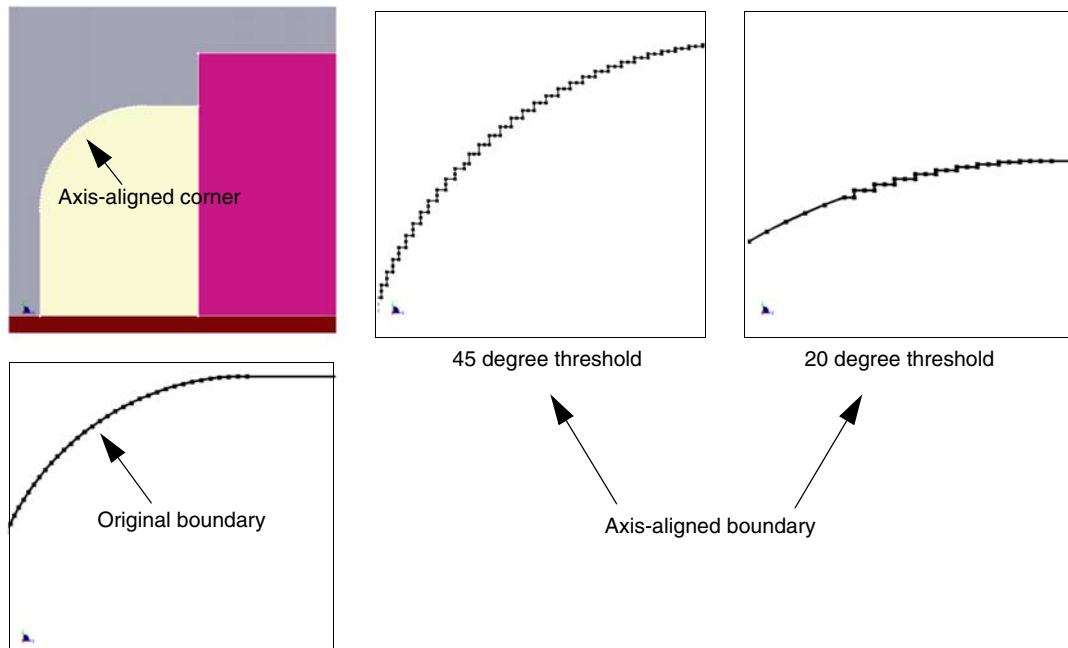


Figure 27 Breaking nearly axis-aligned edges to axis-aligned edges



Two-Dimensional Boundary Smoothing

The `sdegeo:vsmooth` Scheme extension performs 2D boundary smoothing. It uses a least squares fit of a small set of consecutive data points (vertices) to a polynomial and takes the calculated central point of the fitted polynomial curve as the new smoothed data point (vertex). The fitted polynomial can be either quadratic or cubic. The polynomial order (2 or 3) is the first argument of the function. The second argument is the vertex list that will be smoothed.

You can call the `sdegeo:vsmooth` Scheme extension multiple times to perform the boundary-smoothing operation successively.

You can specify all the vertices of the model as input by using the command `(entity:vertices (get-body-list))`. The global Scheme variable `bsmooth-ac` specifies an angular criterion (in degrees) for shape preservation (preserving ridges). The default value is 100. The following figures show different boundaries before and after vertex smoothing.

Figure 28 Vertex smoothing using sdegeo:vsmooth: (left) before and (right) after

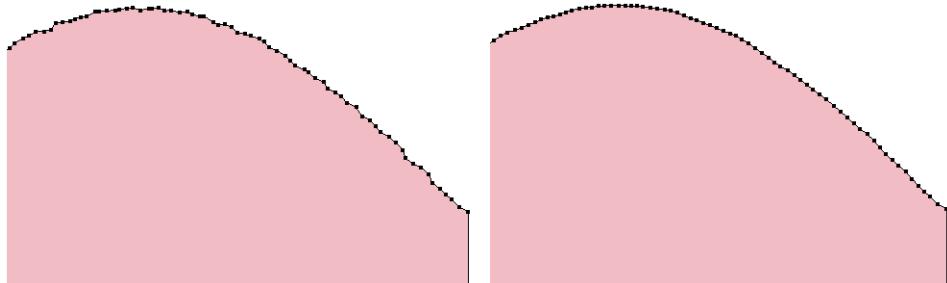


Figure 29 Multiple vertex-smoothing steps using sdegeo:vsmooth, showing ridge preservation

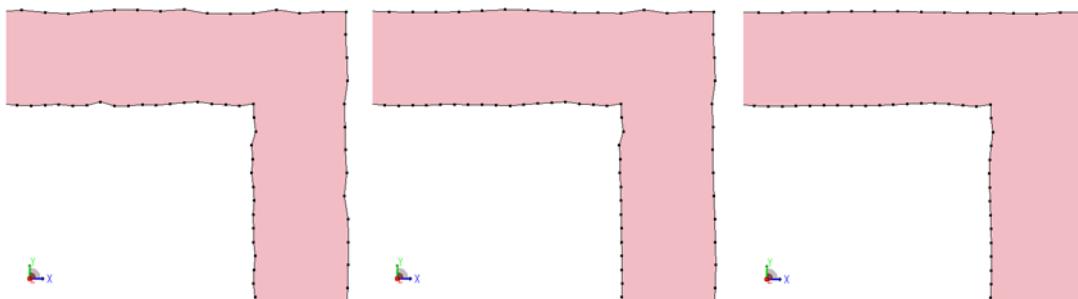
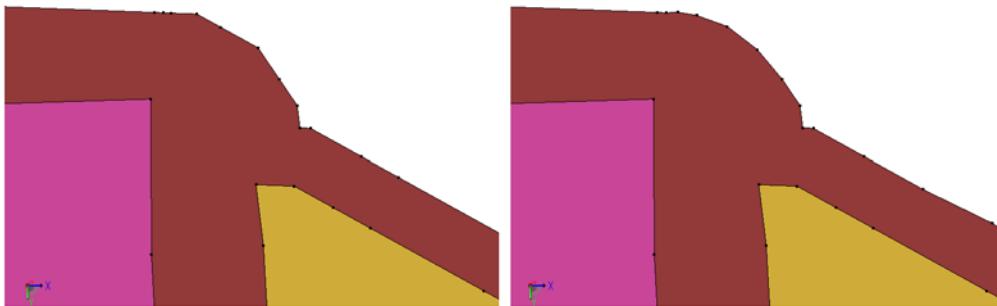


Figure 30 Vertex smoothing, using sdegeo:vsmooth: (left) before and (right) after



Edge Length Queries

Edge length query functions help to determine which edges can be considered as too short and, therefore, can be removed. The Scheme commands for edge length queries are:

```
(sdegeo:max-edge-length edge-list)
(sdegeo:min-edge-length edge-list)
(sdegeo:average-edge-length edge-list)
```

Chapter 4: Generating Geometric Structures

Editing 2D Shapes

For example:

```
(sdegeo:create-polygon (list
  (position 0.0 0.0 0) (position 1.0 0.0 0) (position 1.0 1.0 0)
  (position 0.8 1.02 0) (position 0.6 0.98 0) (position 0.4 1.02 0)
  (position 0.2 0.98 0) (position 0.0 1.0 0) (position 0.0 0.0 0))
  "Silicon" "region_1")
(define EDGES (sde:window-select-2d -0.1 -0.1 1.1 1.1 "all" "edge"))
(sdegeo:max-edge-length EDGES)
--> 1
(sdegeo:min-edge-length EDGES)
--> 0.200997512422418
(sdegeo:average-edge-length EDGES)
--> 0.501734670809496
```

[Table 20](#) lists the boundary-simplifying Scheme functions that are also available from the GUI.

Table 20 Scheme functions for 2D boundary simplification

Scheme function	Description
merge-collinear-edges-2d	Merges the collinear edges of the specified bodies.
sde:window-select-2d	Selects different entity types in a 2D window.
sdegeo:align-horizontal	Aligns points horizontally to a specified position.
sdegeo:align-horizontal-aut	Aligns points horizontally to the average position.
sdegeo:align-to-line	Snaps the specified points to a predefined straight line.
sdegeo:align-vertical	Aligns points vertically to a specified position.
sdegeo:align-vertical-aut	Aligns points vertically to the average position.
sdegeo:average-edge-length	Computes the average edge length (for an edge list).
sdegeo:break-nearly-axis-aligned-edges	Breaks nearly axis-aligned edges into horizontal or vertical components.
sdegeo:delete-collinear-edges	Deletes collinear edges.
sdegeo:delete-edges	Deletes specified edges.
sdegeo:delete-nearly-collinear-edges	Deletes nearly collinear edges.

Chapter 4: Generating Geometric Structures

Drawing Basic 3D Shapes

Table 20 Scheme functions for 2D boundary simplification (Continued)

Scheme function	Description
sdegeo:delete-short-edges	Deletes all edges from an edge list that are shorter than the specified value.
sdegeo:delete-vertices	Deletes specified list of vertices.
sdegeo:del-short-edges	Deletes short edges.
sdegeo:max-edge-length	Computes the longest edge length (for an edge list).
sdegeo:min-edge-length	Computes the shortest edge length (for an edge list).
sdegeo:prune-vertices, sdegeo:dnce	Deletes nearly collinear points.
sdegeo:vsmooth	Vertex smoothing in two dimensions.

Note:

Both sdegeo:del-short-edges and sdegeo:prune-vertices are very fast and robust. Use them to clean up 2D boundaries.

Drawing Basic 3D Shapes

This section discusses how to draw basic three-dimensional shapes.

Drawing Cuboids

To draw a cuboid:

1. Choose **Draw > 3D Create Tools > Create Cuboid**, or click the corresponding toolbar button (see [Table 11 on page 48](#)).
2. Drag to draw the diagonal of the xy rectangle in the view window.
3. Release the mouse button and move the pointer to the top z-coordinate of the cuboid and click.
4. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the first vertex and second vertex of the space diagonal.

Chapter 4: Generating Geometric Structures

Drawing Basic 3D Shapes

The corresponding Scheme command is:

```
(sdegeo:create-cuboid position position material-name region-name)
```

For example:

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 2 3) "Silicon"  
"R.Substrate")
```

Note:

In interactive mode, the base rectangle is always drawn in the $z = 0$ xy plane. To start on a different plane, change the work plane as discussed in [Work Planes on page 152](#). Use the isometric view to visualize all GUI operations (see [Table 6 on page 46](#)). Snapping is not supported in three dimensions.

Drawing Cylinders

To draw a cylinder:

1. Choose **Draw > 3D Create Tools > Create Cylinder**, or click the corresponding toolbar button (see [Table 11 on page 48](#)).
2. Drag to draw the radius of the base xy circle in the view window.
3. Release the mouse button and move the pointer to the top z-coordinate of the cylinder and click.
4. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the first vertex and second vertex of the center axis as well as the radius.

The corresponding Scheme command is [sdegeo:create-cylinder on page 612](#). For example:

```
(sdegeo:create-cylinder (position 0 0 0) (position 0 0 1) 2 "Silicon"  
"region_1")
```

Note:

In interactive mode, the base circle is always drawn in the $z = 0$ xy plane. To start on a different plane, change the work plane as discussed in [Work Planes on page 152](#). Use the isometric view to visualize all GUI operations (see [Table 6 on page 46](#)).

Drawing Spheres

To draw a sphere:

1. Choose **Draw > 3D Create Tools > Create Sphere**, or click the corresponding toolbar button (see [Table 11 on page 48](#)).
2. Drag to draw the radius of the base xy circle in the view window.
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the center of the sphere as well as the radius.

The corresponding Scheme command is [sdegeo:create-sphere on page 628](#). For example:

```
(sdegeo:create-sphere (position 0 0 2) 2 "Silicon" "region_1")
```

Note:

In interactive mode, the base circle is always drawn in the xy plane at Z = 0. To start on a different plane, change the work plane as discussed in [Work Planes on page 152](#).

Drawing Ellipsoids

To draw an ellipsoid:

1. Choose **Draw > 3D Create Tools > Create Ellipsoid**, or click the corresponding toolbar button (see [Table 11 on page 48](#)).
2. Drag to draw the main axis of the ellipsoid in the xy plane in the view window.
3. Release the mouse button and move the pointer to define the secondary axis of the ellipsoid and click.
4. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates of the center of the ellipsoid, the endpoint of the major axis, as well as the ratio between the major and the minor axes. (The two minor axes are degenerate.)

The corresponding Scheme command is [sdegeo:create-ellipsoid on page 615](#). For example:

```
(sdegeo:create-ellipsoid (position 1 2 3) (position 1 2 5) 2  
"Silicon" "region_1")
```

Note:

In interactive mode, the major axis is always drawn in the Z = 0 XY plane. To start on a different plane, change the work plane as discussed in [Work Planes on page 152](#). Use the isometric view to visualize all GUI operations (see [Table 6 on page 46](#)).

Drawing Other Basic 3D Shapes

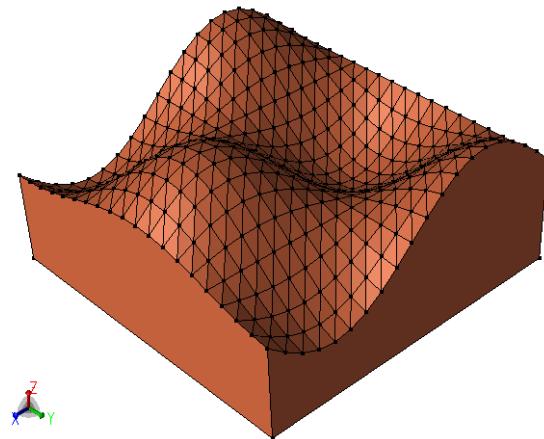
Other basic 3D shapes are available through the following Scheme extensions:

- `sdegeo:create-cone` (see [sdegeo:create-cone on page 609](#))
 - `sdegeo:create-prism` (see [sdegeo:create-prism on page 622](#))
 - `sdegeo:create-pyramid` (see [sdegeo:create-pyramid on page 623](#))
 - `sdegeo:create-torus` (see [sdegeo:create-torus on page 630](#))
-

Import Capability of User-Defined Data

The `build-csv-lens` Scheme extension can be used to create a solid body from user-defined data. The input data file is a comma-separated value (CSV) file, which defines the user data over a tensor grid. The solid body is created from the parsed CSV data by stitching the faces together into a water-tight solid body. [Figure 31](#) shows a solid body created using the extension `build-csv-lens`.

Figure 31 Solid body created using build-csv-lens, using the $\sin(x)\cos(y)$ analytic function defining the CSV data points on a 20×20 tensor grid*



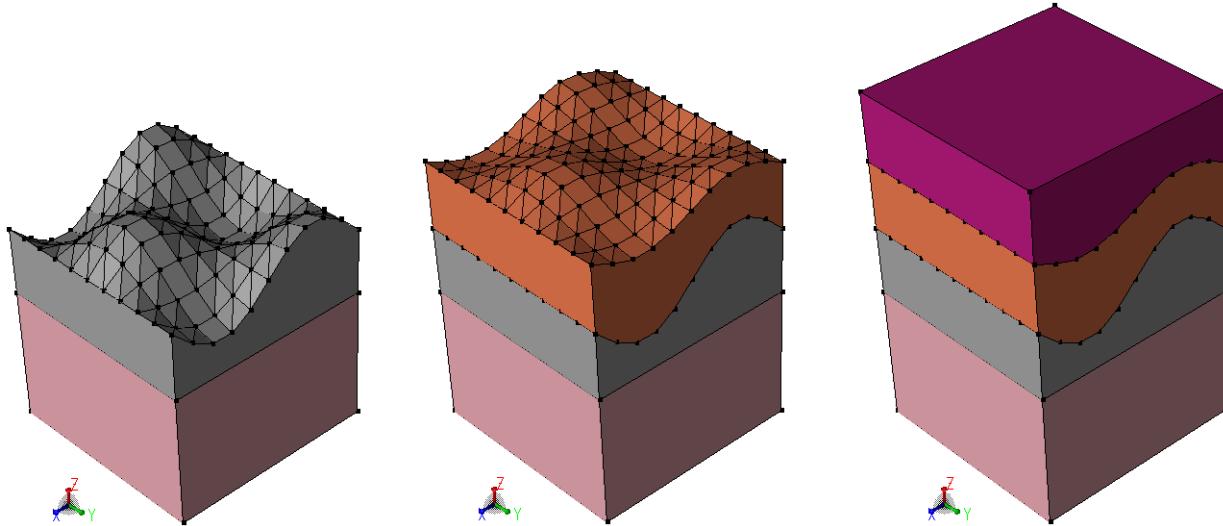
Creating a Layered Lens Structure

The `afm-smooth-layers` Scheme extension can create a layered lens structure with a texture map. The texture map is defined in a comma-separated value (CSV) file. [Figure 32](#) shows an example, created using `afm-smooth-layers` and the Scheme code that follows.

Chapter 4: Generating Geometric Structures

Drawing Basic 3D Shapes

Figure 32 Example of a layered structure created using the afm-smooth-layers command



Scheme example for the usage of afm-smooth-layers:

```
(define n 10)
(define a (* 2 PI))
(define oport (open-output-file "xx.csv"))
(display ", " oport)
(do ((i 0 (+ i 1))) ((> i n))
  (display (/ (* i a) n) oport)
  (display ", " oport)
)
(display "\n" oport)
(do ((i 0 (+ i 1))) ((> i n))
  (display (/ (* i a) n) oport)
  (display ", " oport)
  (do ((j 0 (+ j 1))) ((> j n))
    (define yp (* (sin (* j (/ a n))) (cos (* i (/ a n)))))
    (display yp oport)
    (if (not (equal? j n))
      (display ", " oport)))
  )
  (display "\n" oport)
)
(close-output-port oport)
(sde:clear)
(define layermaterial (list "TCO" "Copper" "PolySi"))
(define layerregion (list "tco" "copper" "polysi"))
(define layerthickness (list 2 2 2))
(sdegeo:create-cuboid (position 0 0 0) (position (* 2 PI) (* 2 PI) 4)
  "Silicon" "xx")
(afm-smooth-layers "xx.csv" 0 4 layerregion layermaterial
  layerthickness)
```

Creating a Solid Body From Faces

User-defined data (vertex points and a face list) also can be used to define 3D solid bodies. The faces must enclose a watertight volume (no gaps or overlaps are allowed between the faces). The `entity:stitch` Scheme extension can be used to build a solid body from the sheet bodies, which must be created first for each face.

The following script creates a tetrahedron with vertices (0,0,0), (1,0,0), (0,1,0) and (0,0,1):

```
(sde:clear)

;; Define the vertex points
(define p0 (position 0 0 0))
(define p1 (position 1 0 0))
(define p2 (position 0 1 0))
(define p3 (position 0 0 1))

;; Create sheet (2d polygonal) bodies for each face
;; (face orientation does not matter)
(define f0 (sheet:2d (sheet:planar-wire (wire-body:points
    (list p0 p1 p2 p0)))))

(define f1 (sheet:2d (sheet:planar-wire (wire-body:points
    (list p0 p1 p3 p0)))))

(define f2 (sheet:2d (sheet:planar-wire (wire-body:points
    (list p1 p2 p3 p1)))))

(define f3 (sheet:2d (sheet:planar-wire (wire-body:points
    (list p0 p2 p3 p0)))))

;; Stitch the faces together to create a solid body
(define mb (car (entity:stitch (list f0 f1 f2 f3))))
(sde:add-material mb "Copper" "mytetrahedron")

;; If the sheet bodies define a watertight enclosure,
;; the solid body will pass the entity:checker and
;; can be used for additional operations (Booleans and so on)
```

Editing 3D Shapes

This section discusses how to edit 3D shapes.

Chamfering Edges

To chamfer (flatten) the edges of a 3D structure:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Edge** (or **Select Vertex**).

Chapter 4: Generating Geometric Structures

Editing 3D Shapes

3. Select the edge (or vertex) and hold the Shift key to select additional edges (or vertices), or drag a box around a set of edges (or vertices).
4. Choose **Edit > 3D Edit Tools > Chamfer**.
5. Enter a chamfer distance in the dialog box.

If the parameter `chamfer-dist` was set before performing this operation (choosing **Edit > Parameters**), this dialog box is suppressed, and the given value of `chamfer-dist` is used instead.

The optional `adaptive-chamfering` Boolean argument can be used to perform adaptive chamfering. If this option is used and the argument is set to `#t`, then if the chamfering operation fails using the original `chamfer-dist` value, the operation is repeated with a sequence of chamfering operations, using an adaptive approach, to set the chamfer distance to smaller values until the operation succeeds.

The corresponding Scheme command is:

```
(sdegeo:chamfer edge | vertex-list chamfer-dist [adaptive-chamfering])
```

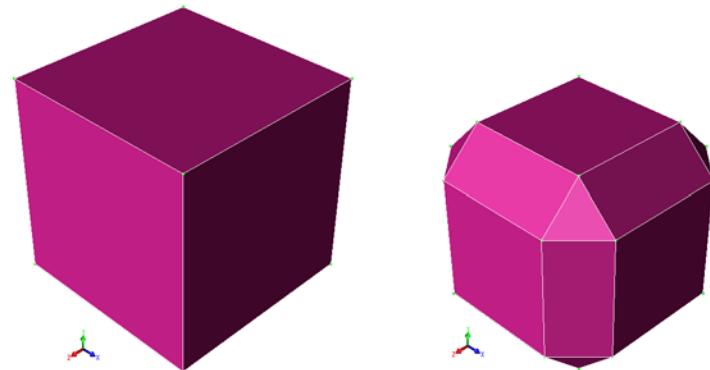
For example:

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon"
  "region_1")
(sdegeo:chamfer (list
  (car (find-vertex-id (position 0 0 1)))
  (car (find-vertex-id (position 0 1 1)))
  (car (find-vertex-id (position 1 0 1)))
  (car (find-vertex-id (position 1 1 1)))) 0.2)
```

Note:

If `chamfer-dist` is too large such that two adjacent chamfers would overlap, then the operation will fail.

Figure 33 Three-dimensional chamfering operation: (left) original cuboid ($1 \times 1 \times 1$) and (right) $\text{chamfer-dist}=0.2$



Rounding Edges

To round the edges of a 3D structure:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Edge** (or **Select Vertex**).
3. Select the edge (or vertex) and hold the Shift key to select additional edges (or vertices), or drag a box around a set of edges (or vertices).
4. Choose **Edit > 3D Edit Tools > Fillet**.
5. Enter a fillet radius in the dialog box.

If the parameter `fillet-radius` was set before performing this operation (choosing **Edit > Parameters**), this dialog box is suppressed, and the given value of `fillet-radius` is used instead.

The optional `adaptive-filletting` Boolean argument can be used to perform adaptive filleting. If this option is used and the argument is set to `#t`, then if the filleting operation fails using the original `fillet-radius` value, the operation is repeated with a sequence of filleting operations, using an adaptive approach, to set the fillet radius to smaller values until the operation succeeds.

The corresponding Scheme command is:

```
(sdegeo:fillet edge | vertex-list fillet-radius [adaptive-filletting])
```

For example:

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon"  
"region_1")  
(sdegeo:fillet (list  
  (car (find-vertex-id (position 0 0 1)))  
  (car (find-vertex-id (position 0 1 1)))  
  (car (find-vertex-id (position 1 0 1)))  
  (car (find-vertex-id (position 1 1 1)))) 0.2)
```

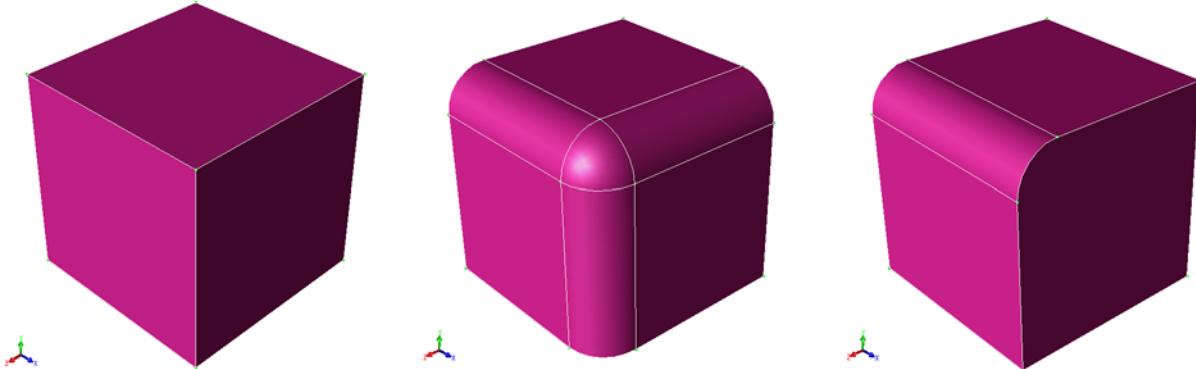
and:

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon"  
"region_1")  
(sdegeo:fillet (list (car (find-edge-id (position 0 0 0.5))))) 0.2)
```

Note:

If `fillet-radius` is too large such that two adjacent fillets overlap, then the operation will fail.

Figure 34 Three-dimensional edge filleting: (left) original cuboid ($1 \times 1 \times 1$), (middle) rounded corner, and (right) rounded edge with fillet-radius=0.2

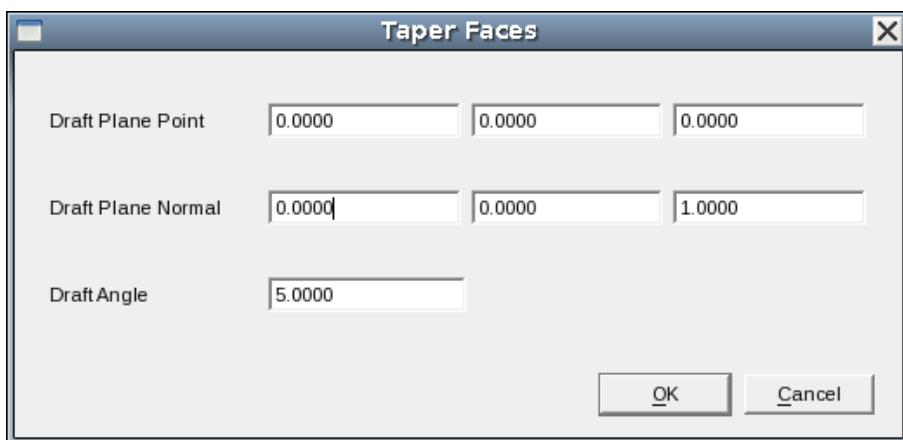


Tapering

To taper one or more faces of a 3D structure:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Select the face and hold the Shift key to select additional faces, or drag a box around a set of faces.
4. Choose **Edit > 3D Edit Tools > Taper**.

The Taper Faces dialog box opens.



5. Enter the draft plane point, the draft plane normal, and the draft angle.
6. Click **OK**.

Chapter 4: Generating Geometric Structures

Editing 3D Shapes

The Taper Faces dialog box is initialized with the following default values: If only one face is selected, the draft plane point is initialized, using the barycentric coordinates of the selected face.

If more than one face is selected:

- The draft plane point is initialized as 0,0,0.
- The draft plane normal is initialized as 0,0,1.
- The draft angle is initialized to 5°.

The corresponding Script command is:

```
(sdegeo:taper-faces face-list point normal angle)
```

The tapering algorithms are discussed using simple examples.

[Figure 35](#) (a) shows a cuboid where the side face to be tapered is blue. For reference, the normal vector of this face is also shown.

The tapering is performed with respect to an auxiliary plane, the so-called draft plane. This plane is defined by the draft plane center point and the draft plane normal vector.

[Figure 35](#) (b) shows the draft plane in green together with the draft plane normal vector originating from the draft plane center point. In addition, the rotation axis, which is defined as the intersection between the face plane (blue) and the draft plane (green), is shown.

With respect to these planes and axes, now two distinct rotations are performed.

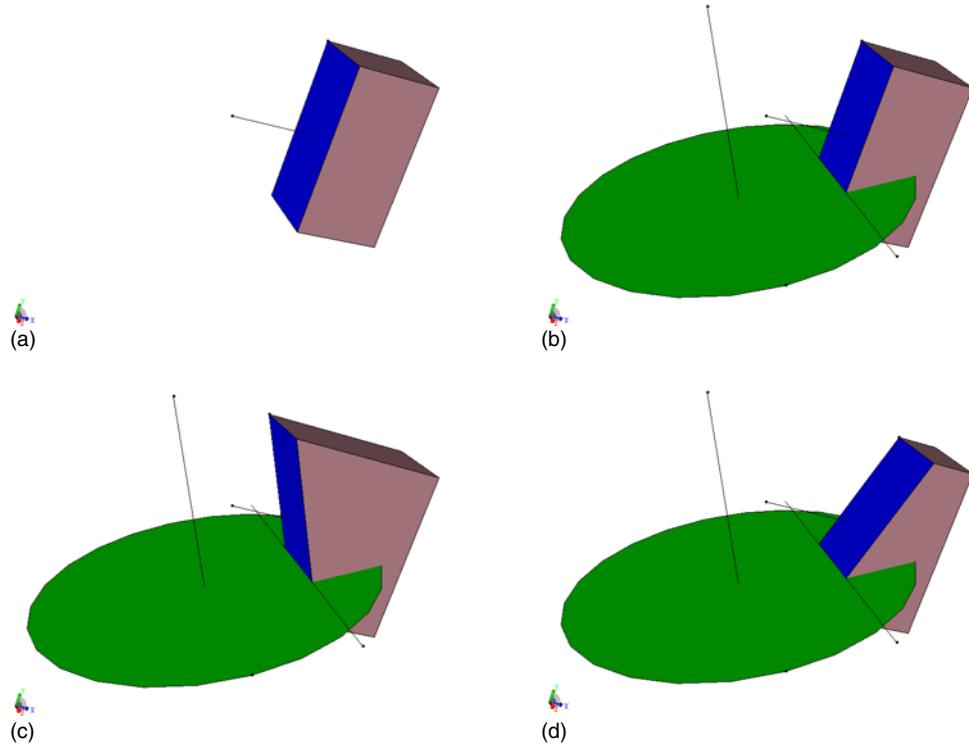
[Figure 35](#) (c) shows the new face location after the first rotation (equivalent to tapering with an angle of 0°). The face is rotated around the axis such that the new face is orthogonal to the draft plane.

[Figure 35](#) (d) shows the face after the second rotation, with an angle of 45°. The rotation direction is given by the *right-hand rule*: To determine the direction of the axis, align the thumb of your right hand with the face plane normal and the index finger with the draft plane normal. Then extend your middle finger orthogonally to both. This finger points now in the direction of the axis vector. Now, point the thumb of your right hand into the axis direction and form a fist with all other fingers. These fingers now curl like a rotation with a positive angle.

Chapter 4: Generating Geometric Structures

Editing 3D Shapes

Figure 35 Tapering example



The following Scheme commands produced the graphics in Figure 35:

```
(sdegeo:create-cuboid (position 0.0 0.0 -0.5) (position 1.0 2.0 0.5)
  "Silicon" "region_1")
(define FACE (find-face-id (position 0.0 0.5 0.0)))
(entity:set-color FACE BLUE)
(define FACE_CENTER (cdr (list-ref (face:prop (car FACE)) 2)))
(define FACE_NORMAL (face:plane-normal (car FACE)))
(wire-body:points
  (list FACE_CENTER (position:offset FACE_CENTER FACE_NORMAL)))
; Figure (a)
(define DRAFT_CENTER (position -1.0 0.0 0.0))
(define DRAFT_NORMAL (gvector -1.0 2.0 0.0))
(wire-body:points
  (list DRAFT_CENTER (position:offset DRAFT_CENTER DRAFT_NORMAL)))
(face:planar-disk DRAFT_CENTER DRAFT_NORMAL 2)
(wire-body:points (list (position 0.0 0.5 -2.0)
  (position 0.0 0.5 2.0)))
; Figure (b)
(sdegeo:taper-faces FACE DRAFT_CENTER DRAFT_NORMAL 0)
; Figure (c)
(sdegeo:taper-faces FACE DRAFT_CENTER DRAFT_NORMAL 45)
; Figure (d)
```

Chapter 4: Generating Geometric Structures

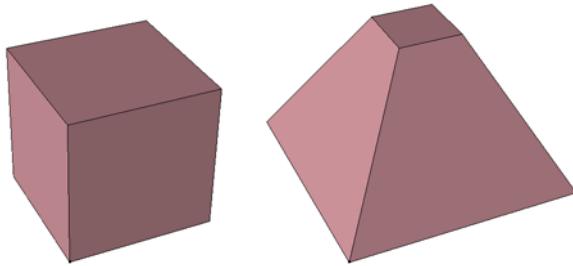
Editing 3D Shapes

The tapering process, based on two independent rotations, allows you to perform two distinct types of taper operation. The first type of tapering results in truncated pyramid-type shapes, where, for example, all faces are bent inwards. For this kind of tapering, the draft normal vector is orthogonal to the face normal vectors and the rotation angle is nonzero. For example:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 1.0 1.0)
  "Silicon" "region_1")
(sdegeo:taper-faces (list
  (car (find-face-id (position 0.5 0.0 0.5)))
  (car (find-face-id (position 1.0 0.5 0.5)))
  (car (find-face-id (position 0.5 1.0 0.5)))
  (car (find-face-id (position 0.0 0.5 0.5))))
  (position 0.5 0.5 0.5) (gvector 0 0 1) 30)
```

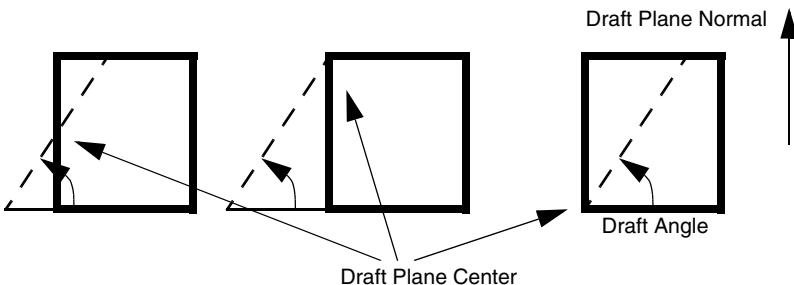
[Figure 36](#) shows a unit cube after applying truncated pyramid-type tapering.

Figure 36 Truncated pyramid-type tapering: (left) original cuboid and (right) after truncation



[Figure 37](#) illustrates the effect of the use of different draft plane centers. For the operations, the same draft plane normal and draft angle were used.

Figure 37 Draft plane center



The second type of tapering is similar to applying a sheer force to the body. For this kind of tapering, the draft normal vector is not orthogonal to the face normal vectors, but the rotation angle is zero.

Chapter 4: Generating Geometric Structures

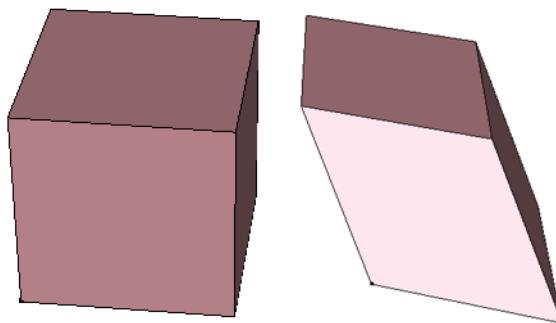
Editing 3D Shapes

For example:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 1.0 1.0)
  "Silicon" "region_1")
(sdegeo:taper-faces (list
  (car (find-face-id (position 0.5 0.0 0.5)))
  (car (find-face-id (position 1.0 0.5 0.5)))
  (car (find-face-id (position 0.5 1.0 0.5)))
  (car (find-face-id (position 0.0 0.5 0.5))))
  (position 0.5 0.5 0.5) (gvector 0.5 0.5 1) 0)
```

[Figure 38](#) shows a unit cube after applying sheer-type tapering.

Figure 38 Sheer-type tapering: (left) original cuboid and (right) after tapering



Even more general tapering can be obtained by mixing both fundamental types.

Note:

The taper operation can result in a geometric shape that is no longer closed. In this case, so-called vent faces are added automatically such that the resulting shape is closed.

Vent faces can only be added if there is a face on the original model that shares a vertex with the ‘mergeable’ edge, that does not have the mergeable edge in its boundary, and that will be adjacent to the vent faces after the taper. Vent faces can also be added at tangent edges when one of the two faces that share the edge is being tapered. However, a vent face will only be added if there is no intersection between the surface of the face that is being tapered and the surface of the face that is not being tapered.

Figure 39 Topology change during face tapering

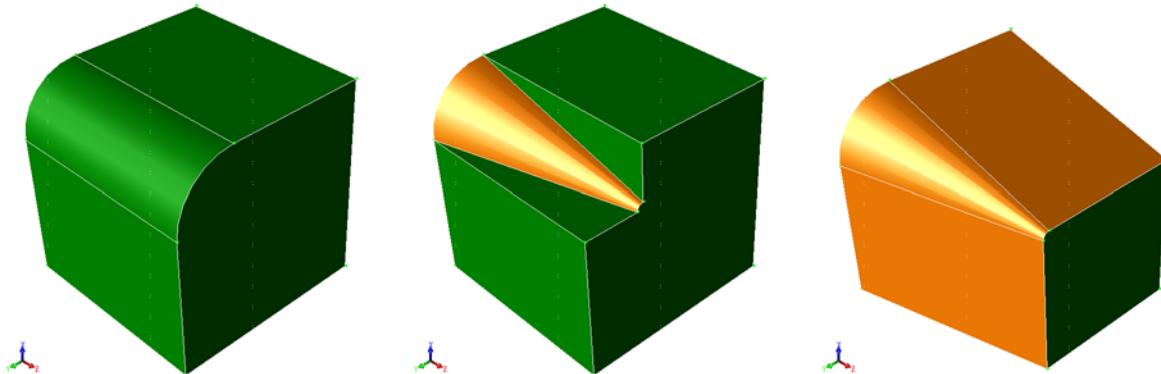


Figure 40 shows a structure generated by the following script:

```
(sdegeo:set-default-boolean "ABA")
(define r1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
    "Silicon" "region_1"))
(define r2 (sdegeo:create-cuboid (position 2 2 8) (position 8 8 12)
    "PolySilicon" "region_2"))
(sdegeo:delete-region (find-body-id (position 5 5 11)))
(define facelist (list
    (car (find-face-id (position 2 5 9)))
    (car (find-face-id (position 8 5 9)))
    (car (find-face-id (position 5 2 9)))
    (car (find-face-id (position 5 8 9)))))

(sdegeo:taper-faces facelist (position 5 5 9) (gvector 0 0 1) 30)
```

Figure 40 Face tapering

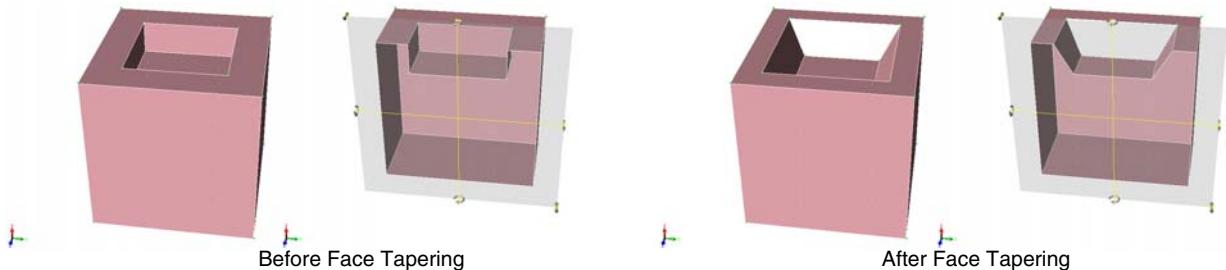


Figure 41 shows the structure generated by the following script:

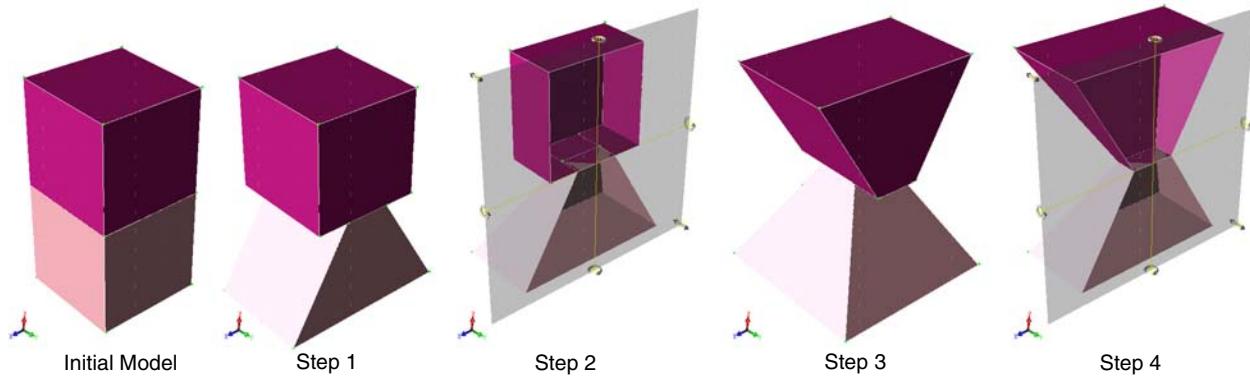
```
; Initial model
(sdegeo:set-default-boolean "ABA")
(define r1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
    "Silicon" "region_1"))
(define r2 (sdegeo:create-cuboid (position 0 0 10) (position 10 10 20)
    "PolySilicon" "region_2"))
; Step 1
(define facelist (list (car (find-face-id (position 0 5 5))))
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

```
(car (find-face-id (position 10 5 5)))))  
(sdegeo:taper-faces facelist (position 5 5 5) (gvector 0 0 1) 30)  
; Step 2  
(define facelist (list (car (find-face-id (position 5 0 5)))  
    (car (find-face-id (position 5 10 5)))))  
(sdegeo:taper-faces facelist (position 5 5 5) (gvector 0 0 1) 30)  
; Step 3  
(define facelist (list (car (find-face-id (position 0 5 15)))  
    (car (find-face-id (position 10 5 15)))))  
(sdegeo:taper-faces facelist (position 5 5 15) (gvector 0 0 1) -30)  
; Step 4  
(define facelist (list (car (find-face-id (position 5 0 15)))  
    (car (find-face-id (position 5 10 15)))))  
(sdegeo:taper-faces facelist (position 5 5 15) (gvector 0 0 1) -30)
```

Figure 41 Face tapering



Creating 3D Objects From 1D and 2D Objects

This section describes how to create 3D objects from 1D and 2D objects.

Wires

A wire is a 1D object that is used to define a sweep path or a wire body for skinning.

To draw a piecewise linear wire (polyline):

1. Choose **Draw > 2D Create Tools > Polyline**, or click the corresponding toolbar button (see [Table 8 on page 47](#)).
2. Click at each vertex of the polyline. To finish drawing the polyline, click the middle mouse button. (Press both buttons when using a two-button mouse.)
3. In Exact Coordinates mode, a dialog box is displayed where you enter the coordinates for each vertex.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

The corresponding Scheme command is:

```
(sdegeo:create-polyline-wire vertex-list)
```

For example:

```
(sdegeo:create-polyline-wire (list  
    (position 0.0 0.0 0.0) (position 1.0 0.0 0.0)  
    (position 1.0 1.0 0.0) (position 1.0 1.0 1.0)  
    (position 0.0 1.0 1.0) (position 0.0 1.0 0.0)))
```

To draw a spline interpolated wire, choose **Draw > 2D Create Tools > Spline**, or click the corresponding toolbar button (see [Table 9 on page 47](#)).

The corresponding Scheme command is:

```
(sdegeo:create-spline-wire vertex-list)
```

For example:

```
(sdegeo:create-spline-wire (list  
    (position 0.0 0.0 0.0) (position 1.0 0.0 0.0)  
    (position 1.0 1.0 0.0) (position 1.0 1.0 1.0)  
    (position 0.0 1.0 1.0) (position 0.0 1.0 0.0)))
```

Extruding

To extrude a 2D object or a face of a 3D object:

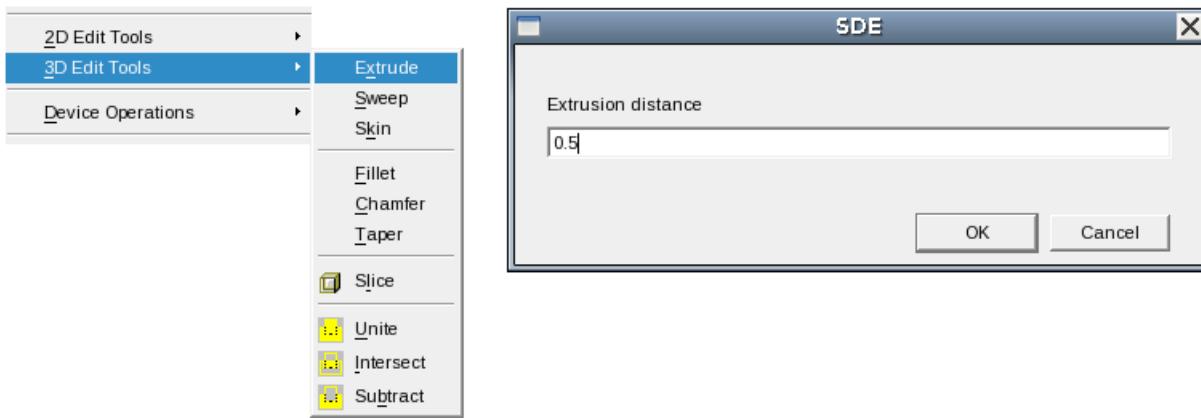
1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Select the face and hold the Shift key to select additional faces, or drag a box around a set of faces.
4. Choose **Edit > 3D Edit Tools > Extrude**.
5. Enter an extrusion distance in the dialog box.

If the parameter `hext` was set before performing this operation (choosing **Edit > Parameters**), this dialog box is suppressed, and the given value of `hext` is used instead.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 42 (Left) Menu for extrusion and (right) Extrusion dialog box



The corresponding Scheme command is:

```
(sdegeo:extrude entity-list hext)
```

For example:

```
(sdegeo:create-rectangle (position 0.0 0.0 0) (position 1.2 1.0 0)
    "Silicon" "region_1")
(sdegeo:create-rectangle (position 0.4 1.0 0) (position 0.8 1.1 0)
    "Oxide" "region_2")
(sdegeo:create-rectangle (position 0.4 1.1 0) (position 0.8 1.5 0)
    "PolySilicon" "region_3")
(sdegeo:extrude (list (car (find-face-id (position 0.6 0.5 0)))) 0.8)
(sdegeo:extrude (list
    (car (find-face-id (position 0.6 1.05 0))))
    (car (find-face-id (position 0.6 1.2 0)))) 0.4)
```

Different extrusion distances can be used for different regions.

In general, the extrusion direction is given by the face normal. For 2D objects, which are created in the xy plane by default, the face normal is defined to point along the z-axis.

Note:

Negative extrusion values, applied to 3D faces, generate an invalid geometry.

[Figure 43](#) illustrates the extrusion operation for 2D devices.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 43 Creating an extruded 3D model from a 2D device; the extrusion length can differ

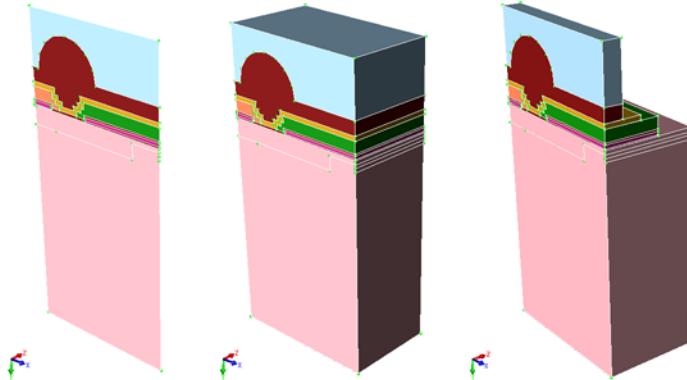
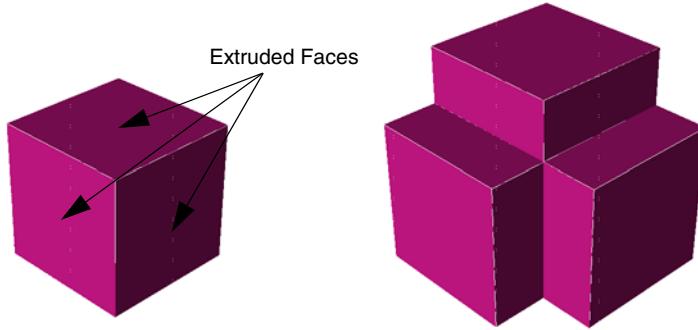


Figure 44 Example of face extrusion in a 3D object



Sweeping a 2D Object

A 3D object can be created by sweeping a 2D object along a path. A 3D object can also be modified by sweeping a set of faces of the 3D object. The sweep path can be defined in several different ways. In the simplest case, only the sweep distance is given and the sweep direction will be normal to the face. The sweep distance and sweep direction can also be defined using a sweep vector. In addition, the sweep path can be defined by a rotation around an axis or explicitly by a wire body.

Sweeping and extruding can also be applied to Ref/Eval windows. If a swept Ref/Eval window is used to place a submesh doping profile, information about the sweep creation is stored in the .cmd file, so that the mesh generator can evaluate the swept 2D doping profile in three dimensions.

In the case of other doping profile types, the resulting swept Ref/Eval window (without sweep information) is stored in the .cmd file, requiring that these elements must be valid reference elements (doping baseline elements) or evaluation windows. The sweep operation transforms the 2D edge contacts to 3D face contacts.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

The sweep options allow additional control over the sweeping behavior and enable drafting, twisting, and other advanced behaviors.

Sweeping is a complex operation involving several different types. In the following sections, the supported sweep types are introduced.

Sweep Distance

The sweep distance operation is similar to extrusion. Faces of 2D bodies or 3D bodies can be swept by a distance. The sweep distance option can be used with various sweep options.

To sweep a 2D object or a face of a 3D object using distance sweeping:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Select the face and hold the Shift key to select additional faces, or drag a box around a set of faces.
4. Choose **Edit > 3D Edit Tools > Sweep**.

The Sweep Operations dialog box opens.

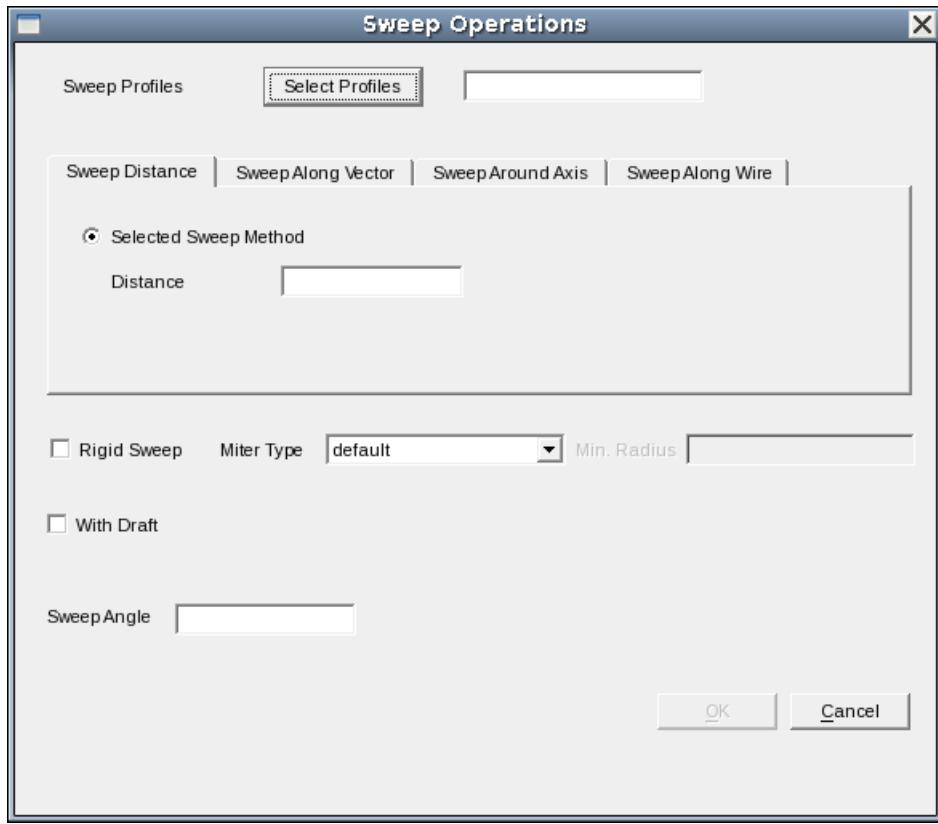
5. Click **Select Profiles**. The entity IDs of the selected profiles are displayed in the **Sweep Profiles** field.
6. Click the **Sweep Distance** tab.
7. Select the **Selected Sweep Method** option.
8. Enter a sweep distance.
9. Select sweep options as needed (see [Sweep Options on page 131](#)).
10. Click **OK**.

[Figure 45](#) shows the Sweep Operations dialog box with the **Selected Sweep Method** option selected. [Figure 46](#) shows the effect of a sweep distance operation, together with the use of the draft angle option. When a draft angle is used during a sweep operation, additional faces must be added to the model if the `gap_type` sweep option is 'extended' or 'rounded.'

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 45 Sweep Operations dialog box showing Sweep Distance tab



The corresponding Scheme command is:

```
(sdegeo:sweep face|face-list distance [sweep-options])
```

The examples in [Figure 46](#) are generated using the following Scheme commands:

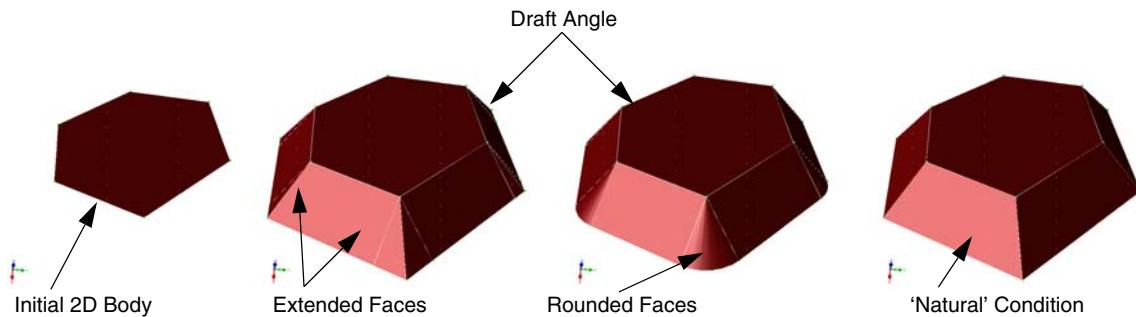
```
(define FACE (sdegeo:create-reg-polygon (position 0 0 0) 1 6 0
    "Oxide" "region_1"))
(sdegeo:translate-selected FACE (transform:translation
    (gvector 3 2 0)) #t 3)
(sdegeo:sweep (sde:find-reg-region "region_1_t1_r1") 0.6
    (sweep:options "solid" #t "rigid" #f "miter_type" "default"
        "draft_angle" 40.0 "gap_type" "extended"))
(sdegeo:sweep (sde:find-reg-region "region_1_t1_r2") 0.6
    (sweep:options "solid" #t "rigid" #f "miter_type" "default"
        "draft_angle" 40.0 "gap_type" "rounded"))

(sdegeo:sweep (sde:find-reg-region "region_1_t1_r3") 0.6
    (sweep:options "solid" #t "rigid" #f "miter_type" "default"
        "draft_angle" 40.0 "gap_type" "natural"))
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 46 Sweep distance with a draft angle (40°) using different gap-type options



Sweep Along a Vector

The sweep along a vector option is similar to the sweep distance option, except that a vector is specified to define the sweep path.

To sweep a 2D object or a face of a 3D object using vector sweeping:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Select the face and hold the Shift key to select additional faces, or drag a box around a set of faces.
4. Choose **Edit > 3D Edit Tools > Sweep**.

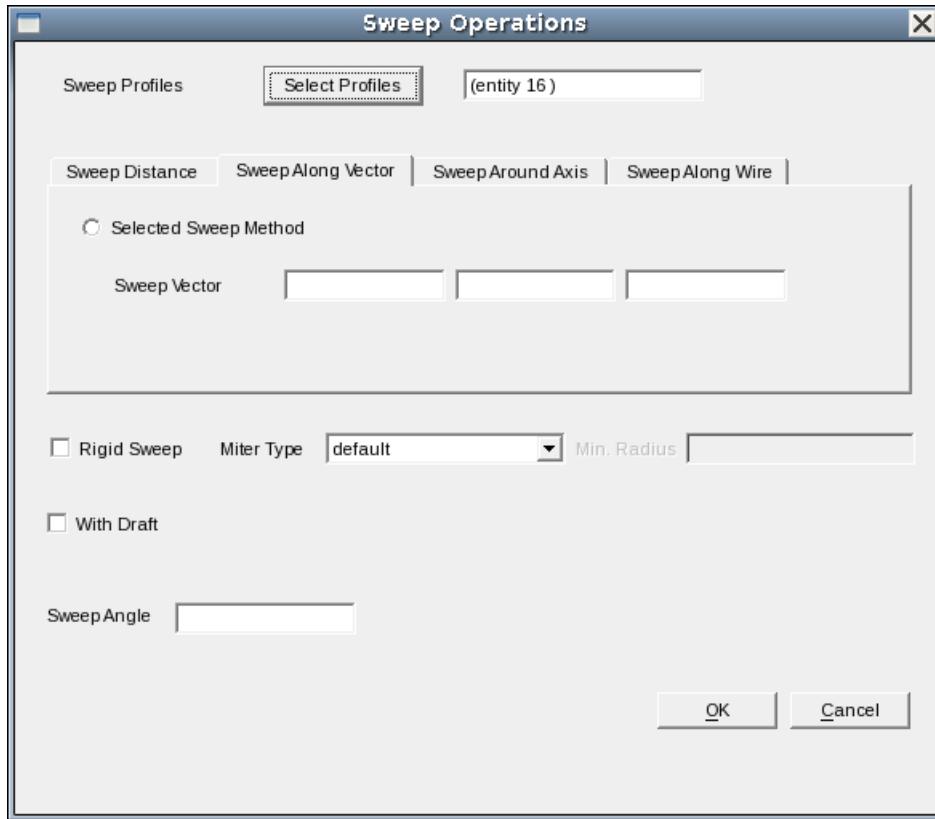
The Sweep Operations dialog box opens.

5. Click **Select Profiles**. The entity IDs of the selected profiles are displayed in the **Sweep Profiles** field.
6. Click the **Sweep Along Vector** tab.
7. Select **Selected Sweep Method**.
8. Enter the x-, y-, and z-coordinates of the sweep vector.
9. Select sweep options as needed (see [Sweep Options on page 131](#)).
10. Click **OK**.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 47 Sweep Operations dialog box showing Sweep Along Vector tab



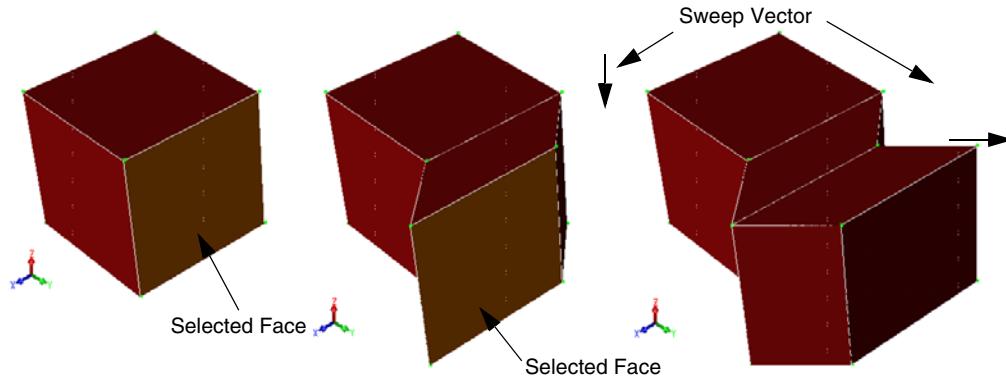
The corresponding Scheme command is:

```
(sdegeo:sweep face|face-list gvector [sweep-options])
```

The examples in [Figure 48](#) are generated using the following Scheme commands:

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Oxide"
  "region_1")
(sdegeo:sweep (find-face-id (position 0.5 1 0.5)) (gvector 0.5 0.5 0)
  (sweep:options "solid" #t "rigid" #f "miter_type" "default"))
(sdegeo:sweep (find-face-id (position 1.0 1.5 0.5))
  (gvector -0.5 0.5 0)
  (sweep:options "solid" #t "rigid" #f "miter_type" "default"))
```

Figure 48 Sweep along a vector



Sweep Around an Axis (Regular and Corner Sweeps)

The sweep around an axis option can be used to sweep a face (2D or 3D) around an axis. The sweep can be regular or such that only the inside edge is swept, while the outside edge forms an axis-aligned corner.

Regular Sweep

To sweep a 2D object or a face of a 3D object using regular sweeping around an axis:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Select the face and hold the Shift key to select additional faces, or drag a box around a set of faces.
4. Choose **Edit > 3D Edit Tools > Sweep**.
The Sweep Operations dialog box opens.
5. Click **Select Profiles**.
The entity IDs of the selected profiles are displayed in the **Sweep Profiles** field.
6. Click the **Sweep Around Axis** tab
7. Select **Selected Sweep Method**.
8. Define the axis of rotation by entering a point on the axis and a direction vector, or click **Pick Position** and click a point in the view window. In this case, the rotation axis will point into the current view plane.
9. Enter the sweep angle. (Do not select the **Sweep Corner** option.)

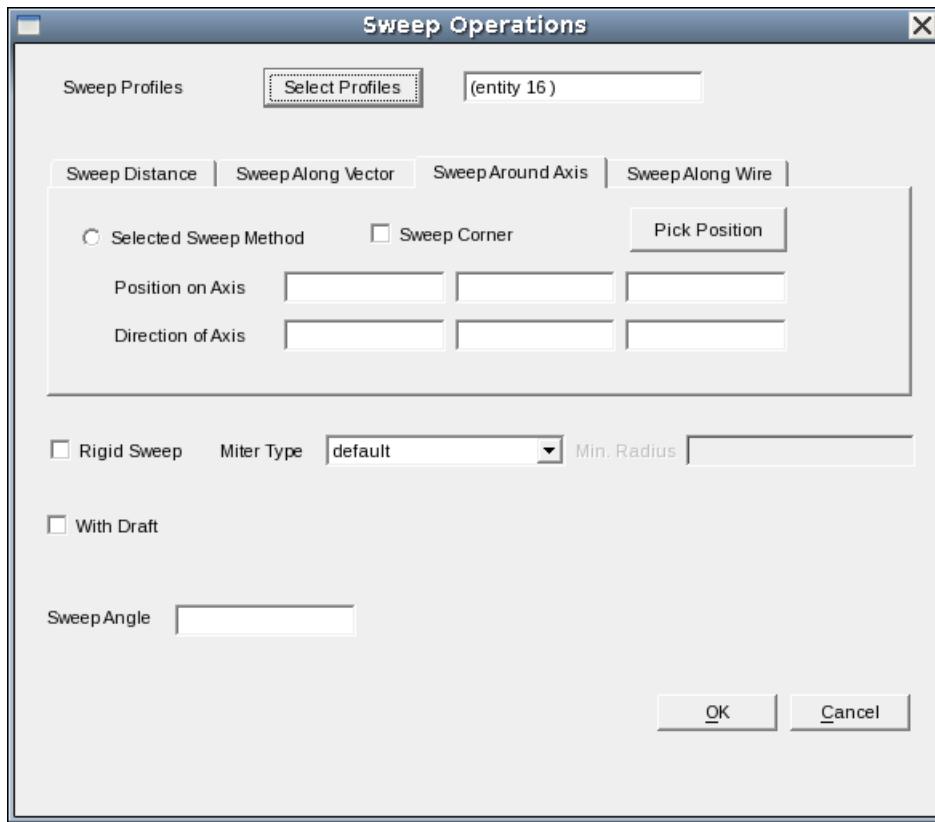
Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

10. Select sweep options as needed (see [Sweep Options on page 131](#)).

11. Click **OK**.

Figure 49 Sweep Operations dialog box showing Sweep Around Axis tab



The corresponding Scheme command is:

```
(sdegeo:sweep face|face-list position gvector  
  (sweep:options "sweep_angle" angle [other-sweep-options]))
```

The examples in [Figure 50](#) are generated using the following Scheme commands:

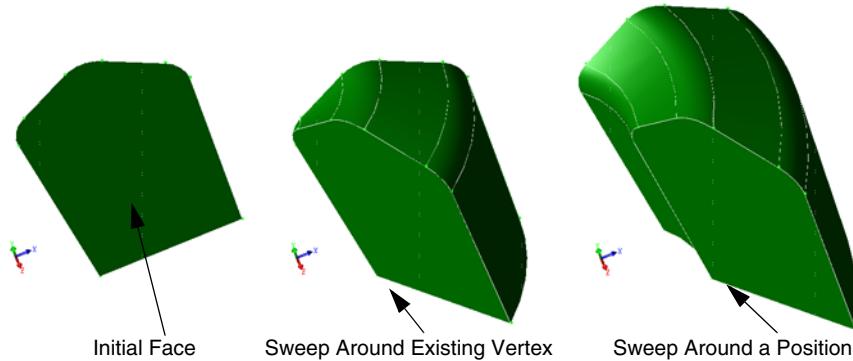
```
(sdegeo:create-rectangle (position 0 0 0.0) (position 1 2 0.0)  
  "Silver" "region_1")  
(define NEWVERT (sdegeo:insert-vertex (position 0.5 2 0)))  
(sdegeo:move-vertex NEWVERT (position 0.5 2.3 0))  
(sdegeo:fillet-2d (sde>window-select-2d -0.1 1.9 1.1 2.4 "all"  
  "vertex") 0.2)  
(sdegeo:translate-selected (find-body-id (position 0.5 0.5 0))  
  (transform:translation (gvector 2 0 0)) #t 2)  
(sdegeo:sweep (find-face-id (position 2.5 0.5 0)) (position 2 0 0)  
  (gvector 0 1 0)  
  (sweep:options "solid" #t "sweep_angle" 60 "rigid" #f
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

```
"miter_type" "default"))
(sdegeo:sweep (find-face-id (position 4.5 0.5 0)) (position 3 0 0)
  (gvector 0 1 0)
  (sweep:options "solid" #t "sweep_angle" 40 "rigid" #f "miter_type"
  "default"))
```

Figure 50 Sweeping around an axis using different options



Corner Sweep

Use this option to design crown moldings, corbels, and so on.

To perform a corner sweep:

1. Follow the procedure in [Regular Sweep on page 126](#).
2. In Step 9, select **Sweep Corner**.
3. Enter or pick a point on the sweep axis.

The axis direction is determined automatically, and the sweep angle for corner sweeps is always 90°.

The sweep is performed in such a way that the resulting shape contains a solid angle.

Note:

If the selection of the sweep axis does not allow for the creation of a solid angle, the sweep operation is suppressed.

The corresponding Scheme command is:

```
(sdegeo:sweep-corner face|face-list position gvector)
```

The example in [Figure 51](#) is generated using the following Scheme commands:

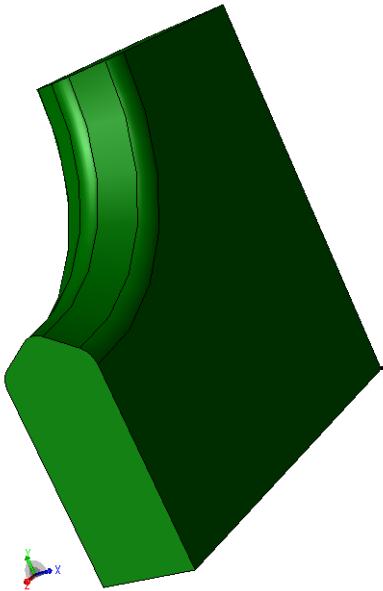
```
(sdegeo:create-rectangle (position 0 0 0.0) (position 1 2 0.0)
  "Silver" "region_1")
(define NEWVERT (sdegeo:insert-vertex (position 0.5 2 0)))
(sdegeo:move-vertex NEWVERT (position 0.5 2.3 0))
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

```
(sdegeo:fillet-2d (sde>window-select-2d -0.1 1.9 1.1 2.4 "all"
    "vertex") 0.2)
(sdegeo:sweep-corner (find-face-id (position 0.5 0.5 0))
    (position -0 4 0) (gvector 1 0 0))
```

Figure 51 Corner sweeping around an axis; the initial profile is the same as in [Figure 50](#)



Sweep Along a Wire

To sweep a profile along a wire, first, the wire body must be generated either manually by using Scheme functions or by using the GUI. To create a wire, see [Wires on page 118](#).

To sweep a 2D object button (see [Table 6 on page 46](#)):

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Select the face and hold the Shift key to select additional faces, or drag a box around a set of faces.
4. Choose **Edit > 3D Edit Tools > Sweep**.

The Sweep Operations dialog box opens.

5. Click **Select Profiles**.

The entity IDs of the selected profiles are displayed in the **Sweep Profiles** field.

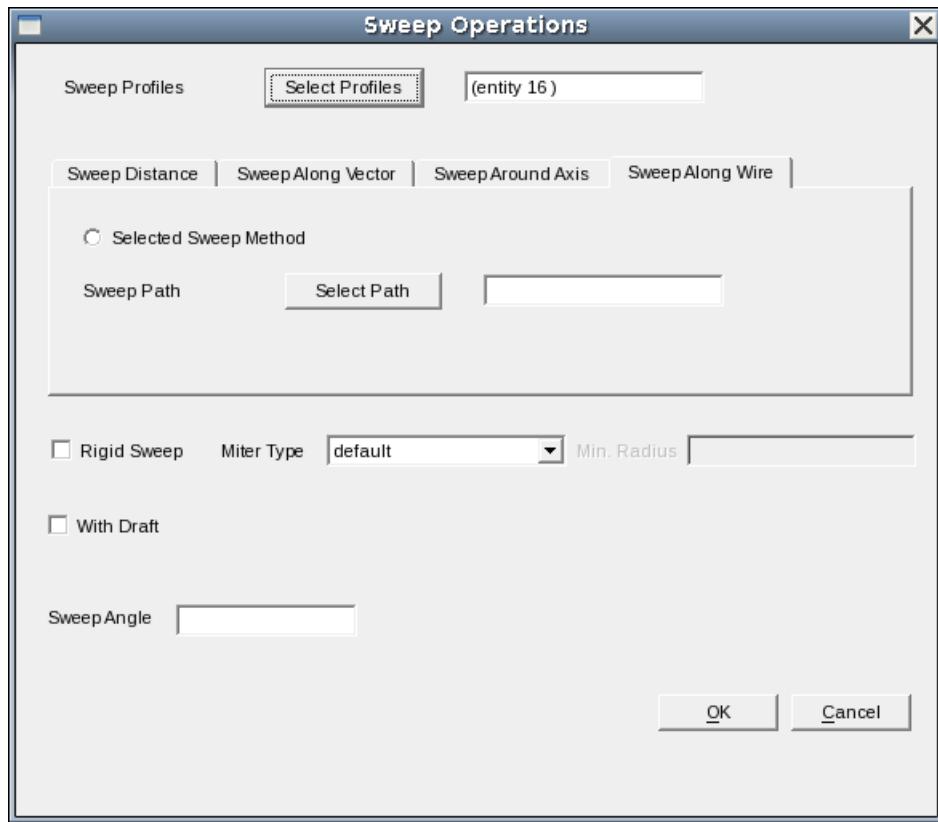
6. Click the **Sweep Along Wire** tab.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

7. Select **Selected Sweep Method**.
8. From the Selection Level list, select **Select Edge**, and select the wire in the view window.
9. Click **Select Path**. The entity IDs of the selected wires are displayed in the **Sweep Path** field.
10. Select sweep options as needed (see [Sweep Options on page 131](#)).
11. Click **OK**.

Figure 52 Sweep Operations dialog box showing Sweep Along Wire tab



The corresponding Scheme command is:

```
(sdegeo:sweep face|face-list position gvector  
  (sweep:options "sweep_angle" angle [other-sweep-options]))
```

The example in [Figure 53](#) is generated using the following Scheme commands:

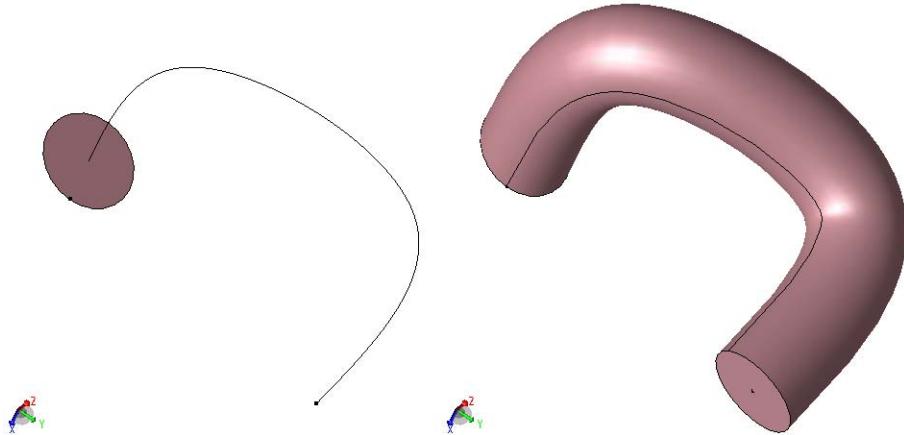
```
(define FACE (sdegeo:create-circle (position 0.0 0.0 0.0) 1.0  
  "Silicon" "region_1"))  
(define WIRE (sdegeo:create-spline-wire (list (position 0.0 0.0 0.0)  
  (position 0.0 0.0 5.0) (position 0.0 5.0 5.0)  
  (position 5.0 5.0 5.0))))
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

```
(sdegeo:sweep FACE WIRE (sweep:options "solid" #t "rigid" #f  
"miter_type" "default"))  
(entity:delete WIRE)
```

Figure 53 Wire sweep: (left) initial profile with sweep wire and (right) final swept body



Sweep Options

The sweep operation can be performed with different options that affect the outcome of the operation. These options are controlled by the `sweep:options` command. For example, when a profile is swept along a path, the profile can remain parallel to the original profile (rigid sweep) or can follow the path in a more natural way, following the curvature of the path, remaining perpendicular to the path during sweeping. The following sections give an overview of the available sweep options.

Rigid Sweep

A rigid sweep is one in which the profile that is swept is translated, but not rotated, along the sweep path. It accomplishes a rigid extrusion of a profile along a path. The `rigid` option specifies whether to make the cross sections of a sweep parallel to each other. The default is `FALSE`, which means the cross sections are perpendicular to the sweep path. No checks are made when a rigid sweep is performed; consequently, the resulting surface is self-intersecting if the given path is closed or if the direction of the path changes by more than 180°.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Mitering

The `miter` option determines how a corner miter is created during a sweep operation. The available mitering options are:

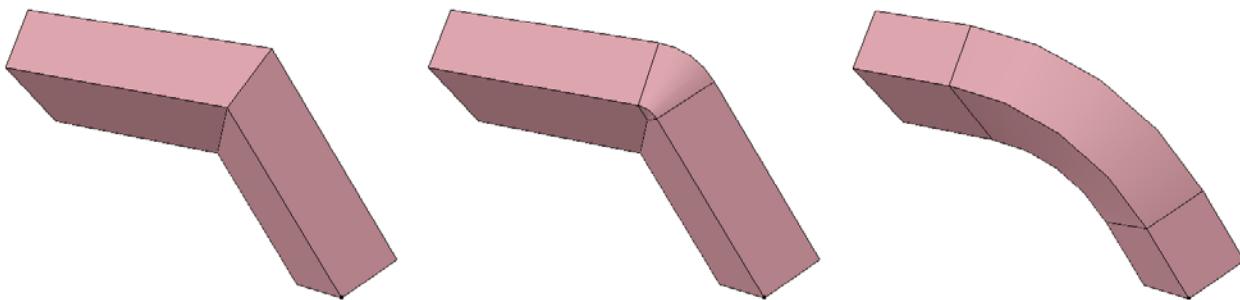
- `new` Reflects the profile to the other side of the corner up to the discontinuous point. The two sides are then extended and intersected, and new edges are formed as necessary. The 'ending' profile is the same as the starting profile. The `new` option is the default mitering.
- `old` Intersects the plane that is perpendicular to the path at the half angle of the corner. The resulting profile on the plane is then swept continuously along the path.
- `crimp` Reflects the profile to the other side of the corner up to the discontinuous point. The portions of the two sides that do not intersect are connected using a smooth rotation about the discontinuous point.
- `bend` Bend mitering requires the specification of a minimum radius to fillet the path. The result is a smooth, curved junction. The minimum radius must be positive.

The `new` setting makes the sweep operation create miters by sweeping each G1 part of the path with a copy of the profile and then filling in the gaps with linear-extended parts that are sliced at the miter planes.

The `crimp` setting creates miters between profiles swept along G1-continuous path segments, filling in gaps by revolving the profile.

G1 continuity refers to geometric continuity of the first degree (that is, the direction of the tangent vector is continuous). Two curves are G1-continuous at a given point if the directions of the tangent vectors at their joining point are the same, even though the vector magnitudes might differ (in the case of C1 continuity, the magnitudes are also the same). G1 continuity means that the tangent lines are smoothly connected.

Figure 54 Mitering a corner with different options: (left) 'new' and 'old' give the same results for the given example, (middle) 'crimp,' and (right) 'bend'



Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

The examples in [Figure 54](#) are created by the following Scheme commands:

```
(define REC (sdegeo:create-rectangle (position -1.0 0.0 0.0)
                                         (position 1.0 1.0 0.0) "Silicon" "region_1"))
(define WIRE (sdegeo:create-polyline-wire (list (position 0.0 0.5 0.0)
                                                 (position 0.0 0.5 3.0) (position 2.0 2.5 5.0))))
; (a): "new"
(sdegeo:sweep REC WIRE
  (sweep:options "solid" #t "rigid" #f "miter_type" "new"))
; (a): "old"
(roll)
(sdegeo:sweep REC WIRE
  (sweep:options "solid" #t "rigid" #f "miter_type" "old"))
; (b): "crimp"
(roll)
(sdegeo:sweep REC WIRE
  (sweep:options "solid" #t "rigid" #f "miter_type" "crimp"))
; (c) "bend"
(roll)
(sdegeo:sweep REC WIRE
  (sweep:options "solid" #t "rigid" #f "miter_type" "bend" 3.5))
```

Sweeping Examples

The following example illustrates the creation of a simple MOSFET with a nonaxis-aligned gate stack using different sweep methods.

Sweep Distance

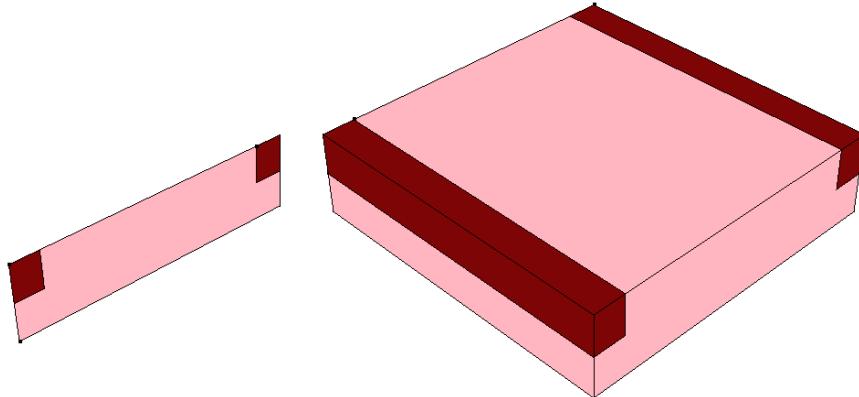
The substrate with two shallow trench isolations is created by sweeping the 2D cross section a given distance:

```
(define SUB (sdegeo:create-rectangle
             (position -2.0 0.0 0.0) (position 2.0 0.0 -1.0) "Silicon" "R.Sub"))
(sdegeo:set-default-boolean "ABA")
(define STI_L (sdegeo:create-rectangle
               (position -2.0 0.0 0.0) (position -1.6 0.0 -0.5) "Oxide" "R.STI_L"))
(define STI_R (sdegeo:create-rectangle
               (position 1.6 0.0 0.0) (position 2.0 0.0 -0.5) "Oxide" "R.STI_R"))
(sdegeo:sweep (list SUB STI_L STI_R) -4.0
  (sweep:options "solid" #t "rigid" #f "miter_type" "default"))
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 55 Distance sweeping creates 3D substrate with shallow trench isolations

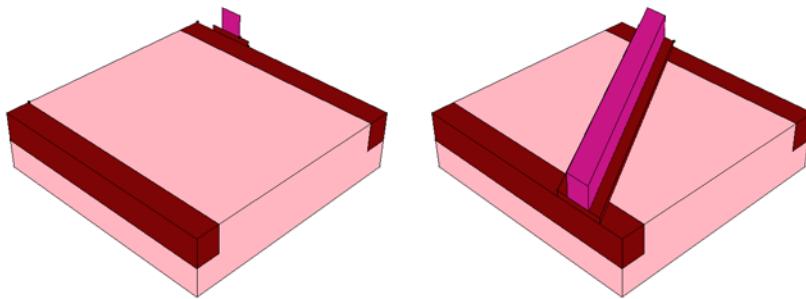


Sweep Along a Vector

In a first version, the 2D gate-stack cross section is swept along a vector:

```
(define GOX (sdegeo:create-rectangle
    (position -2.0 0.4 0.0) (position -2.0 1.2 0.1) "Oxide" "R.Gox"))
(define POL (sdegeo:create-rectangle
    (position -2.0 0.6 0.1) (position -2.0 1.0 0.5) "PolySilicon"
    "R.Pol"))
(sdegeo:sweep (list GOX POL) (gvector 4.0 2.4 0.0)
    (sweep:options "solid" #t "rigid" #f "miter_type" "default"))
```

Figure 56 Vector sweeping creates a nonaxis-aligned gate stack: gate stack profiles (left) before vector sweeping and (right) after vector sweeping



Sweep Along a Wire

In a second version, the 2D gate-stack cross section is swept along a wire:

```
(roll)
(define WIRE (sdegeo:create-polyline-wire (list
    (position -2.0 0.8 0.5) (position -0.8 0.8 0.5)
    (position 0.8 3.2 0.5) (position 2.0 3.2 0.5))))
(sdegeo:sweep (list GOX POL) WIRE
    (sweep:options "solid" #t "rigid" #f "miter_type" "default"))
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

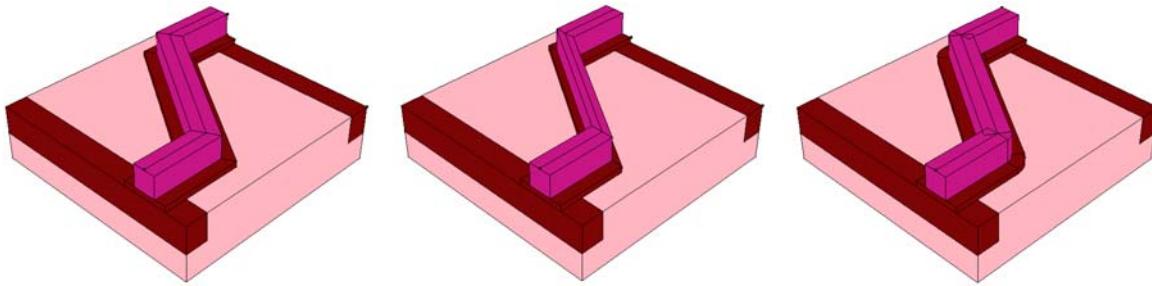
In a third version, the 2D gate-stack cross section is swept along a wire rigidly:

```
(roll)
(sdegeo:sweep (list GOX POL) WIRE
  (sweep:options "solid" #t "rigid" #t "miter_type" "default"))
```

In a final version, crimping is used to round the edges that are created along the sweep path:

```
(roll)
(sdegeo:sweep (list GOX POL) WIRE
  (sweep:options "solid" #t "rigid" #f "miter_type" "crimp"))
(entity:delete WIRE)
```

Figure 57 Wire sweeping creates a meandering gate stack: (left) default, (middle) rigid sweep, and (right) crimping



Sweep Around an Axis

The next example discusses the creation of a simple racetrack LDMOS structure using sweeping along a wire for the curved part and distance sweeping to add the straight parts:

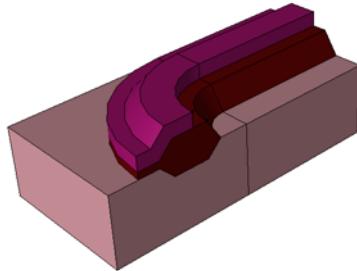
```
(define LOCOS (sdegeo:create-polygon (list
  (position -0.6 0 0.0) (position -0.4 0 -0.2) (position 0.1 0 -0.2)
  (position 0.3 0 0.0) (position 0.6 0 0.0) (position 0.6 0 0.1)
  (position 0.3 0 0.1) (position 0.1 0 0.3) (position -0.4 0 0.3)
  (position -0.6 0 0.0)) "Oxide" "R.LOCOS"))
(define POLY (sdegeo:create-polygon (list
  (position -0.2 0 0.3) (position 0.1 0 0.3) (position 0.3 0 0.1)
  (position 0.6 0 0.1) (position 0.6 0 0.3) (position 0.4 0 0.3)
  (position 0.2 0 0.5) (position -0.2 0 0.5) (position -0.2 0 0.3)))
  "PolySilicon" "R.Poly"))
(sdegeo:sweep (list LOCOS POLY) (position -1 0 0) (gvector 0 0 1)
  (sweep:options "solid" #t "sweep_angle" 90 "rigid" #f "miter_type"
  "default"))

(sdegeo:set-default-boolean "BAB")
(define SUB (sdegeo:create-cuboid
  (position -1.0 0.0 0.0) (position 1.0 2.0 -1.0) "Silicon" "R.SUB"))
(sdegeo:sweep (list
  (car (find-face-id (position 0.0 0 -0.9)))
  (car (find-face-id (position 0.0 0 0.0)))
  (car (find-face-id (position 0.0 0 0.4)))) 2.0
  (sweep:options "solid" #t "rigid" #f "miter_type" "default"))
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 58 Creating a racetrack LDMOS using sweep around an axis



Skinning

A skinning operation can be used to create a 3D body that interpolates a series of wires or faces. The given wires or faces define the cross sections to be interpolated by the resulting 3D body. There must be at least two wire bodies or faces to be selected. The wires can be open or closed. The wires are copies, that is, the originals remain. The wires can share endpoints and do not have to be C1 continuous.

Default Skinning

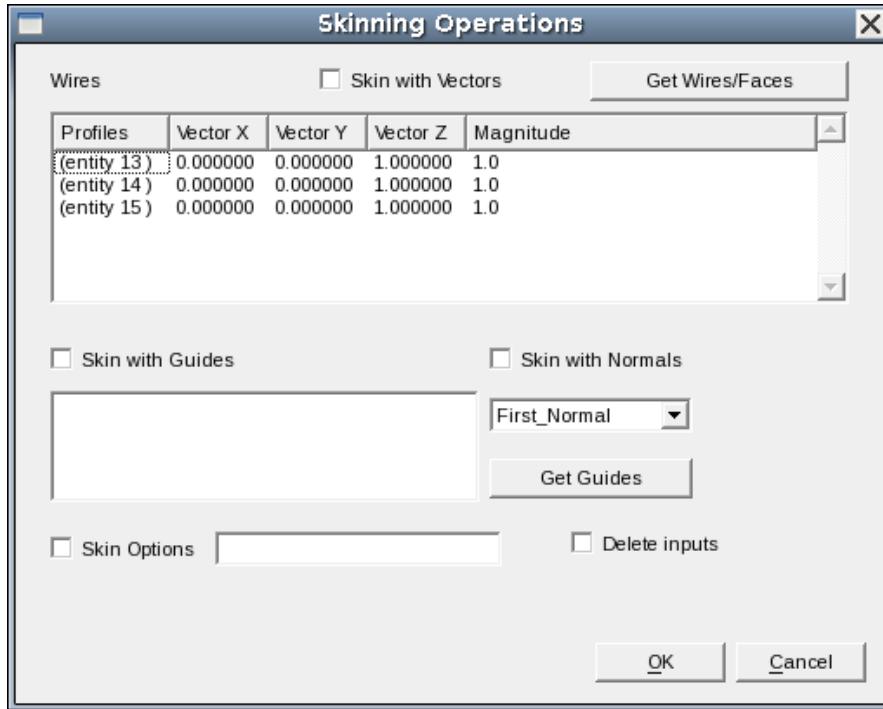
To skin a set of wire bodies or 2D faces using the default options:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Choose **Edit > 3D Edit Tools > Skin**.
The Skinning Operations dialog box opens.
4. Select the first face or wire in the view window, and click **Get Wires/Faces**.
The selected profile appears in the Wires list.
5. Repeat for all other wires and faces.
6. Enter additional skinning options as needed (go to <https://doc.spatial.com/> for a list of options).
7. Select the **Delete Inputs** option if the initial profiles are to be deleted.
8. Click **OK**.

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 59 Skinning Operations dialog box



The corresponding Scheme command is:

```
(sdegeo:skin-wire face-list [skin-options])
```

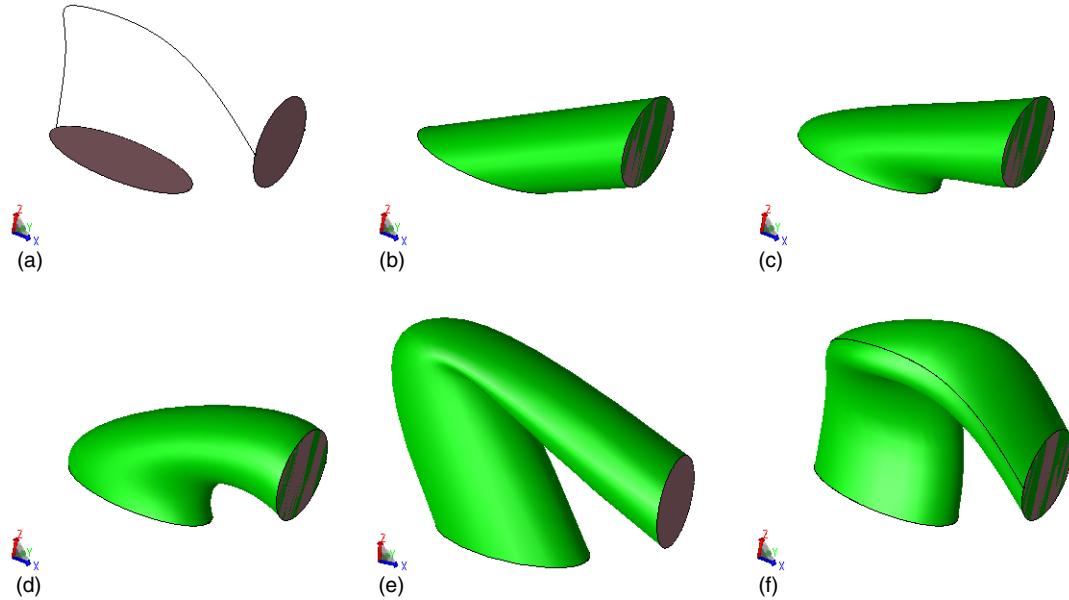
The example in Figure 60 (b) is generated using the following Scheme commands:

```
(define FACE_1 (sdegeo:create-ellipse (position 0 2 0)
                                         (position 1 2 0) 0.5 "Silicon" "region_1"))
(sdegeo:set-active-work-plane "wp_yz")
(define FACE_2 (sdegeo:create-circle (position 2 1 2) 0.5
                                         "Silicon" "region_2"))
(sdegeo:set-active-work-plane "base")
(define FACES (list (car (entity:faces FACE_1)) (car (entity:faces
FACE_2))))
(sdegeo:skin-wires FACES)
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Figure 60 Skinning option overview: (a) initial faces, with guide used in (e); (b) default skinning; (c) skinning with first normal; (d) skinning with vectors, low magnitude; (e) skinning with vectors, high magnitude; and (f) skinning with guide



Skinning With Normals

The default options automatically select the sweep tangent vectors at each of the profiles. To select the face normal as the sweep tangent vectors:

1. Select **Skin with Normals** in the Skinning Operations dialog box.
2. Select the face normals to be used as constraints from the list. The options are First_Normal, Last_Normal, Ends_Normal, or All_Normal.

The corresponding Scheme command is:

```
(sdegeo:skin-wires-normal face-list normal-selector [skin-options])
```

The example in [Figure 60](#) (c) is generated using the following Scheme commands:

```
(define FACE_1 (sdegeo:create-ellipse (position 0 2 0)
                                         (position 1 2 0) 0.5 "Silicon" "region_1"))
(sdegeo:set-active-work-plane "wp_yz")
(define FACE_2 (sdegeo:create-circle (position 2 1 2) 0.5
                                      "Silicon" "region_2"))
(sdegeo:set-active-work-plane "base")
(define FACES (list (car (entity:faces FACE_1)) (car (entity:faces
                                                       FACE_2))))
(sdegeo:skin-wires-normal FACES "first_normal")
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

Skinning With Vectors

Use the **Skin with Vectors** option to explicitly control sweep tangent vectors. For each vector, a magnitude is given, which controls how quickly the sweep direction can deviate from the given tangent vector away from the profile. For larger values, the sweep direction follows the tangent vector longer than for small values (see [Figure 60](#) (d) and (e)).

The corresponding Scheme command is:

```
(sdegeo:skin-wires-vectors faces-list gvector-list magnitude-list  
[skin-options])
```

The example in [Figure 60](#) (d) is generated using the following Scheme commands:

```
(define FACE_1 (sdegeo:create-ellipse (position 0 2 0)  
                                         (position 1 2 0) 0.5 "Silicon" "region_1"))  
(sdegeo:set-active-work-plane "wp_yz")  
(define FACE_2 (sdegeo:create-circle (position 2 1 2) 0.5  
                                         "Silicon" "region_2"))  
(sdegeo:set-active-work-plane "base")  
(define FACES (list (car (entity:faces FACE_1))  
                     (car (entity:faces FACE_2))))  
(sdegeo:skin-wires-vectors FACES  
                           (list (gvector -0.5 0 1) (gvector 1 0 -0.5))  
                           (list 2.0000 2.0000))
```

The example in [Figure 60](#) (e) is generated with the same Scheme commands, but uses magnitudes of 10.0 instead of 2.0.

Skinning With Guides

Similar to sweeping along a wire, a set of profiles can be skinned using a wire as a guide. To use this option:

1. Select **Skin with Guides** in the Skinning Operations dialog box.
2. Select the wire in the view window.
3. Click **Get Guide**.

The corresponding Scheme command is:

```
(sdegeo:skin-wire-guide face-list wire [skin-options])
```

The example in [Figure 60](#) (f) is generated using the following Scheme commands:

```
(define FACE_1 (sdegeo:create-ellipse (position 0 2 0)  
                                         (position 1 2 0) 0.5 "Silicon" "region_1"))  
(sdegeo:set-active-work-plane "wp_yz")  
(define FACE_2 (sdegeo:create-circle (position 2 1 2) 0.5 "Silicon"  
                                         "region_2"))  
(sdegeo:set-active-work-plane "base")  
(define FACES (list (car (entity:faces FACE_1)) (car (entity:faces
```

Chapter 4: Generating Geometric Structures

Creating 3D Objects From 1D and 2D Objects

```
FACE_2)))))

(define WIRE (sdegeo:create-spline-wire (list
    (position -1.0 2.0 0.0) (position -1.0 2.0 1.0)
    (position 0.0 1.0 2.0)
    (position 1.0 1.0 2.0) (position 2.0 1.5 1.0))))
    (position 0.8 3.2 0.5) (position 2.0 3.2 0.5)))
(sdegeo:skin-wires-guides FACES WIRE)
```

Skinning Example

The following example illustrates how to use skinning to emulate line-edge roughness effects in a MOSFET gate stack. First, a set of perturbed wires for the polygate and the gate-oxide cross sections are created (see [Figure 61](#)). Second, these wires are skinned and the substrate is added.

The example is generated using the following Scheme commands:

```
(define index 0) (define PolWires (list)) (define OxiWires (list))
(define MakeWires (lambda (Xl Xr Yo)
    (define PWname (string-append "PolWire_" (number->string index)))
    (define OWname (string-append "OxiWire_" (number->string index)))
    (define PWname (sdegeo:create-polyline-wire (list
        (position Xl Yo 0.1) (position Xr Yo 0.1)
        (position (- Xr 0.1) Yo 0.5) (position (+ Xl 0.1) Yo 0.5)
        (position Xl Yo 0.1))))
    (define OWname (sdegeo:create-polyline-wire (list
        (position Xl Yo 0.0) (position Xr Yo 0.0)
        (position Xr Yo 0.1)
        (position Xl Yo 0.1) (position Xl Yo 0.0))))
    (set! index (+ index 1))
    (set! PolWires (append PolWires (list PWname)))
    (set! OxiWires (append OxiWires (list OWname)))
))

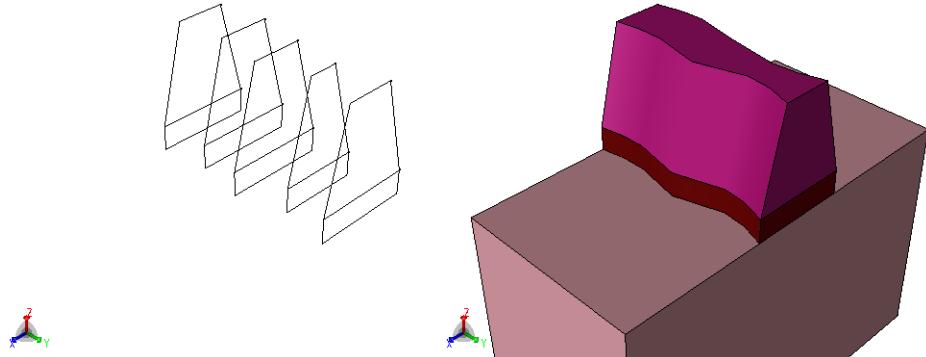
(MakeWires -0.20 0.20 0.0)
(MakeWires -0.23 0.18 0.2)
(MakeWires -0.19 0.22 0.4)
(MakeWires -0.16 0.16 0.6)
(MakeWires -0.20 0.20 0.8)

(define POL (sdegeo:skin-wires PolWires))
(sde:add-material POL "PolySilicon" "R.Pol")
(define OXI (sdegeo:skin-wires OxiWires))
(sde:add-material OXI "Oxide" "R.oxi")
(entity:delete PolWires)
(entity:delete OxiWires)
(sdegeo:create-cuboid (position -0.8 0.0 0.0) (position 0.8 0.8 -1.0)
    "Silicon" "R.SUB")
```

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures

Figure 61 Skinning example of MOSFET with line-edge roughness: (left) wire bodies and (right) final structure



Editing 2D and 3D Structures

This section discusses how to edit 2D and 3D structures.

Explicit Boolean Operations

To perform Boolean operations on 3D (or 2D) regions:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Body**.
3. Select the body and hold the Shift key to select additional bodies, or drag a box around a set of bodies.
4. Choose **Edit > 3D Edit Tools** (or **2D Edit Tools**).
5. Select **Unite**, **Intersect**, or **Subtract**.

The following multiregion operations can be performed.

Unite

Dissolves any internal boundaries that exist in touching or overlapping regions and results in one single region with the name of the first region selected before the unite operation was applied.

The corresponding Scheme command is:

```
(sdegeo:bool-unite body-list)
```

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures

For example:

```
(define A (sdegeo:create-cuboid (position 0.0 0.0 0.0)
    (position 2.0 1.0 1.0) "Silicon" "region_1"))
(define B (sdegeo:create-cuboid (position 2.0 0.0 0.0)
    (position 3.0 3.0 1.0) "Silicon" "region_2"))
(sdegeo:bool-unite (list A B))
```

Intersect

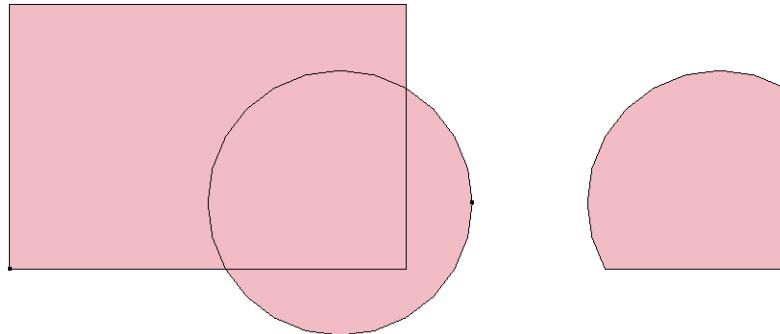
This operation is used if the direct ACIS Scheme functions are used to generate parts of the structure or if the automatic overlap resolution is disabled. The corresponding Scheme command is:

```
(sdegeo:bool-intersect body-list)
```

For example:

```
(sdegeo:set-default-boolean "XX")
(define A (sdegeo:create-rectangle (position 0.0 0.0 0)
    (position 3.0 2.0 0) "Silicon" "region_1"))
(define B (sdegeo:create-circle (position 2.5 1.5 0) 1 "Silicon"
    "region_2"))
(sdegeo:bool-intersect (list A B))
```

Figure 62 *Explicit Boolean intersection: (left) original structure and (right) after intersection*



Subtract

The first region in the body list is selected as the main region, then subsequent regions are subtracted from the main region. The corresponding Scheme command is:

```
(sdegeo:bool-subtract body-list)
```

For example:

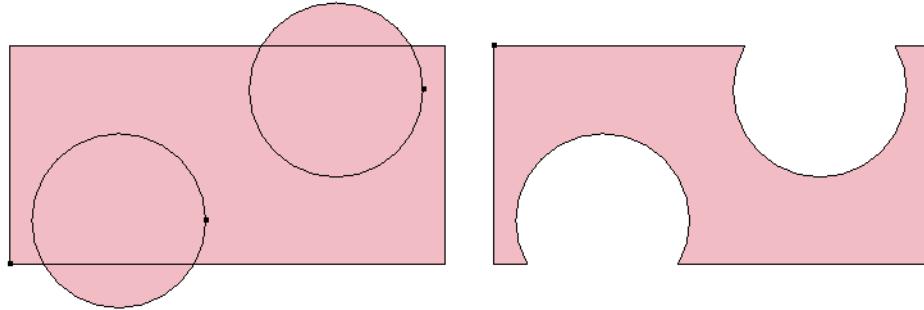
```
(sdegeo:set-default-boolean "XX")
(define A (sdegeo:create-rectangle (position 0.0 0.0 0)
    (position 2.0 1.0 0) "Silicon" "region_1"))
(define B (sdegeo:create-circle (position 0.5 0.8 0) 0.4
    "Silicon" "region_2"))
(define C (sdegeo:create-circle (position 1.5 0.2 0) 0.4
```

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures

```
"Silicon" "region_2"))
(sdegeo:bool-subtract (list A B C))
```

Figure 63 *Explicit Boolean subtraction: (left) original structure and (right) after subtraction*



Changing the Material of a Region

To change the material associated with a region or regions:

1. Click the Select button (see [Table 6 on page 46](#)).
2. From the Selection Level list, select **Select Body**.
3. Select the body and hold the Shift key to select additional bodies, or drag a box around a set of bodies.
4. Select the required material from the Material list.
5. Choose **Edit > Change Material**.

The corresponding Scheme command is:

```
(sde:add-material [body | body-list] new-material-name region-name)
```

Note:

If a list with more than one body is used, all bodies will be assigned the same region name. Upon saving the boundary, the region will be made unique.

For example, to change the material but preserve the region name:

```
(define A (sdegeo:create-rectangle (position 0.0 0.0 0)
  (position 3.0 2.0 0) "Silicon" "region_1"))
(sde:add-material A "PolySilicon" (generic:get A "region"))
```

Changing the Name of a Region

To change the name of a region or regions:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Body**.
3. Select a single body.
4. Choose **Edit > Change Region Name**.
5. Enter the new name in the dialog box.

The corresponding Scheme command is:

```
(sde:add-material body material-name new-region_name)
```

For example, to change the region name but to preserve the material:

```
(define A (sdegeo:create-rectangle (position 0.0 0.0 0)
                                      (position 3.0 2.0 0) "Silicon" "region_1"))
(sde:add-material A (generic:get A "material") "region_new")
```

Note:

If multiple bodies are selected, only the region name of the first selected body is changed.

No check is performed to ensure that a newly chosen region name is not already in use. If a body is assigned a region name through this method and the region name is also used for another body, the results of region-oriented operations are undefined.

Deleting Regions

To delete one or more regions:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Body**.
3. Select the body and hold the Shift key to select additional bodies, or drag a box around a set of bodies.
4. Choose **Edit > 2D Edit Tools > Delete Region**, or press the Delete key or Backspace key, or right-click and choose **Delete**.

The corresponding Scheme command is:

```
(sdegeo:delete-region body-list)
```

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures

For example:

```
(define BODY (sdegeo:create-cuboid (position 0.0 0.0 0.0)
    (position 1.0 1.0 1.0) "Silicon" "region_1"))
(sdegeo:delete-region BODY)
```

Note:

Ref/Eval windows can be deleted in a similar way. The selection filter must be set to Ref/Eval windows to select the Ref/Eval window entities.

Separating Lumps

During model generation, Sentaurus Structure Editor *never* allows the generation of overlapping regions (unless `(sdegeo:set-default-boolean "XX")` is used). When the automatic region-naming feature is switched on, a region name is attached automatically to each generated body. As a result of this, bodies with multiple lumps (disjoint 2D or 3D parts) can be created. These disjoint lumps have the same region-name attribute.

Since this could cause a problem during meshing, these lumps must be separated and a unique region name must be assigned to each lump. This is performed automatically when a boundary file is exported.

To call the lump separation explicitly, for example, to refer to individual lumps of a multiple-lump regions in a Scheme script, choose **Edit > Separate Lumps**. The same action can be performed by using the Scheme function (`sde:separate-lumps`).

The following script illustrates how to generate multiple lumps:

```
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position 0 0 0) (position 10 5 0)
    "Silicon" "region_1")
(sdegeo:create-rectangle (position 4 -2 0) (position 6 7 0)
    "PolySilicon" "region_2")
```

As a result, two regions are created (`region_1` and `region_2`). However, `region_1` is composed of two disjoint lumps. After `(sde:separate-lumps)`, the `region_1` region-name attribute is removed from the silicon region, and `region_1_lump_1` is assigned as the region name to the left lump of the silicon body, and `region_1_lump_2` is assigned to the right lump. The exported boundary file will have three regions with unique region names.

Two-Dimensional Cuts From a 3D Structure

To generate a 2D cross section from a 3D structure:

1. Choose **Edit > 3D Edit Tools > Slice**.

The Cut Plane Controller opens.

2. Select the required option, and either change the value in the box or rotate the corresponding dial.

3. When you are finished, click **Close**.

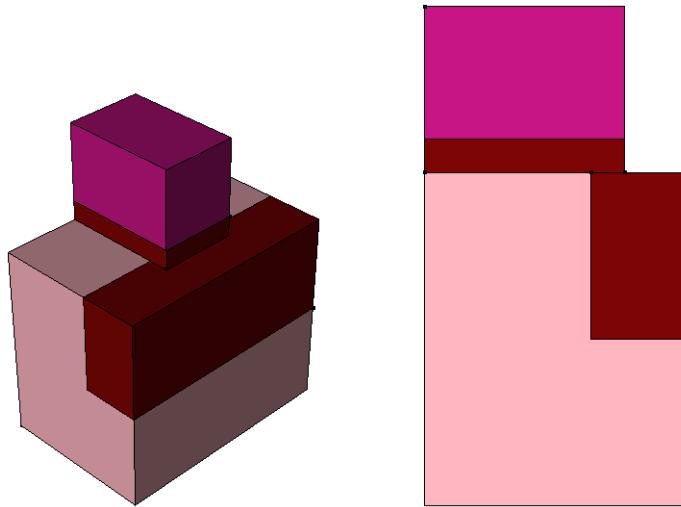
The corresponding Scheme extension is:

```
(sdegeo:3d-cut base-position normal-vector [tr-xy] [tr-record]  
[tr-axisaligned])
```

The Scheme extension needs two arguments: a position argument to place the cut plane and a gvector argument to specify the cut plane normal. For example, the following command generates a 2D cut in such a way that the cut plane goes through 0.0 0.0 0.0, and the cut plane is perpendicular to the 1.0 1.0 0.0 vector:

```
(sdegeo:3d-cut (position 0.0 0.0 0.0) (gvector 1.0 1.0 0.0))
```

Figure 64 (Left) Three-dimensional structure and (right) 2D structure generated after slicing



Up to three additional (optional) arguments can be used in `sdegeo:3d-cut`:

- The first optional Boolean argument determines the placement of the newly created 2D structure. If you specify `#f` (default), then the slice remains at its original location. If you specify `#t`, then the slice will move to the xy plane at $z=0$. In this way, a 2D slice can be saved to a 2D boundary file.

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures

- The second optional Boolean argument can be used to record the transformation that transforms the 2D cut on the xy plane.
- The third optional Boolean argument can be used to trigger special transformation rules for axis-aligned cuts. If the new transformation rule is applied, then the position of the 2D cut is calculated explicitly following the simple rules described in [sdegeo:3d-cut on page 589](#).

For example:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.2 0.8 1.0)
    "Silicon" "R.Substrate")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-cuboid (position 0.0 0.5 0.5) (position 1.2 0.8 1.0)
    "Oxide" "R.STI")
(sdegeo:create-cuboid (position 0.4 0.0 1.0) (position 0.8 0.6 1.1)
    "Oxide" "R.Gox")
(sdegeo:create-cuboid (position 0.4 0.0 1.1) (position 0.8 0.6 1.5)
    "PolySilicon" "R.Poly")
(sdegeo:3d-cut (position 0.6 0.0 1.0) (gvector 1.0 0.0 0.0) #t)
```

Split–Insert Stretching of a Device

For certain applications, such as the computation of the threshold voltage roll-off behavior of a CMOS technology node, it is necessary to create a family of device structures, which are identical in certain regions (here, the source and drain areas) while other regions are stretched (here, the gate area). The split–insert stretching feature conveniently supports this application.

It allows you to split a device at a specified spatial location and to insert a linear segment (with a specified extension length) between the split parts of the original device. The split–insert direction can be specified as well. An optional merge argument can be used to merge the split bodies and inserted bodies.

To use the split–insert stretching feature:

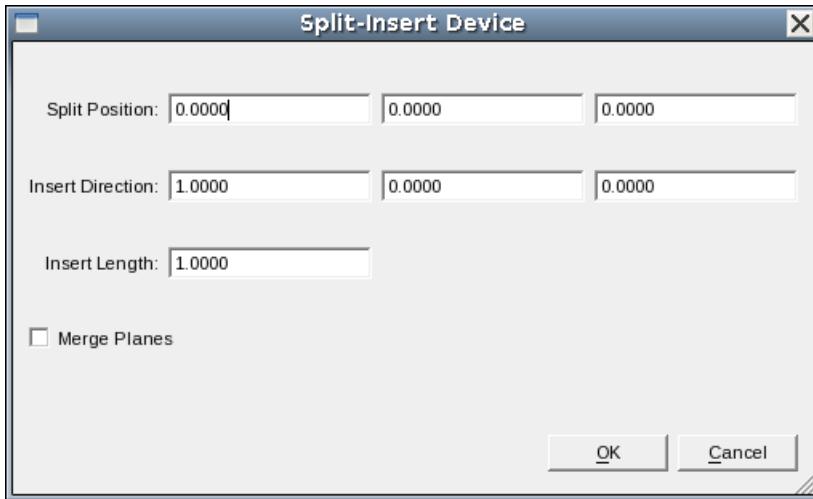
1. Choose **Edit > Device Operations > Split-Insert Device**.

The Split-Insert Device dialog box opens.

2. Enter the split position, insert direction, and insert length.
3. Select **Merge Planes** if the inserted regions should be merged with the existing ones.

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures



4. Click **OK**.

The corresponding Scheme command is:

```
(sdegeo:split-insert-device position gvector insert-length  
[merge-flag])
```

For example:

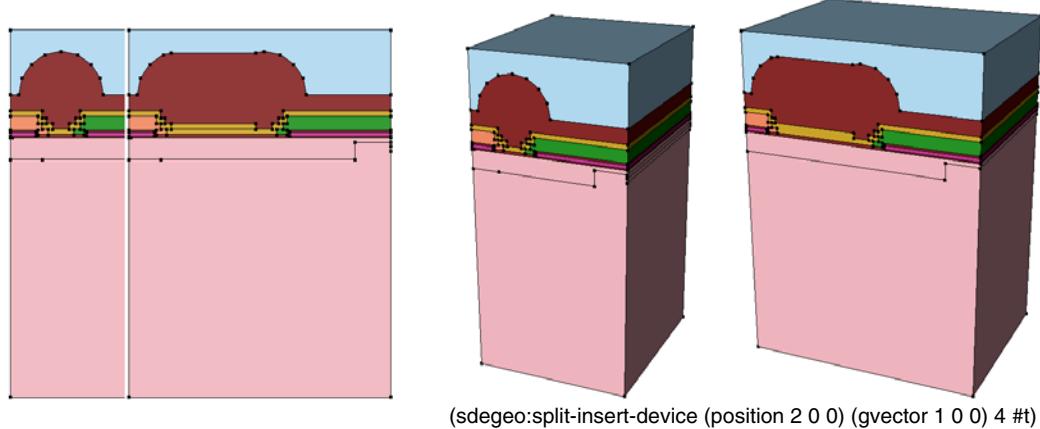
```
(sdegeo:create-cuboid (position -1.0 0.0 0.0) (position 1.0 1.0 -1.0)  
"Silicon" "region_1")  
(sdegeo:create-cuboid (position -0.4 0.0 0.0) (position 0.4 1.0 0.1)  
"SiO2" "region_2")  
(sdegeo:create-cuboid (position -0.4 0.0 0.1) (position 0.4 1.0 0.5)  
"PolySi" "region_3")  
(sdegeo:split-insert-device (position 0 0 0) (gvector 1 0 0) 1 #t)
```

These split–insert operations can be used for both 2D and 3D models.

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures

Figure 65 Split–insert operations in (left) two dimensions and (right) three dimensions



Extending a 2D Device

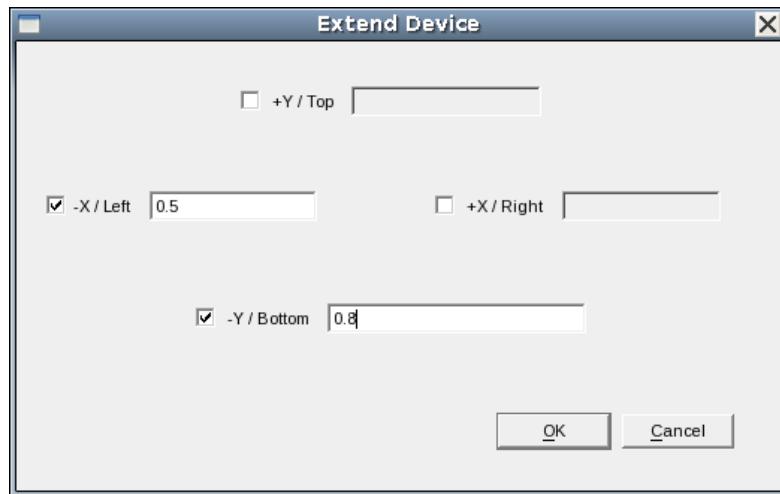
For certain applications, it is necessary to extend the device at one or more boundaries. For example, the default reflecting boundary conditions might be inappropriate for a given application. In such cases, it can be advantageous to simply push out one or more boundaries. This can be achieved conveniently with the extend device feature.

To use the extend device feature:

1. Choose **Edit > Device Operations > Extend Device**.

The Extend Device dialog box opens.

2. Select the boundary to be extended, and enter the extension length.



3. Click **OK**.

Chapter 4: Generating Geometric Structures

Editing 2D and 3D Structures

The corresponding Scheme extension is:

```
(sdegeo:extend-device side amount [side amount [side amount  
[side amount]]])
```

For example:

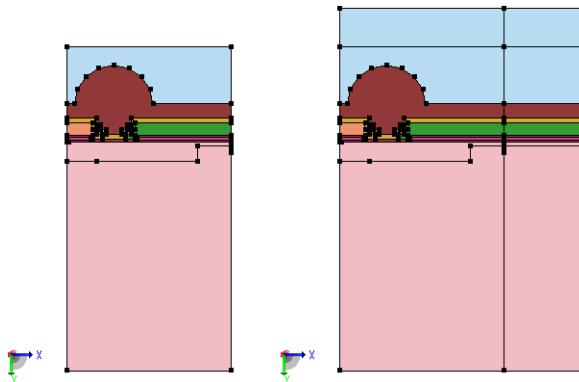
```
(sdegeo:create-rectangle (position 0.0 0.0 0.0)  
    (position 1.0 1.0 0.0) "Silicon" "region_1")  
(sdegeo:set-default-boolean "ABA")  
(sdegeo:create-rectangle (position 0.0 0.0 0.0)  
    (position 0.5 0.5 0.0) "SiO2" "region_2")  
(sdegeo:create-rectangle (position 0.0 -0.2 0.0)  
    (position 0.7 0.0 0.0) "Aluminum" "region_3")  
(sdegeo:extend-device "left" 0.5 "bottom" 0.5)
```

The left, right, top, and bottom parts of the device can be extended sequentially or in combination. If the combination contains a corner, the corner area will be filled as well. Individual region names will be assigned to the extended parts, and the material property is inherited from the original regions.

Note:

This operation is available for 2D only. Only the axis-aligned (horizontal or vertical) parts of the model will be extended.

Figure 66 (Left) Original 2D device and (right) extended device using the (sdegeo:extend-device "right" 4 "bottom" 2) command



Trimming a 3D Structure

A 3D structure can be trimmed such that all parts lying outside of a polygonal base plane are removed. The base plane is restricted to the xy plane. The command is not available from the GUI.

Chapter 4: Generating Geometric Structures

Coordinate Systems and Transformations

The Scheme command is:

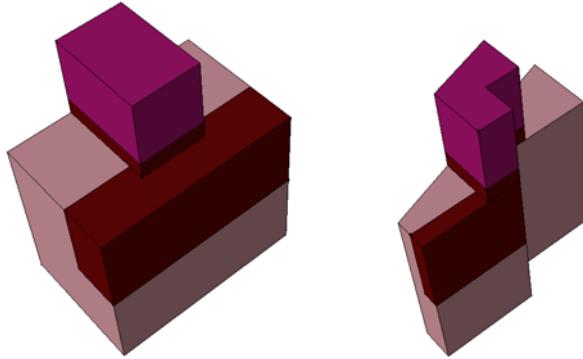
```
(sdegeo:chop-domain coordinate-list)
```

The coordinate-list argument is a list of x,y point pairs.

For example:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.2 0.8 1.0)
    "Silicon" "R.Substrate")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-cuboid (position 0.0 0.5 0.5) (position 1.2 0.8 1.0)
    "Oxide" "R.STI")
(sdegeo:create-cuboid (position 0.4 0.0 1.0) (position 0.8 0.6 1.1)
    "Oxide" "R.Gox")
(sdegeo:create-cuboid (position 0.4 0.0 1.1) (position 0.8 0.6 1.5)
    "PolySilicon" "R.Poly")
(sdegeo:chop-domain (list 0.0 0.0 1.2 0.4 1.2 0.6 0.6 0.6 0.6 0.4 0.0
    0.4))
```

Figure 67 Trimming a 3D structure: (left) initial structure and (right) trimmed structure



Coordinate Systems and Transformations

Sentaurus Structure Editor uses the concept of work planes to define a coordinate system. The name of the work plane refers to the fact that any 2D geometric object is created in the xy plane of the currently used coordinate system. In addition, certain 3D GUI-based geometric operations use the work plane as the plane of reference.

Changing the work plane only affects newly created objects (existing objects can be moved, rotated, and reflected using transformations).

Sentaurus Structure Editor itself makes no assumptions about the unit of length used to define a geometric object. Other TCAD Sentaurus tools, such as Sentaurus Device, however, will assume that the unit of length is 1 μm . In this indirect sense, the effective default unit of length in Sentaurus Structure Editor is 1 μm .

Sentaurus Structure Editor internally performs geometric operations with finite precision. Therefore, it is recommended to use a length scale in which the typical device dimensions are of the order of 1. For example, it is recommended to use nanometer rather than micrometer for a hypothetical device that measures only a few nanometers in length. In this case, it becomes necessary to scale the device structure upon saving the final boundary and the mesh input file. Sentaurus Structure Editor provides functions for scaling a device structure as well as meshing information before exporting the final boundary and mesh command file.

Work Planes

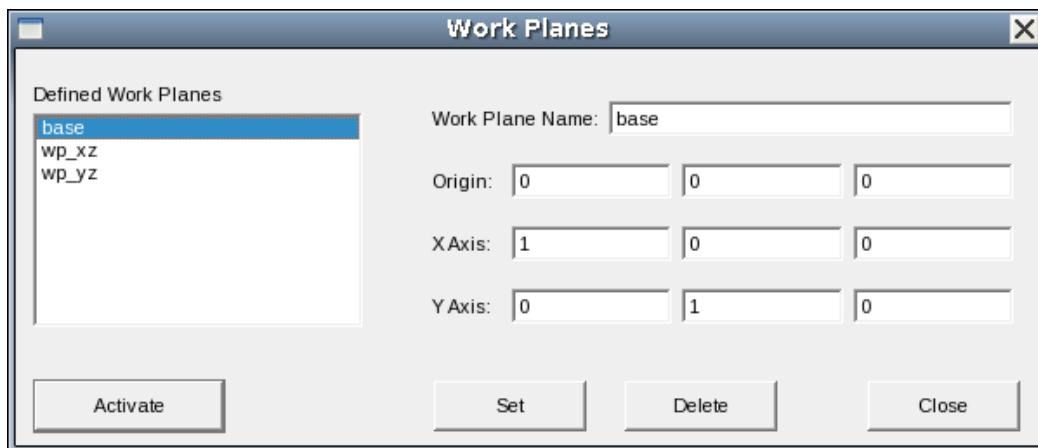
To define and activate a new work plane:

1. Choose **Draw > Work Planes**.

The Work Planes dialog box opens.

2. Enter a name for the new work plane, the coordinates of the new origin, as well as the direction vectors for the x- and y-axis.
3. Click **Set** to define the new work plane.
4. Select the newly created entry from the **Defined Work Planes** list, and click **Activate** to activate the new work plane.

Figure 68 *Work Planes dialog box*



The corresponding Scheme commands are:

```
(sdegeo:define-work-plane wp-name origin-position x-position  
y-position)  
(sdegeo:set-active-work-plane wp-name)
```

Chapter 4: Generating Geometric Structures

Coordinate Systems and Transformations

By default, Sentaurus Structure Editor uses the ‘base’ work plane, which is associated with a global coordinate system. The base work plane is defined as the xy plane at z = 0.

There are also two other predefined work planes. The ‘wp_xz’ work plane is defined as the xz plane at y = 0, and the ‘wp_yz’ work plane is defined as the yz plane at x = 0, both with respect to the global coordinate system. When you define and activate a different ‘work plane,’ then for all subsequent geometric operations, all coordinates are interpreted with respect to this new coordinate system.

To activate any work plane that has already been defined, select the required work plane from the **Defined Work Planes** list (see [Figure 68](#)).

The Work Planes dialog box and the Scheme command use different ways to define the new coordinate system. The dialog box asks for the direction *vectors* for the x- and y-axis. While the Scheme command uses three *positions* to define the work plane.

$$p_x = p_o + e_x$$

The two definitions are related using the expression:

$$p_y = p_o + e_y$$

where p_o , p_x , and p_y are the positions origin-position, x-position, and y-position used by the Scheme command, respectively, and e_x and e_y are the direction vectors for the x-axis and y-axis used by the dialog box.

If direction vectors do not have unit length, they are normalized. If the y-axis is not orthogonal to the x-axis, only the orthogonal component is considered. The z-axis is defined by the right-hand rule.

The following example shows how to create the six faces of a unit cube by drawing the same unit rectangles in different work planes:

```
(sdegeo:set-active-work-plane "base")
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silicon"
    "WP.BASE")

(sdegeo:set-active-work-plane "wp_xz")
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Oxide"
    "WP.XZ")

(sdegeo:set-active-work-plane "wp_yz")
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "PolySi"
    "WP.YZ")

(sdegeo:define-work-plane "base_1" (position 0 0 1) (position 1 0 1)
    (position 0 1 1))
(sdegeo:set-active-work-plane "base_1")
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silicon"
    "WP.BASE_1")

(sdegeo:define-work-plane "wp_xz_1" (position 0 1 0) (position 1 1 0)
    (position 0 1 1))
(sdegeo:set-active-work-plane "wp_xz_1")
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Oxide"
```

Chapter 4: Generating Geometric Structures

Coordinate Systems and Transformations

```
"WP.XZ_1")  
  
(sdegeo:define-work-plane "wp_yz_1" (position 1 0 0) (position 1 1 0)  
    (position 1 0 1))  
(sdegeo:set-active-work-plane "wp_yz_1")  
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "PolySi"  
    "WP.YZ_1")
```

Work planes do not have to be axis aligned. The following script creates a shifted and tilted work plane:

```
(sdegeo:set-active-work-plane "base")  
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silicon"  
    "WP.BASE")  
  
(sdegeo:define-work-plane "wp_shiftandtilt" (position 0.5 0.5 0)  
    (position 0.5 1.5 0) (position 1.5 0.5 1))  
(sdegeo:set-active-work-plane "wp_shiftandtilt")  
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Nitride"  
    "WP.shiftandtilt")
```

Existing work planes can be deleted using the Work Planes dialog box or the Scheme command:

```
(sdegeo:delete-work-plane wp-name)
```

When interactive drawing tools are used, the operations are always performed in the active work plane. Even when the view is rotated, the pointer always moves in the xy plane of the active work plane.

If a planar face of an existing 3D body is selected, before choosing **Draw > Work Planes**, the data that is needed for the work plane definition will be extracted automatically from the selected face, and most fields in the Work Plane dialog box will be already filled. The origin is set to the center of the selected face. The x-axis and y-axis are selected as the principal axes of the selected face. A work plane defined in this way will be the plane in which the selected face lies.

Local Coordinate Systems

In Sentaurus Structure Editor, the coordinate systems feature offers an alternative way to define work planes. The differences are that:

- The coordinate systems method defines the new work plane with respect to the currently active work plane, instead of the global coordinate system.
- This method uses the rotation angle around the current z-axis to define the coordinate system, instead of defining the coordinate axis directly.

Chapter 4: Generating Geometric Structures

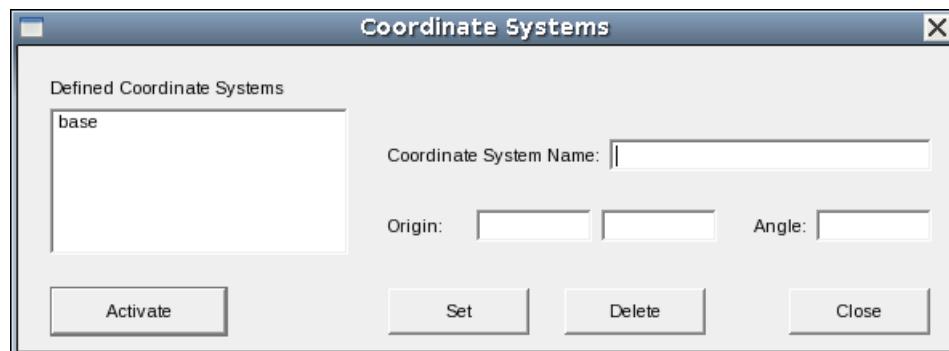
Coordinate Systems and Transformations

To define the local coordinate system:

1. Choose **Draw > Coordinate Systems**.

The Coordinate Systems dialog box opens.

2. Enter a name for the local coordinate system, the origin in the current xy plane, and a rotation angle.
3. Click **Set** to define the new work plane.



4. Select the newly created entry from the **Defined Coordinate Systems** list, and click **Activate** to activate the new work plane.
5. To delete a local coordinate system, select it from the **Defined Coordinate Systems** list, and click **Delete**.

The corresponding Scheme commands are:

```
(sdegeo:define-coord-sys "name" origin_x origin_y angle)
(sdegeo:set-active-coord-sys "name")
(sdegeo:delete-coord-sys "name")
```

Device Scaling

The Scheme command `sdegeo:scale` is used to scale all of the geometric bodies and Ref/Eval windows. The Scheme command `sdedr:write-scaled-cmd-file` is used to scale all refinement settings while writing the mesh command file.

The corresponding Scheme commands are:

```
(sdegeo:scale entity-list scale-factor-x scale-factor-y
[scale-factor-z])
(sdedr:write-scaled-cmd-file file-name scale-factor)
```

Chapter 4: Generating Geometric Structures

Coordinate Systems and Transformations

For example, if the device structure and meshing strategy were defined using nanometer as the unit of length, use Scheme commands such as the ones given here to export the device boundary and the mesh command file in micrometers:

```
(define scale-factor 1000)
(sdegeo:scale (part:entities) scale-factor scale-factor)
(sdeio:save-tdr-bnd (get-body-list) "Scaled-to-um_bnd.tdr")
(sdedr:write-scaled-cmd-file "Scaled-to-um_msh.cmd" scale-factor)
(system:command "snmesh Scale-to-um_msh")
```

These commands are not available from the GUI.

Note:

Previously, the command `sdedr:write-scaled-cmd-file scaled` Ref/Eval windows. Now, Ref/Eval windows can be scaled together with geometric regions, using the `sdegeo:scale` command. Therefore, `sdegeo:scale` must be called first, before the command `sdedr:write-scaled-cmd-file` is called.

After setting the Scheme variable `global-scaling` to a value other than `1.0` in the command-line window with:

```
(set! global-scaling 2.0)
```

scaled device structures and mesh command files can be saved by choosing **Mesh > Build Mesh**, with Scheme commands such as:

```
(set! global-scaling 1000.0)
(sde:build-mesh "Scaled-to-um_msh")
```

Entity Transformations: Scaling, Translation, Rotation, and Reflection

Several different Scheme functions and GUI operations can be used to perform basic geometry transformation operations on some or all top-level entities of the device as well as on Ref/Eval windows.

A top-level entity is at the highest level in the entity hierarchy list (that is, no other entity refers to that entity). For example, a face of a solid body (`s_b`) is not a top-level entity, since the solid body contains the face. (The `(entity:faces s_b)` command returns the particular face.) Similarly, a linear edge of a sheet body is not a top-level entity. However, if the edge was generated separately using the `edge:linear` command, for example, then it is a top-level entity.

The Transform Operations dialog box is used to perform basic transformations. It is displayed by choosing **Edit > Transform**. It has different tabs for scaling, translation, rotation, and reflection operations.

Each transform mode has a Target Entities list, which lists all objects (2D and 3D bodies) to which the transformation can be applied. The target entities can be selected from this list. The selected entities are highlighted. Selection from the GUI is also possible. In that case, the target entities must be selected from the GUI, and the corresponding entity IDs will be highlighted in the Target Entities list.

The translation, rotation, and reflection operations create new bodies if the **Transform Copy** option is selected. The newly created bodies will inherit the material property from the parent body. Region names are also inherited, but an additional string will be attached to the region names to provide unique region names.

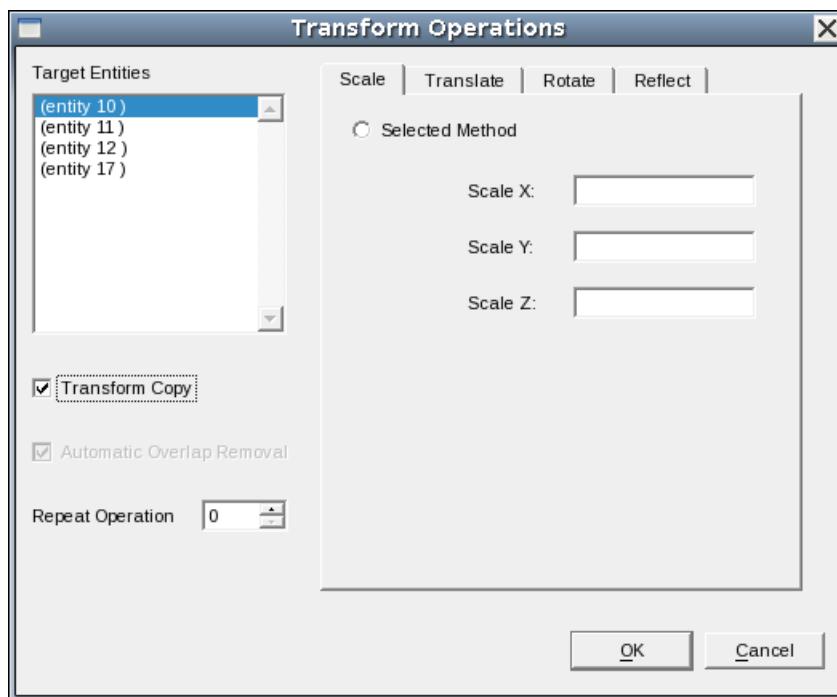
A counter is associated with each type of transformation. In each transformation, the repeat number is also counted. For example, if the name of the parent entity is `region_1` and the fifth transformation is being performed, the newly created entity will be called `region_1_t5`. If a number is entered in the **Repeat Operation** field and, for example, it is set to 3, the new regions will be called `region_1_t5_r1`, `region_1_t5_r2`, and `region_1_t5_r3`.

Local Scaling of Entities

To scale one or more entities:

1. Choose **Edit > Transform**.

The Transform Operations dialog box opens.



2. Select one or more entities in the Target Entities list.

Chapter 4: Generating Geometric Structures

Coordinate Systems and Transformations

3. Click the **Scale** tab.
4. Select **Selected Method**.
5. Enter a scaling factor for the x-, y-, and z-axis.
6. Click **OK**.

The scaling operation can be applied to top-level entities (bodies) only (these are either geometric bodies, which are part of the model geometry, or Ref/Eval windows). Nonuniform scaling is also supported. The scaling factors cannot be zero. To suppress scaling in a given direction, enter a scaling factor of 1.

The corresponding Scheme command is:

```
(sdegeo:scale-selected entity|entity-list  
    (transform:scaling x-scale y-scale z-scale))
```

For example, the following Scheme commands scale a sphere to become a flat ellipsoid:

```
(define SPHERE (sdegeo:create-sphere (position 0 0 0) 1 "Silicon"  
    "region_1"))  
(sdegeo:scale-selected SPHERE (transform:scaling 0.5 2 4))
```

Note:

Scaling is performed about the geometry center of each entity. This means entities that were touching before the scaling will either overlap or be disjoint after the scaling. Therefore, this operation is not intended to scale entire device structures. Use the operations discussed in [Device Scaling on page 155](#) for this purpose.

Difference Between sdegeo:scale and sdegeo:scale-selected

Different scaling operations are available in Sentaurus Structure Editor as follows:

- The `sdegeo:scale-selected` Scheme extension scales each body about its own geometric center (independently from other bodies). In this case, overlaps might be created, which are removed automatically, based on the selected Boolean overlap refinement rule.
- The `sdegeo:scale` Scheme extension multiplies all vertex coordinates with the specified scaling factors. The scaled model will be similar to the original model (therefore, no overlap removal is necessary).

For example:

```
(sde:clear)  
(define mb1 (sdegeo:create-rectangle (position 0 0 0)  
    (position 10 5 0) "Silicon" "mb1"))  
(define mb2 (sdegeo:create-rectangle (position 8 5 0)  
    (position 12 9 0) "PolySilicon" "mb2"))
```

Chapter 4: Generating Geometric Structures

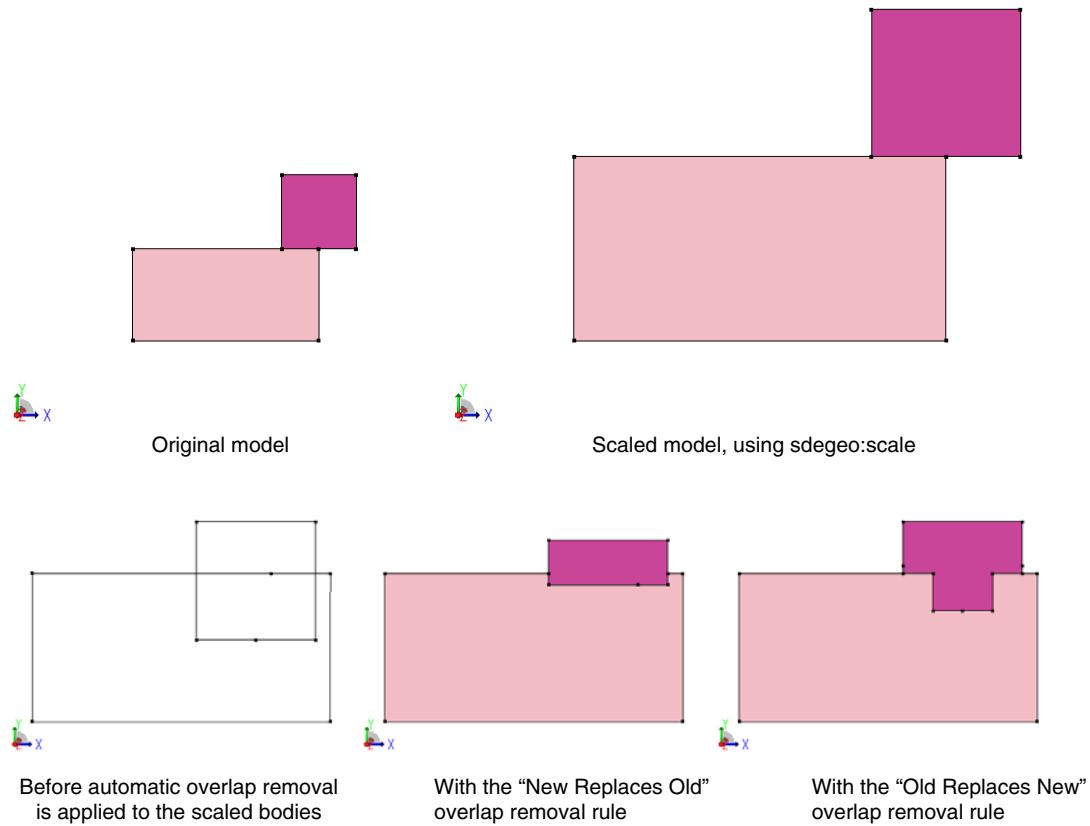
Coordinate Systems and Transformations

```
(bbox mb1)
;; (#[ position 0 0 0 ] . #[ position 10 5 0 ])
(bbox mb2)
;; (#[ position 8 5 0 ] . #[ position 12 9 0 ])
(sdegeo:scale (list mb1 mb2) 2 2 1)
(bbox mb1)
;; (#[ position 0 0 0 ] . #[ position 20 10 0 ])
(bbox mb2)
;; (#[ position 16 10 0 ] . #[ position 24 18 0 ])

(roll)
(sdegeo:set-default-boolean "ABA")
(sdegeo:scale-selected (list mb1 mb2) (transform:scaling 2 2 1))

(roll)
(sdegeo:set-default-boolean "BAB")
(sdegeo:scale-selected (list mb1 mb2) (transform:scaling 2 2 1))
;; the sdegeo:scale-selected function scales each body (independently
;; from each other) about the geometric center of the given body.
```

Figure 69 Difference between sdegeo:scale and sdegeo:scale-selected: (top) scaling using sdegeo:scale and (bottom) scaled models using sdegeo:scaled-selected



Translation

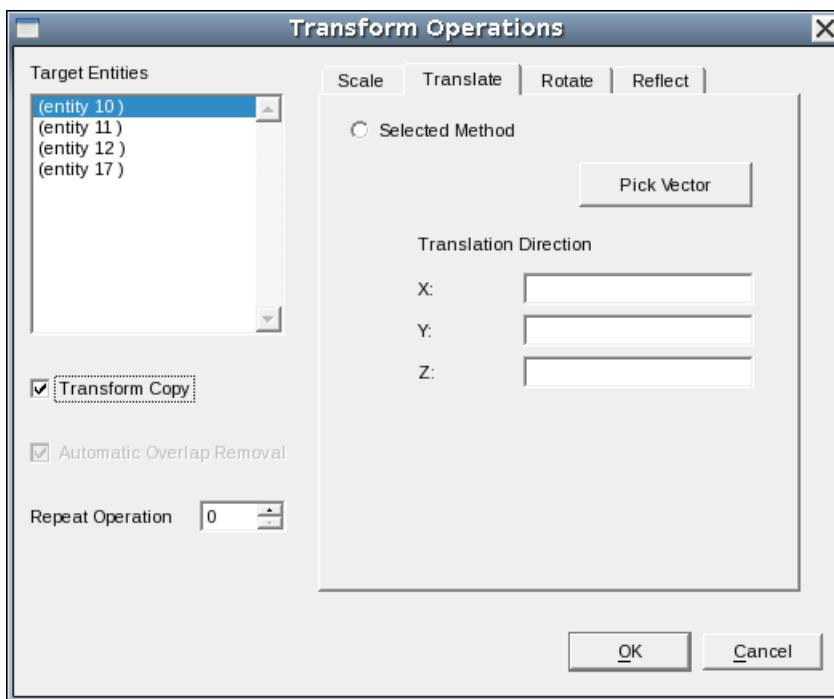
The translation operation is always performed with respect to the currently active coordinate system.

To translate one or more entities:

1. Choose **Edit > Transform**.

The Transform Operations dialog box opens.

2. Select one or more entities from the Target Entities list.
3. Click the **Translate** tab.
4. Select **Selected Method**.
5. Enter the coordinates of the translation vector, or click **Pick Vector** and click the starting point and endpoint of the translation vector in the view window.



6. Select **Transform Copy** if a copy of the original entities should remain at the initial location.
7. If a nonzero **Repeat Operation** value is entered, the translation operation is performed multiple times for the selected entities.
8. Click **OK**.

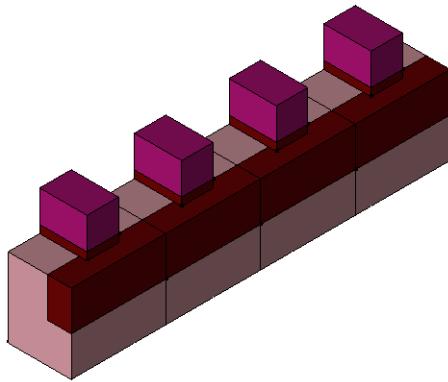
The corresponding Scheme command is:

```
(sdegeo:scale-selected entity|entity-list  
    (transform:translation gvector) copy-flag repeat-counter)
```

For example, the following Scheme commands creates a string of four simple 3D MOSFET structures:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.2 0.8 1.0)  
    "Silicon" "R.Substrate")  
(sdegeo:set-default-boolean "ABA")  
(sdegeo:create-cuboid (position 0.0 0.5 0.5) (position 1.2 0.8 1.0)  
    "Oxide" "R.STI")  
(sdegeo:create-cuboid (position 0.4 0.0 1.0) (position 0.8 0.6 1.1)  
    "Oxide" "R.Gox")  
(sdegeo:create-cuboid (position 0.4 0.0 1.1) (position 0.8 0.6 1.5)  
    "PolySilicon" "R.Poly")  
(sdegeo:translate-selected (get-body-list)  
    (transform:translation (gvector 1.2 0 0)) #t 3)
```

Figure 70 A string of 3D MOSFETs is created using the Transform Copy option and by entering a repeat number value



Rotation

Rotation operations are always performed with respect to the currently active coordinate system.

To rotate one or more entities:

1. Choose **Edit > Transform**.

The Transform Operations dialog box opens.

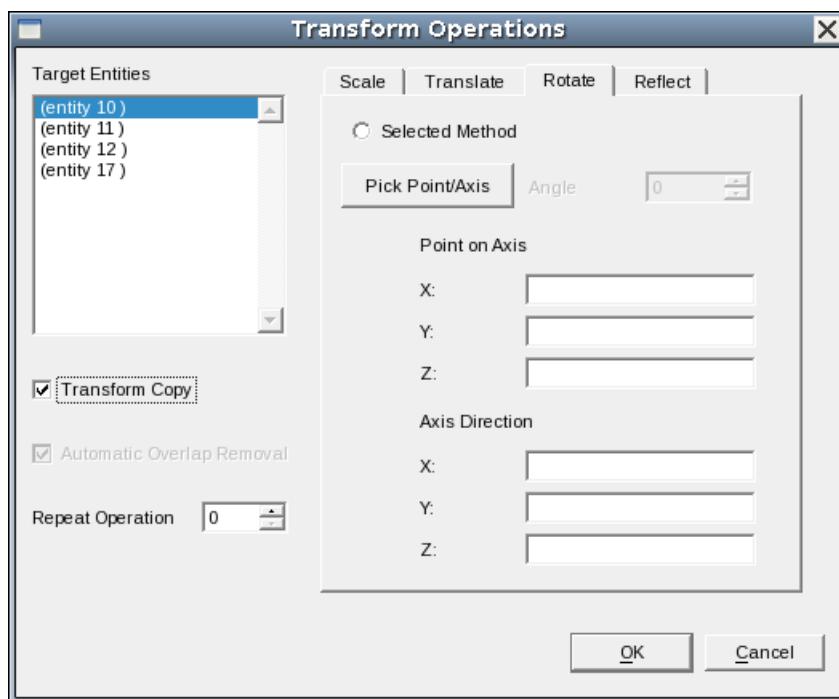
2. Select one or more entities from the Target Entities list.
3. Click the **Rotate** tab.
4. Select **Selected Method**.

Chapter 4: Generating Geometric Structures

Coordinate Systems and Transformations

5. Define the axis of rotation by entering a point on the axis and a direction vector, or click **Pick Point/Axis** and click a point in the view window.

In this case, the rotation axis will point into the current view plane. Enter also the rotation angle.



6. Select **Transform Copy** if a copy of the original entities should remain at the initial location.
7. If a nonzero **Repeat Operation** value is entered, the rotation operation is performed multiple times for the selected entities.
8. Click **OK**.

The corresponding Scheme command is:

```
(sdegeo:rotate-selected entity|entity-list  
  (transform:rotation position gvector angle) copy-flag repeat-counter)
```

For example, the following Scheme commands create a moon sickle:

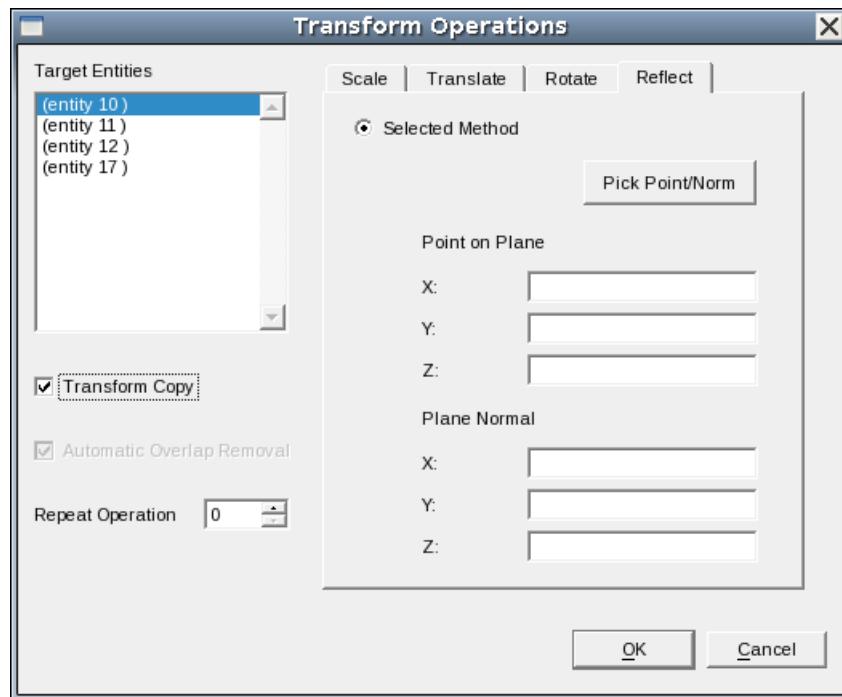
```
(define CIRCLE (sdegeo:create-circle (position 0 0 0) 1  
  "Silicon" "region_1"))  
(sdegeo:set-default-boolean "BAB")  
(sdegeo:rotate-selected CIRCLE (transform:rotation (position 1 0 0)  
  (gvector 0 0 1) -30) #t 0)  
(sdegeo:delete-region CIRCLE)
```

Reflection

To reflect one or more entities:

1. Choose **Edit > Transform**.

The Transform Operations dialog box opens.



2. Select one or more entities from the Target Entities list.
3. Click the **Reflect** tab.
4. Define the reflection plane by entering a point on the plane and a normal vector, or click **Pick Point/Norm** and click a point in the view window.

In this case, the normal vector will point to the right of the current view plane.

5. Define the axis of rotation by entering a point on the axis and a direction vector, or click **Pick Point/Axis** and click a point in the view window.

In this case, the rotation axis will point into the current view plane. Enter also the rotation angle.

6. Click **OK**.

The corresponding Scheme command is:

```
(sdegeo:mirror-selected entity|entity-list  
  (transform:reflection position gvector) copy-flag)
```

For example, the following Scheme commands reflect the half-MOSFET structure to form a full device:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.2 0.8 1.0)
    "Silicon" "R.Substrate")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-cuboid (position 0.0 0.5 0.5) (position 1.2 0.8 1.0)
    "Oxide" "R.STI")
(sdegeo:create-cuboid (position 0.4 0.0 1.0) (position 0.8 0.6 1.1)
    "Oxide" "R.Gox")
(sdegeo:create-cuboid (position 0.4 0.0 1.1) (position 0.8 0.6 1.5)
    "PolySilicon" "R.Poly")

(sdegeo:mirror-selected (get-body-list)
    (transform:reflection (position 0 0 0) (gvector 0 -1 0)) #t)
```

Scheme Functions for Transformations

Scheme functions are available to perform transformation operations.

Table 21 Scheme functions for transformations

Scheme function	Description
sdegeo:mirror-selected	Reflects the specified entities.
sdegeo:rotate-selected	Rotates the specified entities.
sdegeo:scale-selected	Scales the specified entities.
sdegeo:translate-selected	Translates the specified entities.
transform:reflection	Creates a transform to mirror an object through an axis.
transform:rotation	Creates a transform to rotate an object about an axis.
transform:scaling	Creates a scaling transform.
transform:translation	Creates a translation transform.

5

Structure Generation Using Etching and Deposition

This chapter describes how to generate process-oriented structures in Sentaurus Structure Editor using the process emulation mode.

Overview

The three-dimensional process emulation (Procem) module of Sentaurus Structure Editor offers a set of Scheme functions to emulate TCAD process steps such as deposition, etching, and implantation. The Procem Scheme extensions can be accessed either from the command-line window or by running Sentaurus Structure Editor in batch mode.

Note:

The Procem functions are not available from the graphical user interface.

A basic knowledge of the Scheme language is required to use Procem. A thorough knowledge of the Scheme programming language is not required in order to work only with single process steps, without using variables, flow controls, or conditionals. The examples in this chapter provide sufficient background for users to write simple process emulation scripts in Scheme.

In Procem, the process steps are emulated with pure geometric operations. Each process step is translated to a sequence of geometric operations including:

- Boolean operations
- Local operations on the boundary (for example, face offsettings)
- Boundary repair and healing (to perform delooping and to resolve topology changes during process emulation)

Different process emulation commands are available in Sentaurus Structure Editor. These commands usually start with the `sdepe` string.

Chapter 5: Structure Generation Using Etching and Deposition

Base Coordinate System

Procem provides functions to:

- Generate mask layouts
- Import mask layouts
- Generate substrate layers
- Perform different patterning operations
- Perform anisotropic and isotropic deposition
- Perform etching operations with or without a protect region (patterned masks)
- Perform a fill operation
- Polish a device
- Generate boundary output for meshing
- Implant doping species
- Create complex interconnect devices

Procem has a well-defined scripting language, which provides high-level functions for TDR input and output, mask layout definition and input, and geometry generation.

Procem has several high-level procedures to perform process emulation steps. These functions can be embedded in a Scheme script to generate models. The Procem functions can be freely mixed with other `sdegeo` functions to generate 3D devices.

The generated models and the flow steps are visible in the GUI. Other GUI operations and Scheme functions can be applied to the generated model between process steps. The model will always remain conformal and ready for a tessellated boundary file generation.

Typically, a process emulation flow contains several commands to generate or load masks, to create the substrate layer, and to perform patterning operations, deposition, etching, and implantation steps.

Base Coordinate System

Two different coordinate systems can be used during process emulation.

Unified Coordinate System

In the unified coordinate system (UCS), the base of the 3D model is placed in the yz plane (at $x = 0$ by default), and the open top surface of the model is in the $-x$ -direction (so a ray from $-1,0,0$ would hit the top exposed surface). When this coordinate system convention is

Chapter 5: Structure Generation Using Etching and Deposition

Mask Layout Input and Mask Generation

followed, process emulation operations (such as deposition and etching) are performed in the $-x$ -direction.

DF–ISE Coordinate System

Note:

Using the DF–ISE coordinate system is not recommended. Use the UCS instead.

The external boundary of the simulation domain is defined with respect to the global coordinate system. The external domain boundary is placed in the xy plane. All process steps are performed over the domain boundary in the positive z -direction (up direction). The default base is at $z = 0$.

If process steps are to be performed on an existing model, the model might need to be transformed in such a way that the top of the device faces the positive z -direction. This can be accomplished by a simple entity transformation using `sdegeo:rotate-selected`.

Selecting the Coordinate System

The default coordinate system for process emulation is selected in such a way that the base plane is the yz plane and the process steps are performed in the $-x$ -direction.

The Scheme command `sde:set-process-up-direction` specifies which coordinate system is used during process emulation.

If you use the Sentaurus Process–Sentaurus Structure Editor interface to perform process emulation, the Scheme script generated by Sentaurus Process will contain instructions to select the correct coordinate system (UCS) automatically.

Note:

The cross section of the model at the base location in the selected base plane (the external boundary of the simulation domain) must be rectangular.

Mask Layout Input and Mask Generation

Procem supports mask layout file input. There are several ways to define and initialize mask layouts:

- Use the `sdeicwb:generate-mask-by-layer-name` command to generate the mask from an imported `.mac` (ICWB) mask.
- Read in a GDSII file using the `sdeicwb:gds2mac` command.

Chapter 5: Structure Generation Using Etching and Deposition

Mask Layout Input and Mask Generation

- Use DF–ISE layout (.lyt) files directly.
- Define mask layouts directly in Procem using the `sdepe:generate-mask` command.

Table 22 lists the Procem functions that support mask layout input and output, and generation operations.

Table 22 Procem mask layout-related operations

Scheme function name	Description
<code>exists-empty-mask-name</code>	Checks whether the empty mask exists.
<code>exists-mask-name</code>	Checks whether the mask exists.
<code>find-mask</code>	Returns the entity ID of a mask.
<code>get-empty-mask-list</code>	Returns the defined empty masks in a list.
<code>get-mask-list</code>	Returns the defined masks in a list.
<code>mask-refevalwin-extract-2d</code>	Converts a mask to a 2D Ref/Eval window.
<code>mask-refevalwin-extract-3d</code>	Converts a mask to a 3D Ref/Eval window.
<code>sde:hide-mask</code>	Removes the mask from the view window.
<code>sde:offset-mask</code>	Offsets (biases) the specified mask.
<code>sde:show-mask</code>	Shows the mask in the view window.
<code>sde:xshow-mask</code>	Exclusively shows the mask in the view window.
<code>sdeicwb:contact</code>	Creates contacts in a mask-driven manner.
<code>sdeicwb:gds2mac</code>	Converts an existing general mask layout file (GDSII) to an ICWB .mac layout. The converted .mac layout file is loaded automatically.
<code>sdeicwb:generate-mask-by-layer-name</code>	Generates the mask from an imported .mac (ICWB) mask.
<code>sdeio:read-dfise-mask</code>	Loads a layout (.lyt) file to the modeler.
<code>sdepe:extend-masks</code>	Extends all masks about the given minimum and maximum values.
<code>sdepe:generate-empty-mask</code>	Creates an empty mask.

Chapter 5: Structure Generation Using Etching and Deposition

Mask Layout Input and Mask Generation

Table 22 *Procem mask layout-related operations (Continued)*

Scheme function name	Description
sdepe:generate-mask	Creates a mask.
sdepe:trim-masks	Trims all masks with the defined domain boundary.

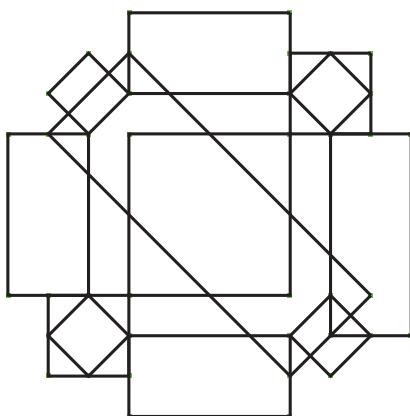
Mask Generation

Examples of mask layout generation are:

```
(sde:clear)
(sdepe:generate-mask "DEMO-MASK-1" (list (list 2 2 5 2 5 3 2 3)
    (list 5 6 9 6 7 9)))
(sdepe:generate-mask "DEMO-MASK-2" (list (list 14 7 13.73 8 13 8.73
    12 9 11 8.73 10.27 8 10 7 10.27 6 11 5.27 12 5 13 5.23 13.73 6)))
(sdepe:generate-mask "DEMO-MASK-3" (list (list 11 3 10.73 4 10 4.73 9
    5 8 4.73 7.27 4 7 3 7.27 2 8 1.27 9 1 10 1.23 10.73 2)))

(sde:clear)
(sdepe:generate-mask "MASK1" (list (list 1 1 3 3) (list 7 7 9 9)))
(sdepe:generate-mask "MASK2" (list (list 7 1 9 3 3 9 1 7)))
(sdepe:generate-mask "MASK3" (list (list 2 1 3 2 2 3 1 2)
    (list 8 1 9 2 8 3 7 2) (list 8 7 9 8 8 9 7 8)
    (list 2 7 3 8 2 9 1 8)))
(sdepe:generate-mask "MASK4" (list (list 3 3 7 7)))
(sdepe:generate-mask "MASK5" (list (list 3 0 8 2) (list 8 3 10 7)
    (list 3 8 7 10) (list 0 3 2 7)))
```

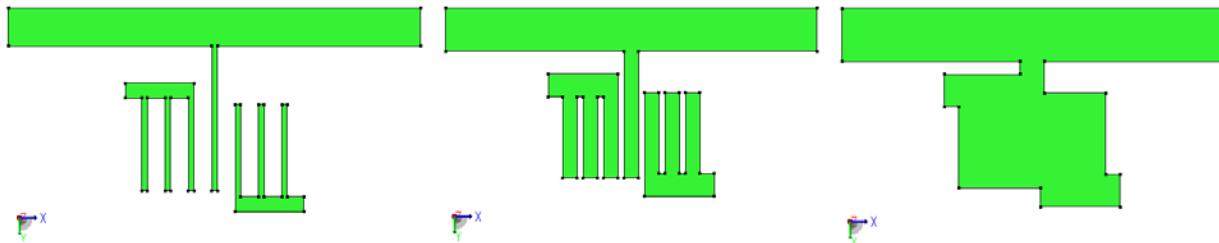
Figure 71 *Example of a mask layout*



Offsetting (Biasing) Mask

The Scheme extension `sde:offset-mask` can be used to offset (bias) a mask.

Figure 72 Mask offsetting: (left) original mask, (middle) mask after one offset step, and (right) mask after an additional offset step (using the same offset value)



TCAD Layout Reader of Sentaurus Structure Editor

The TCAD layout reader of Sentaurus Structure Editor provides a file-based interface between IC Validator WorkBench (ICVWB) and Sentaurus Structure Editor.

The ICVWB–Sentaurus Structure Editor interface can be used to perform layout-driven structure and mesh generation. This interface supports 3D structures exclusively and its key features are:

- Loading the TCAD layout (optional rescaling)
- Layout query functions
- Selecting a simulation domain
- Applying stretches
- Creating masks
- Mask-driven meshing
- Mask-driven contact assignment

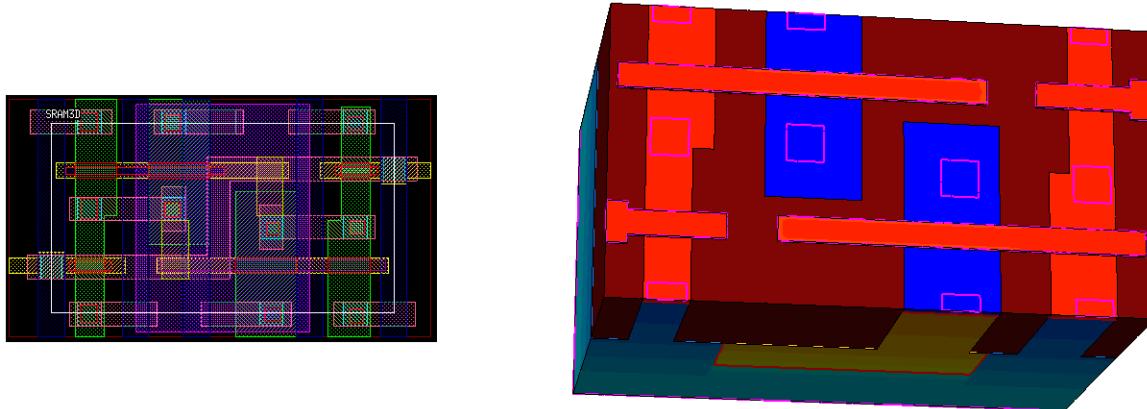
See *Sentaurus™ Process User Guide*, Chapter 12, for more information about the ICVWB–TCAD Sentaurus interface.

[Figure 73](#) shows an SRAM cell layout as well as the 3D structure generated using this layout.

Chapter 5: Structure Generation Using Etching and Deposition

TCAD Layout Reader of Sentaurus Structure Editor

Figure 73 (Left) Layout of SRAM cell and (right) corresponding 3D structure generated using process emulation mode of Sentaurus Structure Editor



Loading TCAD Layouts

Before the TCAD layout can be used in Sentaurus Structure Editor, it must be loaded.

To load a TCAD layout, use:

```
(sdeicwb:load-file layout-filename scaling-factor)
```

The scaling factor allows you to change the layout units to those used in Sentaurus Structure Editor. Use a scaling factor of 1 if no scaling is needed.

For example, to load the TCAD layout file `SRAM_lyt.mac` (which is written in nanometer) into Sentaurus Structure Editor, which here uses micrometer as the unit of length, use the command:

```
(sdeicwb:load-file "SRAM_lyt.mac" 0.001)
```

Finding Layer Names and Layer IDs

Each layer in the TCAD layout file has a unique ID of the form `<int>:<int>`, for example, `3:0`. A layer can also have an optional explicit layer name such as `NWELL`. If no explicit layer name has been set in ICVWB, 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 list the layer names defined in the TCAD layout file:

```
(sdeicwb:get-layer-names)
```

For example:

```
(define LNames (sdeicwb:get-layer-names))  
; -> icwb: LNames: (Active NWELL SRAM_mesh Acc_mesh 51:0)
```

To list the layer IDs defined in the TCAD layout file:

```
(sdeicwb:get-layer-ids)
```

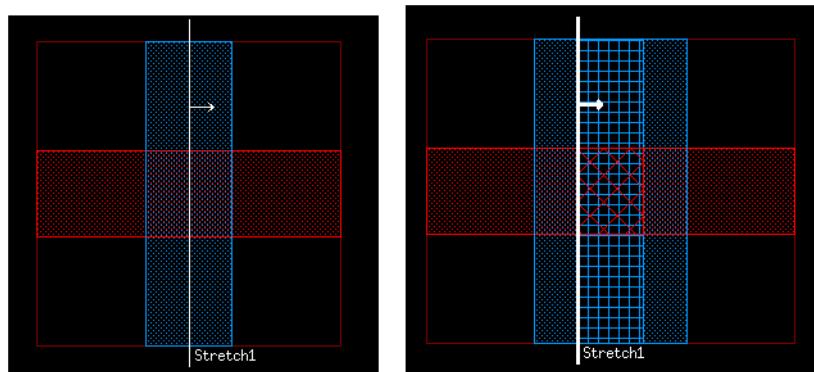
For example:

```
(define LIDs (sdeicwb:get-layer-ids))  
; -> icwb: LIDs: (1:0 2:0 43:0 44:0 51:0)
```

Applying a Stretch

The stretch utility provides a convenient way to parameterize a layout by inserting a uniformly stretched segment into the layout. This feature can be used, for example, to generate a set of transistors that have different gate lengths but are otherwise identical.

Figure 74 (Left) Snapshot of a simple ICVWB layout with a stretch utility line and (right) effective layout seen by Sentaurus Structure Editor when the layout is loaded with a positive stretch amount



A stretch line must be first defined in ICVWB. The stretch amount is set after loading the TCAD layout with the Scheme command:

```
(sdeicwb:stretch stretch-name stretch-amount)
```

For example:

```
(sdeicwb:stretch "Lgate" 0.2)
```

Selecting the Simulation Domain

The active simulation domain is selected with the Scheme command:

```
(sdeicwb:set-domain domain-name | list-of-domain-names)
```

Finding Coordinates of Bounding Box

To find the coordinates of the bounding box that is defined with regard to the global layout coordinates, use the commands:

```
process-up-direction "+z"           "-x"  
                      (sdeicwb:get-xmin) (sdeicwb:get-ymin)  
                      (sdeicwb:get-xmax) (sdeicwb:get-ymax)  
                      (sdeicwb:get-ymin) (sdeicwb:get-zmin)  
                      (sdeicwb:get-ymax) (sdeicwb:get-zmax)
```

To find the coordinates of the bounding box that automatically re-centers the simulation domain to start at the origin, use the commands:

```
(sdeicwb:get-left)  
(sdeicwb:get-right)  
(sdeicwb:get-front)  
(sdeicwb:get-back)
```

Use these commands, for example, to define the process emulation domain:

```
(define L (sdeicwb:get-left))      (define B (sdeicwb:get-back))  
(define R (sdeicwb:get-right))     (define F (sdeicwb:get-front))  
(sdepe:define-pe-domain (list L B R F))
```

Creating and Using Masks

To create a mask, use:

```
(sdeicwb:generate-mask-by-layer-name maskname layername)
```

For example:

```
(sdeicwb:generate-mask-by-layer-name "Active" "2:0")
```

The created mask can be used, for example, in the `sdepe:pattern` command:

```
(sdepe:pattern "mask" "Active" "polarity" "light" "material" "Resist"  
               "thickness" 1 "type" "iso")
```

Coordinate System Support

Sentaurus Structure Editor supports both the UCS and the DF–ISE coordinate system (see [Base Coordinate System on page 166](#)).

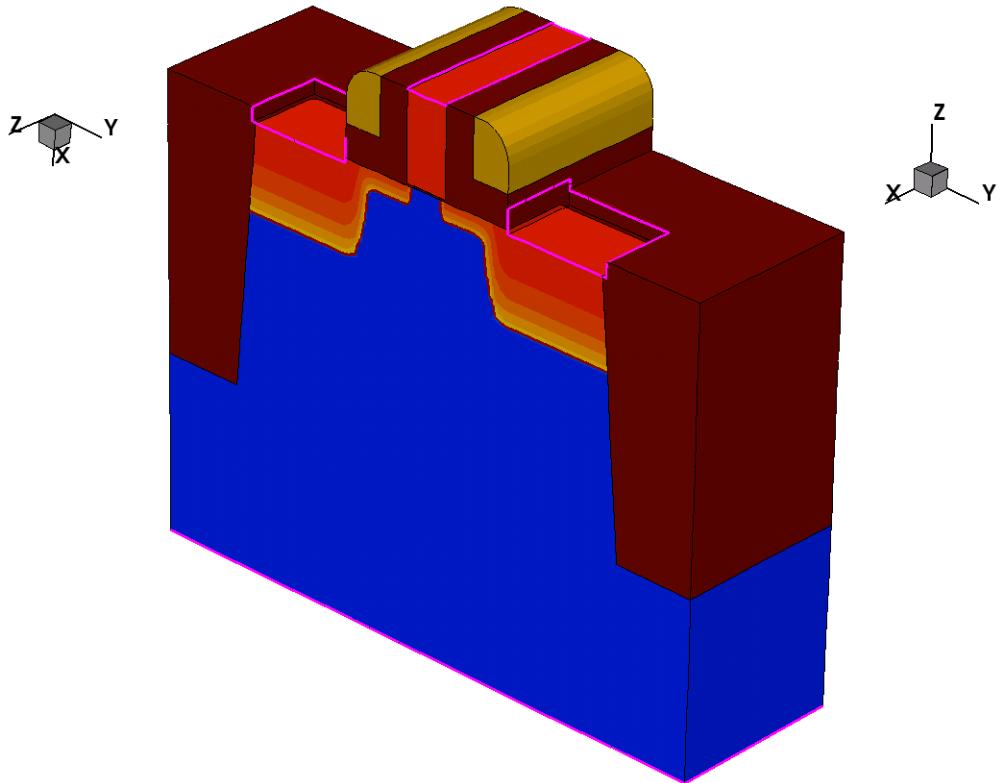
To allow for the creation of Sentaurus Structure Editor input files that support the UCS as well as the DF–ISE simulation domains, a set of utilities has been implemented.

Chapter 5: Structure Generation Using Etching and Deposition

TCAD Layout Reader of Sentaurus Structure Editor

[Figure 75](#) shows the conventions for both coordinate systems.

Figure 75 Orientation of axes for (left) UCS where the x-axis points downwards and (right) DF–ISE coordinate system where the z-axis points upwards



The concepts of *top* and *bottom* are introduced, which apply to the UCS as well as DF–ISE coordinate system:

- Global or region top coordinate:

```
(sdeicwb:get-global-top)
```

```
(sdeicwb:get-region-top RegionID)
```

Returns xmin (UCS) or zmax (DF–ISE).

- Global or region bottom coordinate:

```
(sdeicwb:get-global-bot)
```

```
(sdeicwb:get-region-bot RegionID)
```

Returns xmax (UCS) or zmin (DF–ISE).

Chapter 5: Structure Generation Using Etching and Deposition

TCAD Layout Reader of Sentaurus Structure Editor

- Translate a coordinate up:

```
(sdeicwb:up A B)
```

Returns $A - B$ (UCS) or $A + B$ (DF–ISE).

- Translate a coordinate down:

```
(sdeicwb:down A B)
```

Returns $A - B$ (UCS) or $A + B$ (DF–ISE).

Layout-Driven Contact Assignment

The `sdeicwb:contact` command creates contacts, for subsequent device simulations, that are tied to a mask or a text label in the TCAD layout file.

The `sdeicwb:contact` command supports both box-type and point-type contacts. See [sdeicwb:contact on page 708](#) for details and examples.

Layout-Driven Meshing

Layers can also be used for layout-driven meshing. The command `sdeicwb:define-refinement-from-layer` serves as an interface between the ICWB TCAD layout and the Sentaurus Structure Editor commands `sdedr:define-refeval-window`, `sdedr:define-refinement-size`, and `sdedr:define-refinement-placement` by automatically obtaining the lateral dimension of the refinement box from the specified ICWB layers, taking the vertical refinement box dimensions from the argument list. It also generates 2D and 3D refinement size settings with dimension-independent syntax.

The syntax for this command is:

```
(sdeicwb:define-refinement-from-layer "lname" lname ["rname" rname]
  ["oversize" oversize] "top" top "bot" bot "dlrmin" dlrmin
  "dlrmax" dlrmax "dbtmin" dbtmin "dbtmax" dbtmax
  ["dbfmin" dbfmin] ["dbfmax" dbfmax]
  ["material" material] ["region" region] ["use-bbox" use-bbox])
```

If no explicit refinement name (`rname`) is given, the layer name (`lname`) is used to create a unique refinement name.

For a 3D IC WorkBench simulation domain, a refinement window is created for each polygon found in the specified layers. The lateral extent of respective refinement windows is given by the bounding box of the polygon. The extent of the refinement window along the height direction (UCS: negative x-axis, DF–ISE: z-axis) is given by the values of the `top` and `bot` parameters.

Chapter 5: Structure Generation Using Etching and Deposition

TCAD Layout Reader of Sentaurus Structure Editor

The arguments `dlrmin` and `dlrmax` define the minimum and maximum refinement in the left-right dimension, respectively.

The parameters `dbtmin` and `dbtmax` define the refinement in the bottom-top dimension, and `dbfmin` and `dbfmax` define the refinement in the back-front dimension.

The name of the refinement window has the form `RPlace.<rname>_<counter>`. The name of the refinement size setting is `RSize.<rname>`.

You can use the "material" and "region" arguments to restrict the refinement to the specified bodies within the refinement box. The "material" argument is followed by either a DATEX material name or a list of materials. For example:

```
"material" "Silicon"  
"material" (list "Silicon" "PolySilicon" ...)
```

The "region" argument is followed by either a region name or a list of region names. For example:

```
"region" "region_1"  
"region" (list "region_1" "region_2" ...)
```

The argument `oversize` allows you to increase the area of refinement beyond the extent of the actual segments or polygon bounding boxes. A nonzero `oversize` value is subtracted from or added to the minimum and maximum segment or the polygon bounding box coordinates, respectively.

Note:

To use `oversize` for a non-rectangular mask while continuing to avoid unnecessary refinement, use `ICVWB` to break, for example, an L-shaped polygon into two rectangular polygons such that the union of the polygon bounding boxes coincides with the shape itself.

The argument `use-bbox` (default value `#t` or `true`) specifies that when creating 3D refinement volumes from 2D polygons, the bounding box of the polygon must be used as the basis for a cuboid volume. If set to `#f` or `false`, the polygon shape itself is used to create a 3D prism using the polygon as the top and bottom faces, with rectangular sides. This argument is used only for 2D polygons in 3D structures. When `use-bbox` is set to `#f` or `false`, `oversize` is ignored.

You can make explicit calls to `sdedr:define-refinement-function` to activate other meshing-specific options such as refinement on doping gradients or interface refinements.

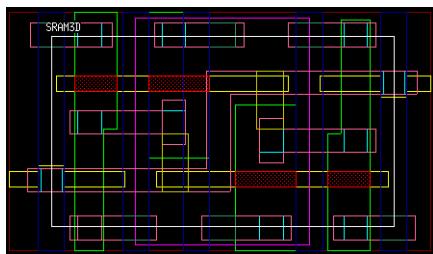
Note:

Layout-driven refinement is available only for the area under the given layer itself, *not* for the inverse of the layer. If refinement is needed in an area not covered by the layer, it is necessary to create the inverse of the layer as an auxiliary layer explicitly in IC Validator WorkBench.

The following example shows how to use an auxiliary layer `SRAM_mesh` for refinement (this layer is highlighted in red in [Figure 76](#)):

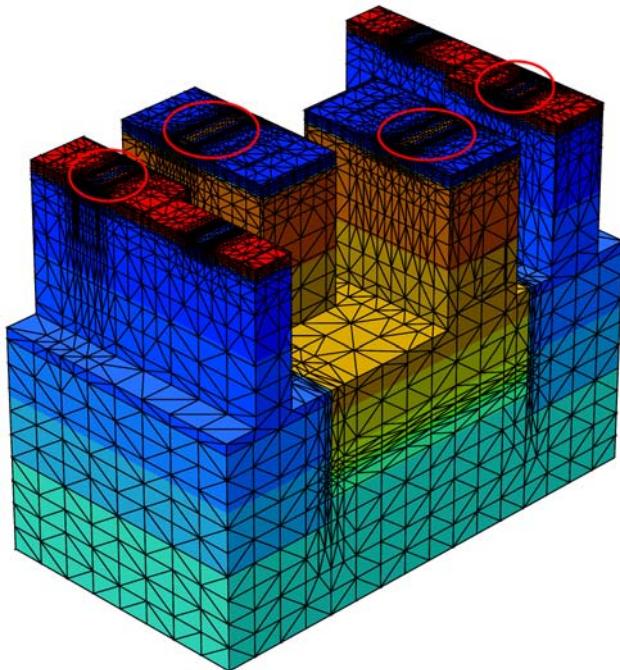
```
(sdeicwb:define-refinement-from-layer
  "lname" "SRAM_mesh" "rname" "sram_ch"
  "oversize" 0.0 "top" SiTop "bot" (sdeicwb:down SiTop 0.05)
  "dlrmax" 0.03 "dlrmin" 0.0035
  "dbtmax" 0.015 "dbtmin" 0.0035
  "dbfmax" 0.03 "dbfmin" 0.0035)
(sdedr:define-refinement-function "RSize.sram_ch" "MaxLenInt "Silicon"
  "Oxide" 0.005 2.0)
(define BBoxes (sdeicwb:get-polygon-bounding-boxes-by-layer-name LNAME))
```

Figure 76 SRAM cell layout with SRAM_mesh auxiliary layer in red



[Figure 77](#) shows a 3D mesh of an SRAM cell (only silicon regions are shown). The areas in which the layout-driven mesh refinement is applied are indicated by red circles.

Figure 77 Three-dimensional structure of SRAM cell showing position of SRAM_mesh layer



Process Emulation Commands

Scheme functions (with the prefix `sdepe`) perform process emulation operations.

Table 23 Scheme functions for process emulation operations

Scheme function	Description
<code>sdepe:add-substrate</code>	Generates the first device layer (substrate).
<code>sdepe:define-pe-domain</code>	Defines the simulation domain.
<code>sdepe:depo</code>	Performs a deposition step.
<code>sdepe:etch-material</code>	Performs an etching step.
<code>sdepe:fill-device</code>	Performs a fill operation. (Fills the device with material up to the specified height.)
<code>sdepe:icon_layer</code>	Generates a deposited, patterned, filled planar layer. (The initial structure must be planar.)
<code>sdepe:pattern</code>	Performs a pattern step. The used mask must be defined beforehand.
<code>sdepe:photo</code>	Performs a pattern step with a flat patterned top. The used mask must be defined beforehand.
<code>sdepe:polish-device</code>	Performs a polish step. (Removes the top part of a device.)
<code>sdepe:remove</code>	Removes the specified region or material.
Implantation	
<code>sdepe:doping-constant-placement</code>	Defines and implants a constant doping profile.
<code>sdepe:implant</code>	Implants an already defined analytic doping profile.

See [Appendix A on page 339](#) for detailed descriptions of these Scheme functions.

Syntax and Keywords

All process emulation commands (`sdepe`) use the same conventions for argument specification. In the Procem commands, each variable is preceded by a keyword such as

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Commands

"material", "region", "thickness", and "step". The keywords are followed by the actual value, forming a keyword–value argument pair.

The argument pairs can be specified in an arbitrary order in the argument list. For example:

```
"material" "Silicon" "thickness" H
```

is equal to:

```
"thickness" H "material" "Silicon"
```

Several argument pairs have default values. When the default value is to be used, the argument pair can be omitted. For example, the default step size is 1. The keyword "step" can be used to specify a step size; however, if it is 1, it can be omitted. A "step" 10 will perform the particular operation (deposition, etch, and so on) in 10 steps.

Some argument list items must be specified together. For example, when a rounding is required, the rounding radius must always be specified using the keyword "radius" and a numeric value.

Note:

When a keyword is used in the argument list, the corresponding value pair must be given. Otherwise, the argument list parser fails. For example, if the keyword "material" is used, a DATEX material string must follow.

Table 24 Process emulation keywords

Keyword	Type	Default	Description
"adaptive"	BOOLEAN	#f if "rounding" option is not set to #t. #t if "rounding" option is set to #t.	This keyword is used to perform the offset in more than one step (if the geometry requires this for robustness). The offset size in each step is calculated automatically, based on some geometric considerations. The offsetting continues in steps until the total deposit thickness is reached. If "adaptive" is used, in some cases, the number of steps might be large and the operation can take considerably longer than without this keyword. Note: Use "adaptive" only if the process step fails otherwise.

Function:

```
sdepe:depo  
sdepe:etch-material  
sdepe:pattern
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Commands

Table 24 Process emulation keywords (Continued)

Keyword	Type	Default	Description
"advanced-rounding"	BOOLEAN	#f	If the "rounding" option is used during deposition before the rounding operation is performed on the deposited layer, the minimal edge length is checked and the rounding radius is adjusted not to be greater than the minimal edge length. This automatic adjustment of the rounding radius provides more robustness for the rounding algorithm. In some special cases, this restriction is not needed and rounding with a larger radius than the minimal edge length is possible. This keyword allows you to bypass the automatic radius control. Function: sdepe:depo
"algorithm"	STRING	If "type" "iso", then "lopx". If "type" "aniso", then "sweep".	Different face-offsetting algorithms are available, which can be activated by the keyword "algorithm". They are "lopx", "lop-move", "sweep", and "PT". Function: sdepe:depo sdepe:etch-material sdepe:pattern
"base"	REAL	0	For process-up-direction "+z", it is the base (bottom) z-position of the substrate layer. For process-up-direction "-x", it is the base (top) x-position of the substrate layer. Function: sdepe:add-substrate
"BC"	STRING	"Extend" for "type" ("none" , "Extend" , "Periodic" , "Reflect") "aniso" ("algorithm" "sweep") and "none" for "type" "iso" ("algorithm" "lopx")	During directional operations, "BC" can be used to avoid some side effects around the domain boundary. Function: sdepe:depo sdepe:etch-material

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Commands

Table 24 Process emulation keywords (Continued)

Keyword	Type	Default	Description
"chamfer"	REAL	0	<p>Chamfer distance for chamfered deposition and etching.</p> <p>Note:</p> <p>The "chamfer" keyword is completely analogous to "radius", and the chamfer cutback is interpreted similarly to the rounding radius. The "vexity" keyword is used in the same way as for rounding.</p> <p>The keywords "chamfer" and "radius" cannot be used in the same command. The edges are either chamfered or rounded.</p> <p>Function: <code>sdepe:depo</code> <code>sdepe:etch-material</code></p>
"depth"	REAL	0	<p>Etch depth.</p> <p>Function: <code>sdepe:etch-material</code></p>
"ebl"	STRING { "all" , "top" }	"all"	<p>If the value is "top", only the top initially exposed bodies, which are made of the etch material, are etched even if the complete "depth" value is not etched away. If the value is "all", the thickness of the top layer is estimated and the etch process continues until the complete "depth" value is etched.</p> <p>Function: <code>sdepe:etch-material</code></p>
"ext-dist"	REAL	0 if "BC" is "none". For "BC" "Extend", it is calculated automatically from the ray direction and from the thickness.	<p>The applied extension distance when "BC" "Extend" is used during a directional operation.</p> <p>Function: <code>sdepe:depo</code> <code>sdepe:etch-material</code></p>

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Commands

Table 24 Process emulation keywords (Continued)

Keyword	Type	Default	Description
"height"	For process-up- direction "+z", it is the maximum z-coordinate For process-up- direction "-x", it is the minimum x-coordinate	Maximum z-coordinate	If it is not specified in <code>sdepe:fill-device</code> , then the device is filled up to the maximum z-coordinate (for process-up-direction "+z") or to the minimum x-coordinate (for process-up-direction "-x"). In <code>sdepe:polish-device</code> , if "height" is given, then the device is polished up to the specified z-coordinate value (for process-up-direction "+z") or the specified x-coordinate value (for process-up-direction "-x"). Alternatively, you can specify "thickness" in <code>sdepe:polish-device</code> , in which case, the specified value is removed from the top z-position (for process-up-direction "+z") or from the bottom x-position (for process-up-direction "-x"). Function: <code>sdepe:fill-device</code> <code>sdepe:polish-device</code>
"initial- thickness"	REAL	By default, this parameter is not used.	When selective deposition is performed, the deposition can be performed automatically in two steps using this keyword. In the first step, initial thickness * thickness is offset; in the second step, (1-initial thickness) * thickness. The initial thickness is a real value between 0 and 1. Function: <code>sdepe:depo</code>
"mask"	STRING	None	Name of the mask that is used for pattern. Function: <code>sdepe:pattern</code>

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Commands

Table 24 Process emulation keywords (Continued)

Keyword	Type	Default	Description
"material"	STRING (DATEX material)	<p>"Silicon" for sdepe:depo and sdepe:add- substrate</p> <p>" " for sdepe:etch- material</p> <p>"Resist" for sdepe:pattern</p> <p>"Gas" for sdepe:fill- device</p>	<p>Specifies the material type that will be used for the operation.</p> <p>Function: sdepe:add-substrate sdepe:depo sdepe:etch-material sdepe:fill-device sdepe:pattern sdepe:remove</p>
"overetch"	REAL	0	<p>An optional overetch distance for etching.</p> <p>Function: sdepe:etch-material</p>
"polarity"	STRING { "light", "dark" }	"light"	<p>Mask polarity for pattern.</p> <p>Function: sdepe:pattern</p>
"radius"	REAL	0	<p>Rounding radius for rounded deposition and etch.</p> <p>Function: sdepe:depo sdepe:etch-material</p>
"ray-vector"	GVECTOR	<pre>(gvector 0 0 -1) for process-up- direction "+z" (gvector 1 0 0) for process-up- direction "-x"</pre>	<p>The direction vector of the ray for directional deposition and etching.</p> <p>Function: sdepe:depo sdepe:etch-material</p>

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Commands

Table 24 Process emulation keywords (Continued)

Keyword	Type	Default	Description
"region"	STRING	Automatically generated as region_N	<p>This keyword specifies a region name for the Procem-generated new region. If it is not used, the region name is generated automatically in sequential order. The region name is generated as region_N, where N is a region counter that starts at 1 and increases sequentially.</p> <p>Function:</p> <ul style="list-style-type: none"> sdepe: add-substrate sdepe: depo sdepe: fill-device sdepe: pattern sdepe: remove
"rounding"	BOOLEAN	#f	<p>If used, the single "rounding" keyword replaces all other rounding-related keywords. The rounding radius ("radius") is computed automatically, and "vexity" is set to "convex".</p> <p>Function:</p> <ul style="list-style-type: none"> sdepe: depo sdepe: etch-material
"selective-material"	STRING (DATEX material)	None	<p>During regular deposition (when this keyword is not used), all the exposed surface is covered with the deposited layer. When this keyword is used, deposition is performed only on that part of the exposed top surface where the selective material is the exposed material.</p> <p>Function:</p> <ul style="list-style-type: none"> sdepe: depo sdepe: etch-material
"selective-taper"	BOOLEAN	#f	<p>This keyword can be used with "selective-material" when "taper-angle" is also used. In this case, the vertical faces of the newly deposited layer are tapered. (Otherwise, the newly deposited layers are first united with the complete model, and the vertical side faces of that body are tapered, and then the original bodies are subtracted from this body.)</p> <p>Function:</p> <ul style="list-style-type: none"> sdepe: depo sdepe: etch-material

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Commands

Table 24 Process emulation keywords (Continued)

Keyword	Type	Default	Description
"shadowing"	BOOLEAN	#f	If it is set to #t, the shadowing effect is taken into consideration. Function: sdepe:depo sdepe:etch-material
"steps"	INTEGER	1	Number of steps for the given operation. Function: sdepe:depo sdepe:etch-material sdepe:pattern
"taper-angle"	REAL	0	Vertical deposited and etched faces are tapered automatically if the keyword is used with an angle value (in degrees). Function: sdepe:depo sdepe:etch-material sdepe:pattern
"taper-direction"	LIST (STRING) ("x" "-x" "y" "-y" or any combination of these) for process-up- direction "z" LIST (STRING) ("y" "-y" "z" "-z" or any combination of these) for process-up- direction "-x"	None	Specifies the face normals for those axis-aligned planar faces that will be tapered. The "taper-angle" keyword must be used with this keyword. Function: sdepe:depo sdepe:etch-material
"taper-position"	"top" "bottom"	"bottom"	Specifies the base point location for tapered faces.

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Algorithms

Table 24 Process emulation keywords (Continued)

Keyword	Type	Default	Description
"thickness"	REAL	None	Thickness value (for deposition and pattern). Function: sdepe: add-substrate sdepe: depo sdepe: pattern sdepe: polish-device
"time"	REAL	None	During etching, "time" is used instead of a "depth" value. In this case, the global list called etchrates must be set as well before the etch command is used. Function: sdepe: etch-material
"type"	STRING	"iso" (deposition) "aniso" (etching)	Deposition and etching can be "iso" (isotropic) or "aniso" (anisotropic). Function: sdepe: depo sdepe: etch-material sdepe: pattern
"vexity"	STRING	"convex" (deposition) "concave" (etching)	Rounding type for rounded deposition. Possible values are "convex", "concave", or "all". The default vexity is "convex" for deposition and "concave" for etching. Function: sdepe: depo sdepe: etch-material

Process Emulation Algorithms

Different algorithms can be used during process emulation. In each sdepe function, the algorithm can be specified using the keyword `algorithm` and the name of the actual algorithm that will be used during the operation. Table 25 provides some information about how to select the algorithm, depending on the model and the type of the operation. General models are the ‘native’ CAD-type models that are generated in Sentaurus Structure Editor. The polyhedral models are typically generated in a process simulator and are loaded into Sentaurus Structure Editor for further processing.

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Algorithms

Table 25 *Process emulation algorithms and their application areas*

Algorithm	Model	Process emulation type
"lop-move"	General	Isotropic
"lopx"	General	Isotropic
"PT"	Polyhedral	Isotropic
"sweep"	General, polyhedral	Anisotropic

The lop-move Algorithm

The "lop-move" algorithm is a simple algorithm where the exposed body is translated simply by an offset vector, and Boolean operations are used to find the offset body. This algorithm is very robust and, in some cases (for example, trench formation), it generates an acceptable result.

The lopx Algorithm

The "lopx" algorithm is a string algorithm that moves the individual faces of the exposed top surface, calculates the necessary face–face intersections, and reconstructs the topologically correct offset surface.

The "lopx" algorithm is used for isotropic operations where the topology can change during offsetting. It is a complex algorithm that works best when the face connectivity (the number of faces at a given vertex) is relatively low. The "lopx" algorithm is the default algorithm for isotropic operations.

The PT Algorithm

The "PT" algorithm is a special algorithm that has been designed to be a fast and robust algorithm to handle large polyhedral models. Typically, polyhedral models have many planar faces, with possibly high face connectivity.

The "PT" algorithm is called using "algorithm" "PT". It cannot be applied to models that contain nonplanar faces. When the "PT" algorithm is used, normal rounding and edge blending cannot be used. A similar effect to rounding can be achieved with the algorithm by using the `max-chamfer-angle` argument.

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Algorithms

The "PT" algorithm cannot be combined with any other keywords that would result in a nonpolyhedral (curved) model. There is some user control that can be applied to the "PT" algorithm. [Table 26](#) lists these control parameters.

The following parameters must be used with "PT":

- `material` specifies the DATEX material that will be assigned to the deposited layer.
- `thickness` specifies the isotropic offset value that will determine the thickness of the deposited layer.

Table 26 Procem keywords for PT algorithm

Keyword	Type	Default	Description
"max-chamfer-angle"	REAL	30	If the angle between two adjacent face-normals is greater than the specified angle, chamfer faces are inserted. A nonpositive value indicates that no chamfer faces are to be inserted between the offset faces; a value in the range [30., 90.] is usually reasonable for the chamfer face insertion criteria. Function: sdepe:depo sdepe:etch-material sdepe:pattern
"min-feature-angle"	REAL	45	The PT algorithm sorts the exposed faces into face groups. Each face group is offset separately, creating offset bodies with a uniform offset. The offset bodies are then connected with chamfer bodies. The "max-chamfer-angle" parameter determines how many chamfer faces are inserted. The default value of 45 for "min-feature-angle" generates good results in most cases. If the angle between two adjacent face-normals is greater than the specified angle, the edge between the two faces will be considered a feature edge. Function: sdepe:depo sdepe:etch-material sdepe:pattern

The sweep Algorithm

The "sweep" algorithm is a very robust offsetting algorithm, which can be used for anisotropic and directional operations where the topology does not change. When the "sweep" algorithm is used, exposed faces are swept individually and the generated swept bodies are united using Boolean operations.

Chapter 5: Structure Generation Using Etching and Deposition

Restrictions on Handling Mixed Models

When it is applied for directional operations with shadowing, first the silhouette edges are computed and the faces are grouped by visibility. Only the visible faces are swept so no topology change occurs. Since there is no topology change, the "sweep" algorithm works very robustly. It is the default algorithm for anisotropic operations; therefore, anisotropic operations can be performed robustly using Procem.

Restrictions on Handling Mixed Models

If a 3D model is already present, there are some restrictions regarding the model geometry, when additional Procem steps are to be performed on the model:

- The existing device must be placed in such a way that the base of the device lies in the xy plane for process-up-direction "+z" or that the base of the device lies in the yz plane for process-up-direction "-x". All process emulation functions should be performed on the top of the device (in the +z-direction for process-up-direction "+z" or in the -x-direction for process-up-direction "-x"). If some rear-side processing is required, then the model can be transformed before any Procem functions are called. After the process emulation steps, the model can be transformed back to the original coordinate system.
- The domain boundary in the base plane must be a convex, simply connected boundary (a rectangular domain boundary is preferred).
- If the domain boundary is not yet defined, the domain boundary is calculated automatically based on the existing 3D regions on the device. The global variable `domainboundary` stores the domain boundary (wire body).
- All subsequent Procem steps are performed over the domain boundary.
- The Procem operations check the domain boundary against the bounding box of the model in each process step and, if there is a mismatch, the domain boundary will be recomputed. If the model undergoes a rigid body transformation (for example, reflection or translation) between any Procem steps, the domain boundary will be updated automatically. The `domainboundary` global Scheme variable (which holds the wire body that represents the domain boundary) also is changed to define the new domain boundary.
- The domain boundary also can be created manually using the Scheme function (`entity:delete domainboundary`).

Process Emulation Steps

This section discusses the process emulation steps.

Defining Domain

If a device is created from the beginning, the domain boundary must be specified first, using the `sdepe:define-pe-domain` Scheme extension. All subsequent Procem steps are performed over this domain boundary. If Procem steps are to be performed on an existing 3D structure, the domain boundary need not be defined. The domain boundary is extracted automatically from the device geometry. The global variable `domainboundary` stores the domain boundary. If the simulation domain changes between process steps (for example, the device is reflected or mirrored), the original domain boundary must be deleted using `(entity:delete domainboundary)`. A subsequent process step recalculates the correct domain boundary.

Note:

Process emulation commands operate on the entire device and cannot be restricted to operate only on a certain part (defined by a rectangular domain). However, if you want to modify the process emulation domain, use the `sdegeo:chop-domain` command to remove the unwanted part of the model. After the unwanted part is removed, the previously computed domain boundary must be deleted using the `(entity:delete domainboundary)` command.

Generating Substrate

The `sdepe:add-substrate` Scheme extension generates an initial substrate layer.

Patterning

The `sdepe:pattern` Scheme extension performs a pattern process-emulation step.

Patterning Keywords and Keyword Combinations

[Table 27](#) lists all the keywords that the `sdepe:pattern` command accepts in addition to the keywords that are defined in [Table 28](#) for deposition.

All the keywords that can be used for deposition can also be used during a patterning step, so those keywords are not explained here; they are explained in [Deposition on page 193](#).

Table 27 Patterning keywords and keyword combinations

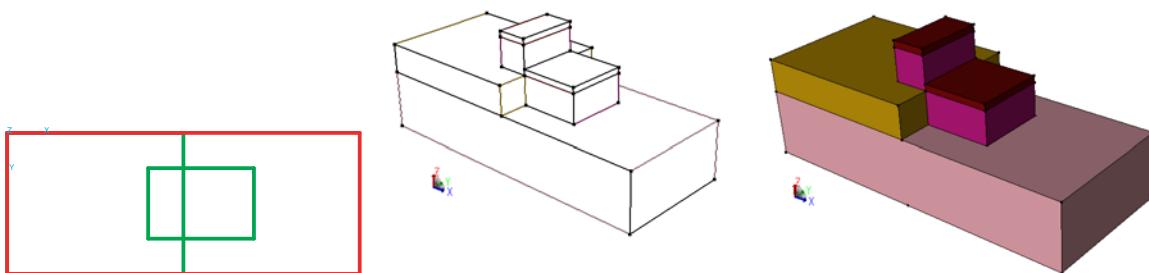
Type	Keyword	Functionality
All keywords listed in Table 28 on page 194 plus:		
Basic parameter	"mask"	The keyword "mask" identifies the mask that is used for patterning. The first step during a pattern operation is a deposition step (without using the specified mask). In the second step, the mask (or the complementary mask, depending on the specified polarity) is extruded and is combined with the deposited layer (using some Boolean operations) to form the patterned layer.
Basic control	"polarity"	This keyword can be "light" (default value) or "dark". It is used to specify whether the used mask or the complementary mask (with respect to the domain boundary) should be used during the pattern operation.

Anisotropic Patterning

[Figure 78](#) was generated using Procem and the following script:

```
(sde:clear)
(sde:set-process-up-direction "+z")
;# Mask definition #
(sdepe:generate-mask "MASK1" (list (list 0 0 5 4)))
(sdepe:generate-mask "MASK2" (list (list 4 1 7 3)))
;# Process Flow #
(sdepe:define-pe-domain 0 0 10 4)
(sdepe:add-substrate "material" "Silicon" "thickness" 2)
(sdepe:pattern "mask" "MASK1" "polarity" "light" "material" "Nitride"
    "thickness" 1 "type" "aniso")
(sdepe:pattern "mask" "MASK2" "polarity" "light" "material"
    "PolySilicon" "thickness" 1 "type" "aniso")
(sdepe:pattern "mask" "MASK2" "polarity" "light" "material" "Oxide"
    "thickness" 0.2 "type" "aniso")
```

[Figure 78 Examples of anisotropic patterning](#)



Isotropic Patterning

Figure 79 was generated with Procem using the script for the anisotropic patterning operation, except that, in this case, instead of "type" "aniso", the argument "type" "iso" was used.

Figure 79 Examples of isotropic pattern operations

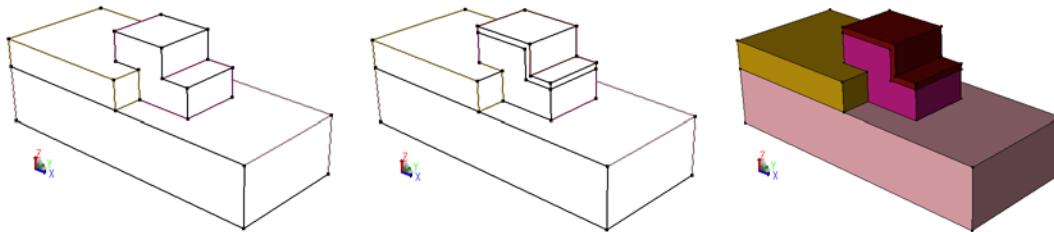


Photo Operation

The `sdepe:photo` Scheme extension is used to perform a flat pattern process-emulation step. The operation uses an existing mask to perform a pattern operation. The difference between a pattern step (`sdepe:pattern`) and a photo step (`sdepe:photo`) is that, in the case of a photo step, the top of the device will be flat. This provides more robust behavior if the `sdepe:photo` command is used.

Photo Keywords and Keyword Combinations

See the description of [sdepe:photo on page 767](#) for all available keywords and arguments.

Example

Figure 80 was generated with Procem using the following script:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 4) "Silicon"
  "x1")
(sdegeo:create-cuboid (position 0 3 4) (position 4 7 6)
  "PolySilicon" "x2")
(define f1 (car (find-face-id (position 1 1 4))))
(define f2 (car (find-face-id (position 2 5 6))))
(sdegeo:taper-faces (list f1) (position 5 5 4) (gvector 1 0 0) 10)
(sdegeo:taper-faces (list f2) (position 2 5 6) (gvector 1 0 0) -10)
(sdepe:generate-mask "M1" (list (list 0 0 10 5)))

(sdepe:photo "mask" "M1" "material" "Copper" "polarity" "light"
  "height" 7)
```

Figure 80 (right) was created using:

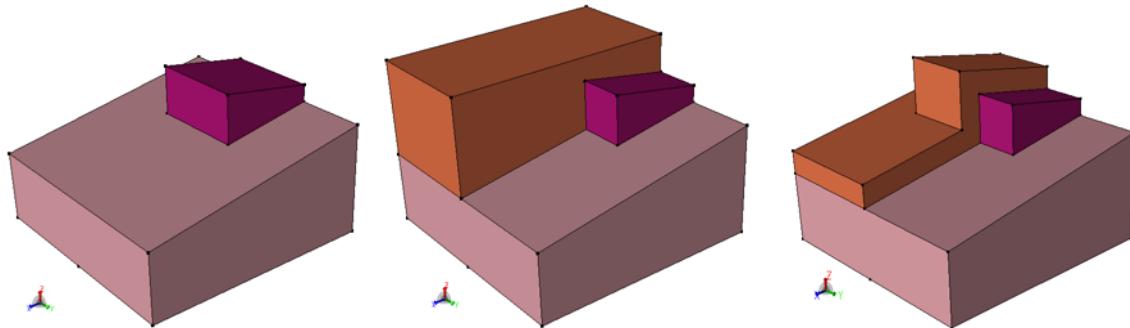
```
(sdepe:pattern "mask" "M1" "material" "Copper" "polarity" "light"  
"thickness" 1)
```

instead of:

```
(sdepe:photo "mask" "M1" "material" "Copper" "polarity" "light"  
"height" 7)
```

If "height" is used, then the flat top of the created body is placed at the "height" position. Instead of "height", you can use the "thickness" parameter. In this case, the top z-coordinate of the created body is placed at $z_{max} + thickness$, where z_{max} is the top z-coordinate of the initial device.

Figure 80 Example of a photo (flat pattern) operation: (left) initial structure with slanted top faces, (middle) sdepe:photo-created layer with flat top surface, and (right) sdepe:pattern-created layer, which follows the underlying geometry



Deposition

The `sdepe:depo` Scheme extension is used to perform a deposition process-emulation step.

Deposition Keywords and Keyword Combinations

Table 28 lists all of the keywords that the `sdepe:depo` command accepts. It is organized into functional groups:

- The *basic parameter group* contains the minimal set of keywords that can be used to create a deposited layer (only the deposit material and thickness belong here).
- The *basic control group* contains all the keywords that can be used in addition to the basic parameters to control the deposition type.

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

- The *advanced control groups* list all the keywords that can be used to achieve a certain goal (such as rounding and tapering). These groups can be used alone or in combination with each other.
- The last group (*technical parameters*) can be used to improve the robustness of the `sdepe : depo` operation. In some cases (especially when global topology changes occur) with some step control, the operation can behave more robustly.

Table 28 Deposition keywords and keyword combinations

Type	Keyword	Functionality
Basic parameters	"thickness" "material"	Default Manhattan-type deposition with isotropic offsetting. The "thickness" keyword controls the offset distance and "material" controls the assigned material for the deposited layer.
Basic control	"region"	Region name control. The specified region name is assigned to the created deposited layer. If "region" is not used, the region name is assigned automatically to the deposited layer.
	"type"	Controls the isotropic or anisotropic offset type. The keyword-value pair "type" "iso" creates a uniformly offset deposit layer and "type" "aniso" creates an anisotropic offset layer. (The actual thickness at each point (in the vertical direction) depends on the face normal. Vertical faces are not offset.)
Advanced controls (these groups can be combined)		
Rounding	"rounding" "radius" "vexity" "advanced-rounding"	Rounding and control parameters for rounding. Either "rounding" or "radius" is used. If "rounding" is used, the rounding radius is determined automatically. If "radius" is used, the specified radius is used during the rounding phase. If "radius" is greater than the minimum edge length of the exposed surface, the radius is readjusted. (This automatic feature is switched off using the "advanced-rounding" keyword. In this case, the rounding might not be as robust as with the readjusted rounding radius but, for simpler geometries (for example, extruded models), this option can be used.) The "vexity" keyword controls which edges are rounded. The possibilities are "convex", "concave", or "all".

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Table 28 Deposition keywords and keyword combinations (Continued)

Type	Keyword	Functionality
Tapering	"taper-angle" "taper-direction"	Tapering and taper face selection control. The "taper-angle" keyword controls the angle that is used to taper the vertical planar side faces of the deposited layer. In addition, "taper-direction" can be specified for additional filtering of the vertical planar side faces, by selecting only the x- or y-axis-aligned faces or both.
Selective deposition	"initial-thickness" "selective-material" "selective-taper"	Selective deposition and control parameters for selective deposition. If "selective-material" is used, the deposited layer is created only on those parts of the exposed top surface that were originally assigned the defined selective material. Further control can be applied using "selective-material" and "selective-taper". See Selective Deposition on page 200 for a detailed explanation of these keywords.

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Table 28 Deposition keywords and keyword combinations (Continued)

Type	Keyword	Functionality
Directional deposition with or without shadowing	"ray-vector" "BC" "ext-dist" "shadowing"	<p>Directional deposition with control parameters for boundary conditions and shadowing.</p> <p>By default, deposition is performed on the top (+z-direction for process-up-direction "+z" or -x-direction for process-up-direction "-x") exposed faces assuming a vertical direction. This can be changed by using a ray vector that differs from the default one, which is <code>gvector 0 0 -1</code> for process-up-direction "+z" and <code>gvector 1 0 0</code> for process-up-direction "-x".</p> <p>The "ray-vector" keyword specifies the ray direction. If directional deposition is used, the domain boundary can be extended automatically using "BC", which can be set to "none", "Extend", "Reflect", or "Periodic".</p> <p>If the "Reflect" option is used, the model is reflected automatically to those sides of the domain boundary that are in the ray direction. When the "Periodic" option is used, a copy of the model is added along those sides of the domain boundary that are in the ray direction.</p> <p>The "Extend" option (default) is used to extend the model. If the extension distance is not specified, the size of the extension (based on the ray direction and the offset distance) is calculated automatically. The extension distance can be specified using "ext-dist".</p> <p>The "shadowing" keyword activates shadowing. In this case, deposition will not occur on those parts of the device that are shadowed in the ray direction. After deposition, the domain boundary is trimmed back to the original domain boundary.</p>

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Table 28 Deposition keywords and keyword combinations (Continued)

Type	Keyword	Functionality
Technical parameters	"adaptive"	<p>These parameters control some of the technical aspects of the operation.</p> <p>The "adaptive" option is used to perform the offset in more than one step (if the geometry requires that for robustness). The offset size in each step is calculated automatically, based on some geometric considerations. The offsetting continues in steps until the total deposit thickness is reached. If "adaptive" is used, in some cases, the number of steps might be too large and the operation takes considerably longer than without this option.</p> <p>Note: Use this option only if the deposition step fails without this option.</p>
	"algorithm"	<p>Controls the selection of the offset algorithm.</p> <p>If "type" "iso" is used, the default algorithm is set to "lopx". In some cases, this algorithm might fail, but the simpler "lop-move" algorithm can work and result in an acceptable deposited layer.</p> <p>If "type" "aniso" is used, the default algorithm is set to "sweep". This is the only available algorithm for anisotropic deposition and should be sufficiently robust to handle even large (tessellated) models. For large tessellated (polyhedral) models, in the case of isotropic deposition, the "PT" algorithm is recommended.</p> <p>Note: None of the other advanced option arguments can be used if the "PT" algorithm is used.</p>
	"steps"	<p>This optional parameter controls the number of steps used during deposition. The default is 1. If a different step number is used, then the total deposit thickness is divided by the number of steps, and the operation is performed in steps.</p>

Anisotropic Deposition

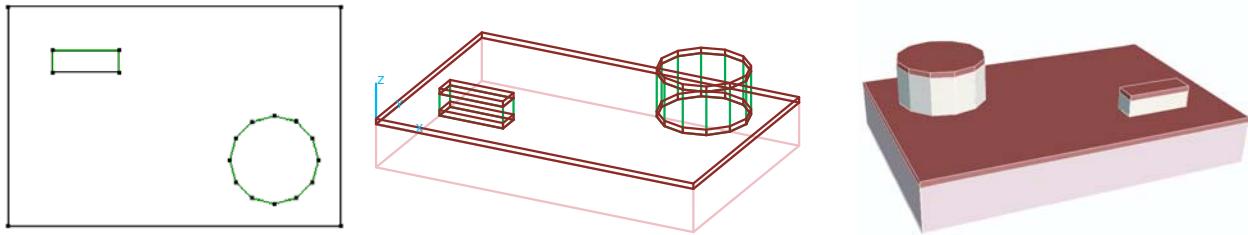
```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdepe:generate-mask "MASK1" (list (list 2 2 5 3)))
(sdepe:generate-mask "MASK2" (list (list 14 7 13.73 8 13 8.73 12 9
11 8.73 10.27 8 10 7 10.27 6 11 5.27 12 5 13 5.23 13.73 6)))
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

```
(sdepe:define-pe-domain (list 0 0 15 10))
(sdepe:add-substrate "material" "Silicon" "thickness" 2)
(sdepe:pattern "mask" "MASK1" "polarity" "light" "material" "Resist"
  "thickness" 1)
(sdepe:pattern "mask" "MASK2" "polarity" "light" "material" "Resist"
  "thickness" 2)
(sdepe:depo "material" "Oxide" "thickness" 0.25 "type" "aniso")
```

Figure 81 Anisotropic deposition, mask layout, and deposited layers



Isotropic Deposition

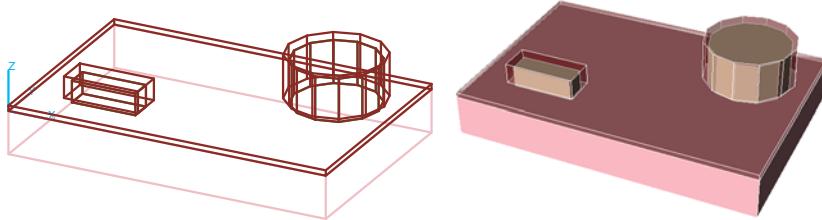
This section presents various examples of isotropic deposition.

Example: Isotropic Deposition With blend-vexity = "none"

In this example, this is the same code as used to illustrate anisotropic deposition. The only difference is that the anisotropic step is replaced with:

```
(sdepe:depo "material" "Oxide" "thickness" 0.25 "type" "iso")
```

Figure 82 Manhattan-type isotropic deposition



The default "type" is "iso" for deposition. Therefore, if the "type" is not specified in the argument list, an isotropic operation is performed. The isotropic deposition step can also be performed by using the optional arguments and applying different types of rounding.

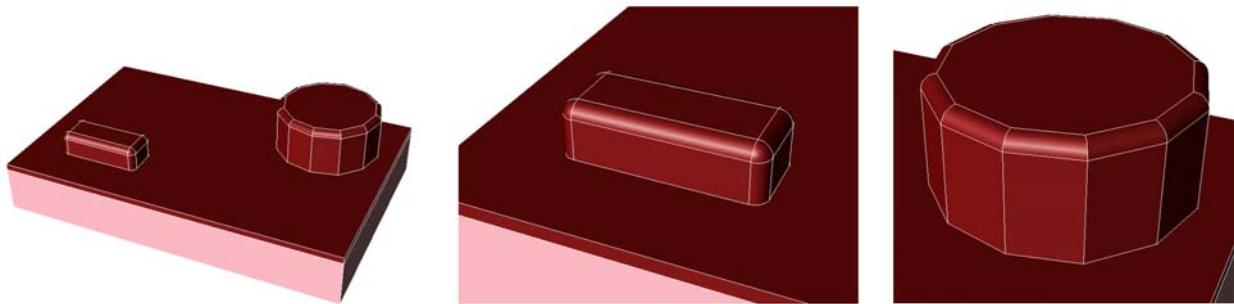
Example: Isotropic Deposition With blend-vexity = "convex"

```
(sdepe:depo "material" "Oxide" "thickness" 0.25 "vexity" "convex"
  "radius" 0.25)
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

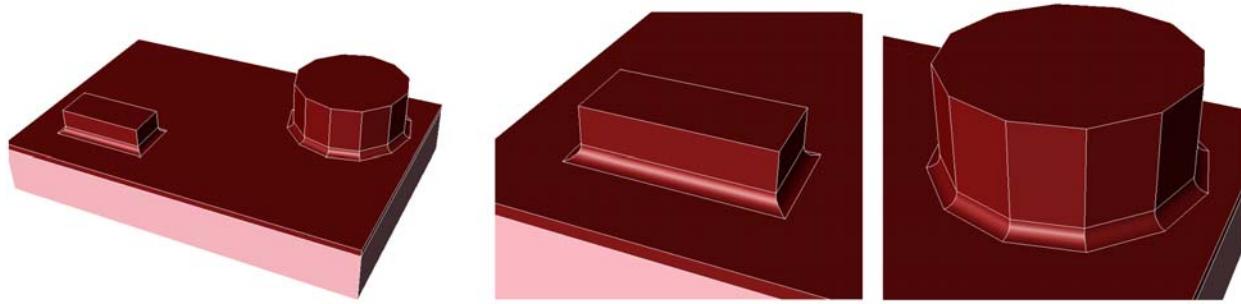
Figure 83 Isotropic deposition with "vexity" "convex"



Example: Isotropic Deposition With blend-vexity = "concave"

```
(sdepe:depo "material" "Oxide" "thickness" 0.25 "vexity" "concave"  
"radius" 0.25)
```

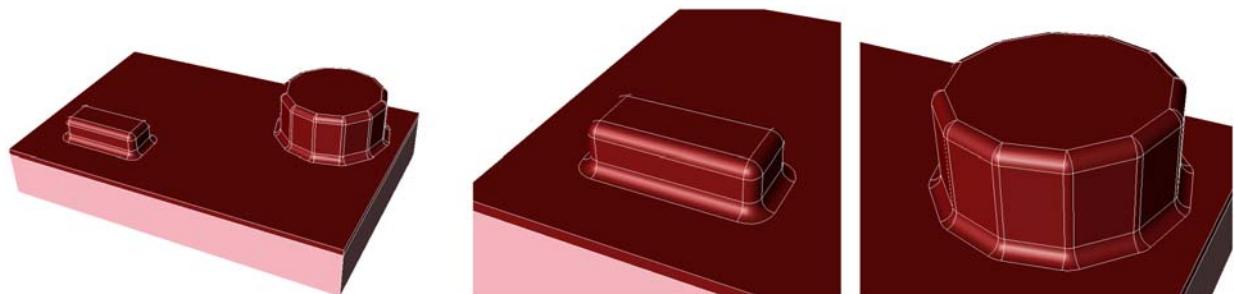
Figure 84 Isotropic deposition with "vexity" "concave"



Example: Isotropic Deposition With blend-vexity = "all"

```
(sdepe:depo "material" "Oxide" "thickness" 0.25 "vexity" "all"  
"radius" 0.25)
```

Figure 85 Isotropic deposition with "vexity" "all"



Selective Deposition

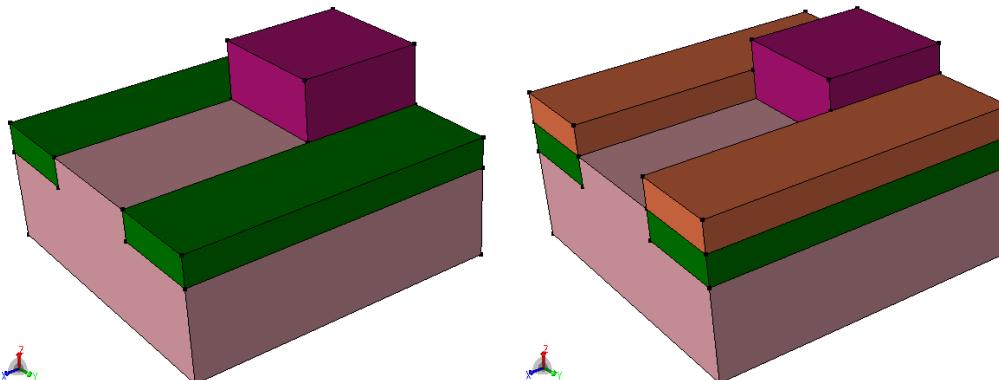
By default, deposition is performed on the entire exposed top surface, unless the argument "selective-material" is used, in which case, the deposition is performed only on those selected faces that originally belonged to the specified material.

[Figure 86](#) was generated with Procem using the following script:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 4) "Silicon"
    "x1")
(sdegeo:create-cuboid (position 0 3 4) (position 4 7 6)
    "PolySilicon" "x2")
(sdegeo:create-cuboid (position 0 0 3) (position 10 3 4) "Silicide"
    "x3")
(sdegeo:create-cuboid (position 0 7 3) (position 10 10 4) "Silicide"
    "x4")

(sdepe:depo "material" "Copper" "light" "thickness" 1
    "selective-material" "Silicide")
```

Figure 86 Selective deposition using the "selective-material" argument: (left) before deposition and (right) after deposition



An additional parameter "initial-thickness" can be used to control how much anisotropy or isotropy is applied during selective deposition. The "initial-thickness" parameter is a real number between 0 and 1.

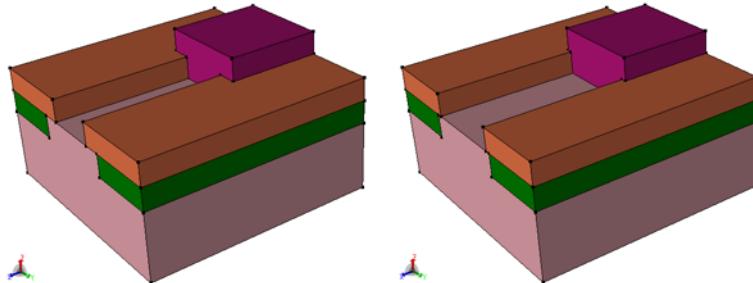
If "initial-thickness" is used, during the first phase, $\text{initial-thickness} * \text{thickness}$ is deposited on the selective material by inserting vent faces between the selective material and the other materials. In the second phase, the selective material is offset by $(1 - \text{initial-thickness}) * \text{thickness}$.

When the deposition type is isotropic, the vent faces that were inserted during the first phase of the selective deposition are also offset. (Therefore, the deposited body will overstretch the selective material.) [Figure 87](#) shows the effect of different values of "initial-thickness".

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

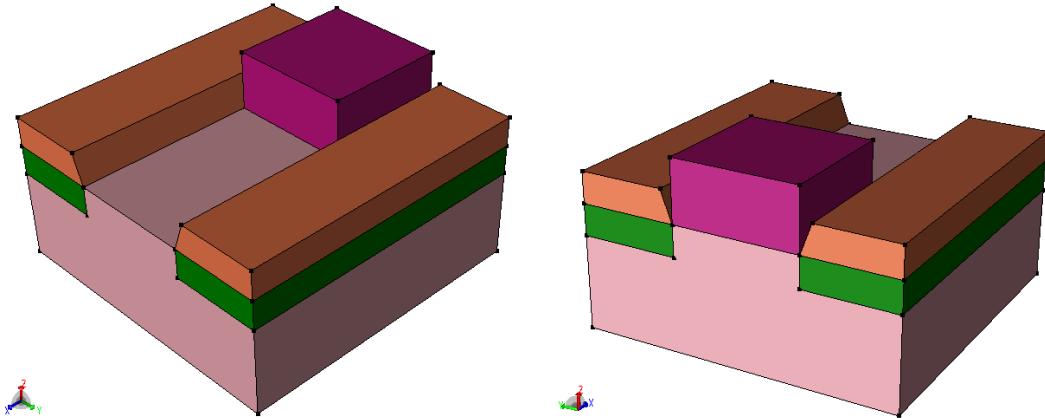
Figure 87 Selective deposition with "initial-thickness" values of (left) 0.2 and (right) 0.8



In addition, tapering can be combined with selective deposition. If the "selective-taper" argument is used with the `#t` value, the selectively deposited body is separated, and the vertical walls of the offset body are tapered. If this parameter is not used, tapering is performed globally on the united body, where the selectively deposited body is combined first with all other bodies, and the vertical walls of the united body are tapered.

Figure 88 shows the deposited layer with tapered vertical walls in the case where the "selective-taper" `#t` argument pair is used.

Figure 88 Selective deposition using the "selective-taper" `#t` argument pair



Directional Deposition

By default, Procem operations are performed in the global z-direction for process-up-direction "`+z`" or in the global $-x$ -direction for process-up-direction "`-x`", assuming a vertical ray direction. The device is placed in the global xy plane for process-up-direction "`+z`" or in the global yz plane for process-up-direction "`-x`", and the exposed top surface of the device points in the $+z$ -direction for process-up-direction "`+z`" or in the $-x$ -direction for process-up-direction "`-x`".

If you take directional effects into consideration, then you can define a ray vector other than $(0\ 0\ -1)$ for process-up-direction "`+z`" or $(1\ 0\ 0)$ for process-up-direction "`-x`". The

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

ray vector can be specified using the "ray-vector" argument and the ray direction (GVECTOR). For example: "ray-vector" (gvector 1 0 -1)

Note:

You must define the "ray-vector" in such a way that the ray hits the top surface. Therefore, for example, (gvector 1 0 0) is not permitted for process-up-direction "+z".

Directional effects are associated with anisotropy; therefore, use "type" "aniso" for directional operations.

For anisotropic operations, the suggested algorithm is the "sweep" algorithm, which can be called, by using the keywords "algorithm" "sweep".

As a result of the directional operation, an unwanted side-rim artifact is usually generated around the domain boundary. This side effect can be eliminated by applying an additional boundary condition during the Procem operation. The keyword "BC" is used to specify an additional boundary condition. The keyword "BC" has the following options:

- "none", in which case, no boundary condition is applied.
- "Extend" extends the device around the domain boundary before the directional operation and trims the device to the original domain afterwards.

The extension length is calculated automatically based on the ray direction, or it can be specified explicitly by using the "ext-dist" keyword and an extension length.

- "Periodic" applies a periodic boundary condition about the device (the actual ray direction is taken into consideration), so the periodicity is usually not applied to all sides. After the operation, the device is trimmed to the original domain.
- "Reflect" applies a reflective boundary condition about the device (the actual ray direction is taken into consideration), so the periodicity is usually not applied to all sides. After the operation, the device is trimmed to the original domain.

The following example illustrates directional deposition:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:set-default-boolean "AB")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 4) "Silicon"
    "x1")
(sdegeo:create-cuboid (position 3 3 4) (position 7 7 8) "Silicon"
    "x1")
```

Figure 89 (left) was created using:

```
(sdepe:depo "thickness" 1 "material" "Copper" "ray-vector"
    (gvector 1 1 -1) "algorithm" "sweep")
```

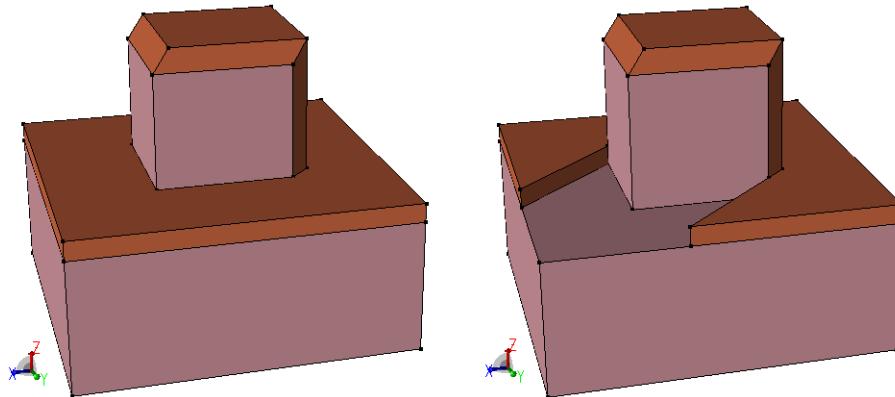
Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Figure 89 (right) was created using:

```
(sdepe:depo "thickness" 1 "material" "Copper" "shadowing" #t  
"ray-vector" (gvector 1 1 -1) "algorithm" "sweep")
```

Figure 89 *Directional deposition with different options: (left) using the "ray-vector" parameter only without the "shadowing" option and (right) the same operation with "shadowing" #t*



Shadowing

In addition, the shadowing effect can be taken into consideration by adding "shadowing" #t to the Procem argument list. When the shadowing option is used, the following additional steps are performed during process emulation:

- Based on the ray direction, the silhouette edges of the device are imprinted to the top exposed surface of the device.
- Raytracing is applied to determine face visibility. (After the silhouette edges are imprinted, exposed faces can be classified as visible or nonvisible.)
- Based on the visibility result, the face displacement map, which was calculated based on the "type" keyword, is modified. A zero displacement is assigned to all nonvisible faces.

The "shadowing" option can be used to compute face visibility and to omit the nonvisible faces from being offset. If it is needed, faces will be split along the contour lines, which will be imprinted to the faces during the visibility calculation. The following example illustrates the effect of the "shadowing" option:

```
(sde:clear)  
(sde:set-process-up-direction "+z")  
(sdegeo:set-default-boolean "AB")  
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 4) "Silicon"  
"x1")  
(sdegeo:create-cuboid (position 0 0 4) (position 10 5 8) "Silicon"  
"x1"))  
(sdegeo:set-default-boolean "ABA")
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

```
(define mb (sdegeo:create-cylinder (position 0 5 4) (position 10 5 4)
 2 "Silicon" "x1"))
(entity:delete mb)
```

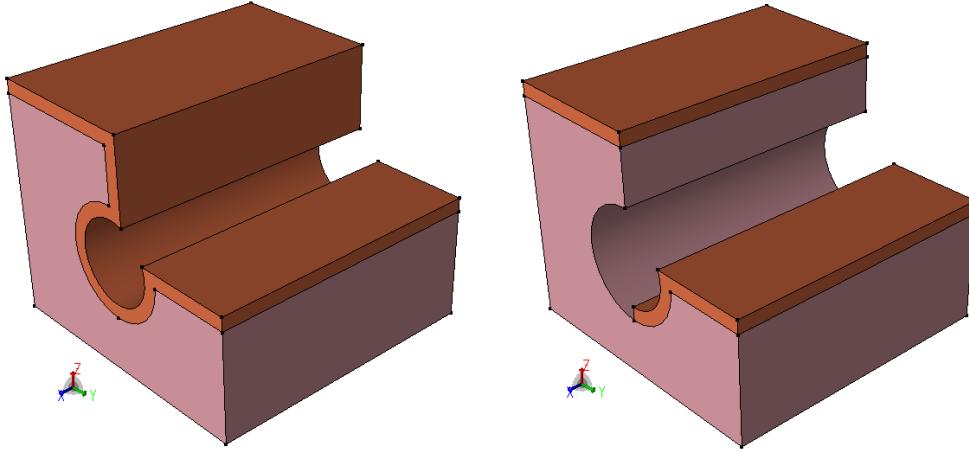
Figure 90 (left) was created using:

```
(sdepe:depo "material" "Copper" "thickness" 0.5)
```

Figure 90 (right) was created using:

```
(sdepe:depo "material" "Copper" "thickness" 0.5 "shadowing" #t)
```

Figure 90 Effect of "shadowing" option during deposition: (left) without shadowing and (right) with shadowing



Rounding and Advanced Rounding

If rounding is used, the rounding radius must be specified. To add more robustness to the rounding operation, the rounding radius is corrected automatically when a small edge is detected on the top exposed surface. If "vexity" is "convex" (default) or "concave", the rounding radius is set to $0.9 * \text{minimal edge length}$ if this value is smaller than the actual rounding radius. If "vexity" is "all", then the multiplication factor is 0.45 not 0.9.

For some cases (for example, extruded models), the rounding can be performed with a larger radius. In these cases, the automatic reassignment of the rounding radius can be overruled by using the "advanced-rounding" parameter. The following example shows the effect of the parameter:

```
(sde:clear)
(sde:set-process-up-direction "+z")
  (sdegeo:set-default-boolean "AB")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 4) "Silicon"
  "xx")
(sdegeo:create-cuboid (position 0 3 4) (position 10 7 6) "Silicon"
  "xx")
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Figure 91 (left) was created using:

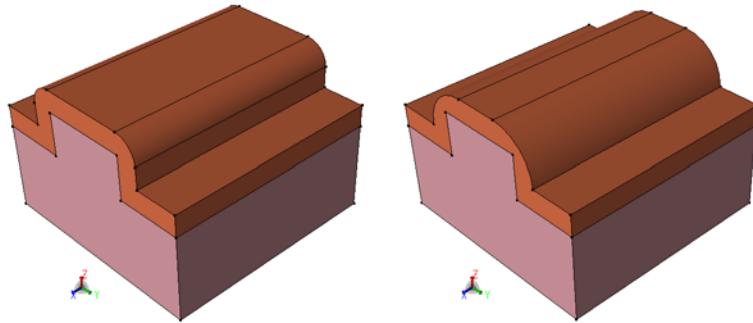
```
(sdepe:depo "thickness" 1 "material" "Copper" "radius" 2)
```

(In this case, the rounding radius is corrected automatically due to the small edge on the top exposed surface. Sentaurus Structure Editor displays the warning message "APPLIED DEPO BLEND RADIUS 1".)

Figure 91 (right) was created using:

```
(sdepe:depo "thickness" 1 "material" "Copper" "radius" 2  
"advanced-rounding" #t)
```

Figure 91 Deposition (left) without "advanced-rounding" and (right) with "advanced-rounding"



Etching

The `sdepe:etch-material` Scheme extension is used to perform an etching process emulation step.

Etching Keywords and Keyword Combinations

Table 29 lists all the keywords that the `sdepe:etch-material` command accepts in addition to the keywords that are defined in [Table 28](#) for deposition. It is organized around functional groups.

The *basic parameter group* contains the minimal set of keywords that can be used to perform an etching step in Procem (only the depth parameter belongs here).

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Table 29 *Etching keywords and keyword combinations*

Type	Keyword	Functionality
Basic parameter	"depth"	Etch offset distance. (Instead of "thickness", "depth" must be used in etching.)
Advanced controls (these groups can be combined)		
The keywords and keyword combinations that can be used for deposition can also be used for etching. See the rounding, tapering, directional deposition, and technical parameters, keywords, and keyword combinations in Table 28 . In addition to these functional groups, the following controls can be used:		
Overetch	"overetch"	Controls the amount of overetch that is applied during etching.
Selective etching	"time"	If the global <code>etchrates</code> list is defined, when you use "time", selective etching can be achieved.

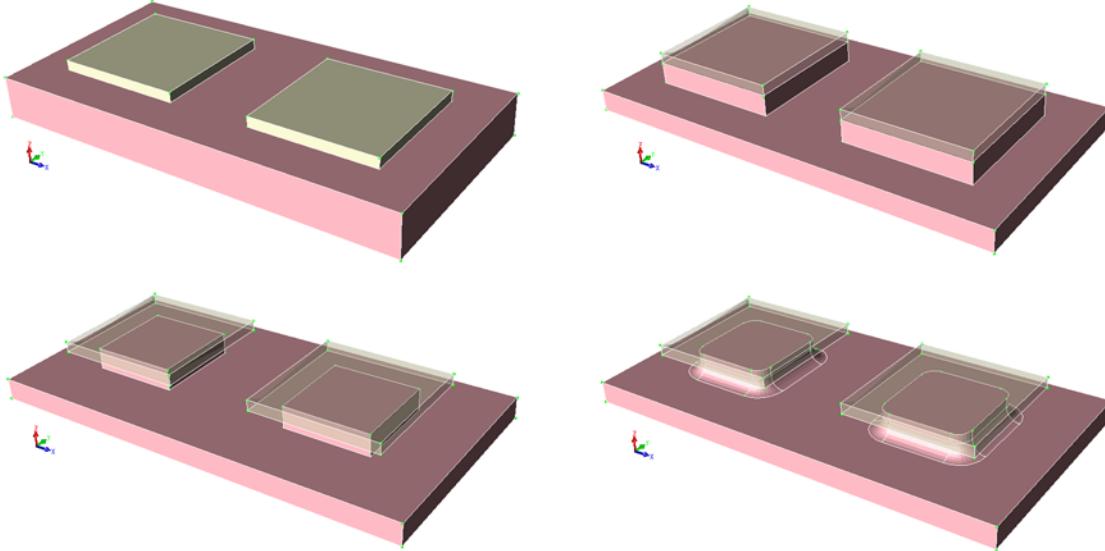
Example: Etching Operation Using Different Etch Options

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdepe:generate-mask "MASK1" (list (list 2 2 8 8) (list 12 2 18 8)))
(sdepe:define-pe-domain (list 0 0 20 10))
(sdepe:add-substrate "material" "Silicon" "thickness" 2)
(sdepe:pattern "mask" "MASK1" "polarity" "light" "material" "Resist"
               "thickness" 0.5)
; Use one of the following operations to perform the etch step
(sdepe:etch-material "material" "Silicon" "depth" 1 "type" "iso"
                      "algorithm" "lopx")
(sdepe:etch-material "material" "Silicon" "depth" 1 "type" "iso"
                      "algorithm" "lopx" "overetch" 1)
(sdepe:etch-material "material" "Silicon" "depth" 1 "type" "iso"
                      "algorithm" "lopx" "overetch" 1 "vexity" "all" "radius" 0.6)
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Figure 92 Etch operation: (upper left) original model, (upper right) etching without overetch, (lower left) etching with overetch, and (lower right) etching with overetch and rounding



Example: Multimaterial Etching

If the "time" parameter is used instead of "depth" and the global variable `etchrates` is set, then multimaterial etching can be performed. The following example illustrates multimaterial etching:

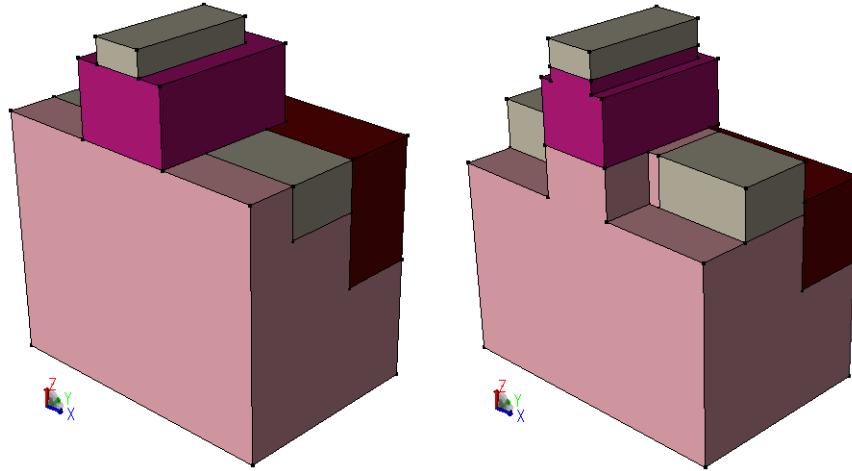
```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.2 0.8 1.0)
    "Silicon" "R.Substrate")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-cuboid (position 0.0 0.5 0.5) (position 1.2 0.8 1.0)
    "Oxide" "R.STI")
(sdegeo:create-cuboid (position 0.0 0.2 0.8) (position 0.4 0.5 1.0)
    "Silicide" "R.Source")
(sdegeo:create-cuboid (position 0.8 0.2 0.8) (position 1.2 0.5 1.0)
    "Silicide" "R.Drain")
(sdegeo:create-cuboid (position 0.4 0.0 1.0) (position 0.8 0.6 1.3)
    "PolySilicon" "R.Poly")
(sdegeo:create-cuboid (position 0.5 0.0 1.3) (position 0.7 0.5 1.4)
    "Silicide" "R.Gate")
; Multimaterial etch, using an etch rate table
(define etchrates (list (list "Oxide" 0.15) (list "PolySilicon" 0.05)
    (list "Silicon" 0.20) (list "Silicide" 0.00)))
(sdepe:etch-material "time" 1.0)
```

Figure 93 shows the initial model and multimaterial etching using the command
`(sdepe:etch-material "time" 1.0).`

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Figure 93 Multimaterial etching: (left) initial model and (right) multimaterial etching



Some Notes About Multimaterial Etching

The multimaterial etching code removes material (according to the etch table rates) only from the initially exposed bodies. If a given material layer is not exposed initially, it will not be etched if the etching is performed in one step. When etching is performed in multiple steps, the buried layers are exposed gradually, and the multimaterial etching can remove initially nonexposed layers as well.

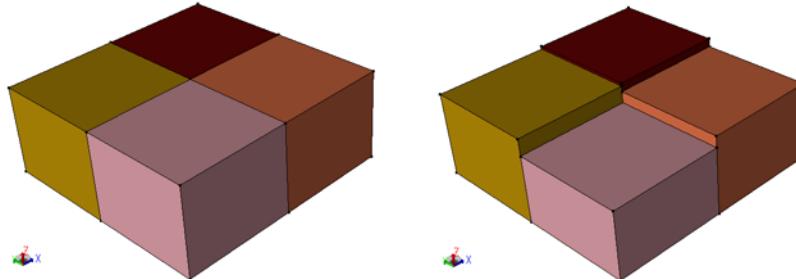
In the following example, the multimaterial etching works well in one step, since all materials, with specified nonzero etch rates, are exposed initially ([Figure 94](#) shows the initial structure and the etched model):

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 5 5 4) "Silicon" "x1")
(sdegeo:create-cuboid (position 5 0 0) (position 10 5 4) "Copper" "x2")
(sdegeo:create-cuboid (position 0 5 0) (position 5 10 4) "Nitride" "x3")
(sdegeo:create-cuboid (position 5 5 0) (position 10 10 4) "Oxide" "x4")
(define etchrates (list (list "Silicon" 1) (list "Copper" 0.5)
    (list "Nitride" 0.25) (list "Oxide" 0.125)))
(sdepe:etch-material "time" 1 "type" "aniso" "algorithm" "sweep")
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Figure 94 Multimaterial etch example (all bodies are exposed initially): (left) initial structure and (right) etched model

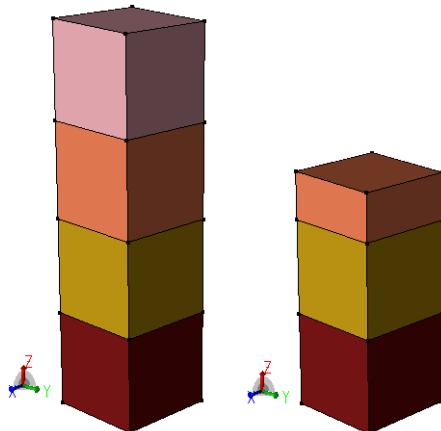


In the following example, only the silicon layer is exposed initially. The thickness of the silicon layer is 5, and the etch rate for the silicon is 1. This means that, if the time parameter is ≥ 5 , the complete silicon layer will be removed by the etch operation. If the etch operation would be performed in two steps, the first step would remove the silicon layer completely and the copper layer would be exposed and, in the second step, $5 \times 0.5 = 2.5$ thickness would be removed from the copper layer as well:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 5 5 5) "Oxide" "x1")
(sdegeo:create-cuboid (position 0 0 5) (position 5 5 10) "Nitride" "x2")
(sdegeo:create-cuboid (position 0 0 10) (position 5 5 15) "Copper" "x3")
(sdegeo:create-cuboid (position 0 0 15) (position 5 5 20) "Silicon"
    "x4")
(define etchrates (list (list "Silicon" 1) (list "Copper" 0.5)
    (list "Nitride" 0.25) (list "Oxide" 0.125)))
(sdepe:etch-material "time" 10 "type" "aniso" "algorithm" "sweep"
    "steps" 10)
```

Figure 95 shows the initial structure and the etched model.

Figure 95 Multimaterial etch example (only one layer is exposed initially): (left) initial structure and (right) etched model



If the thickness of each layer is known, an optimal step size can be selected. If the layer thicknesses are not known, a reasonably large step size is needed to etch all initially nonexposed material, with a nonzero etch rate.

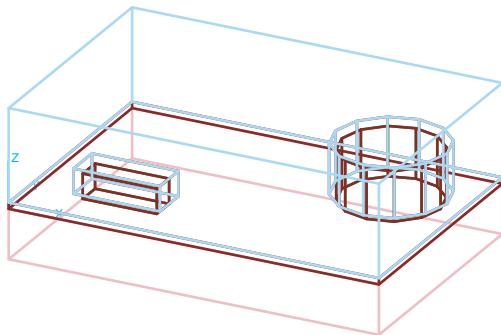
Some Notes About Shadowing

In the case of anisotropic etching, the internal buried materials (other than the etch material) will protect the underlying shadowed part. In the case of isotropic etching, the above described shadow protection is disabled, so the etch operation will remove etch material under the protected layers if these areas are connected to the exposed surface (that is, the initial exposed etch material touches these etch material layers). If there are buried (initially nonexposed) etch material layers, which are completely isolated by some other materials, these materials will be protected as well.

Fill

The command `sdepe:fill-device` can be used to perform a fill process-emulation step. The command `(sdepe:fill-device "material" "Gas" "height" 6)` was used to demonstrate the fill operation in [Figure 96](#). The optional "region" parameter can be used to specify the region name of the filled region.

Figure 96 Fill operation



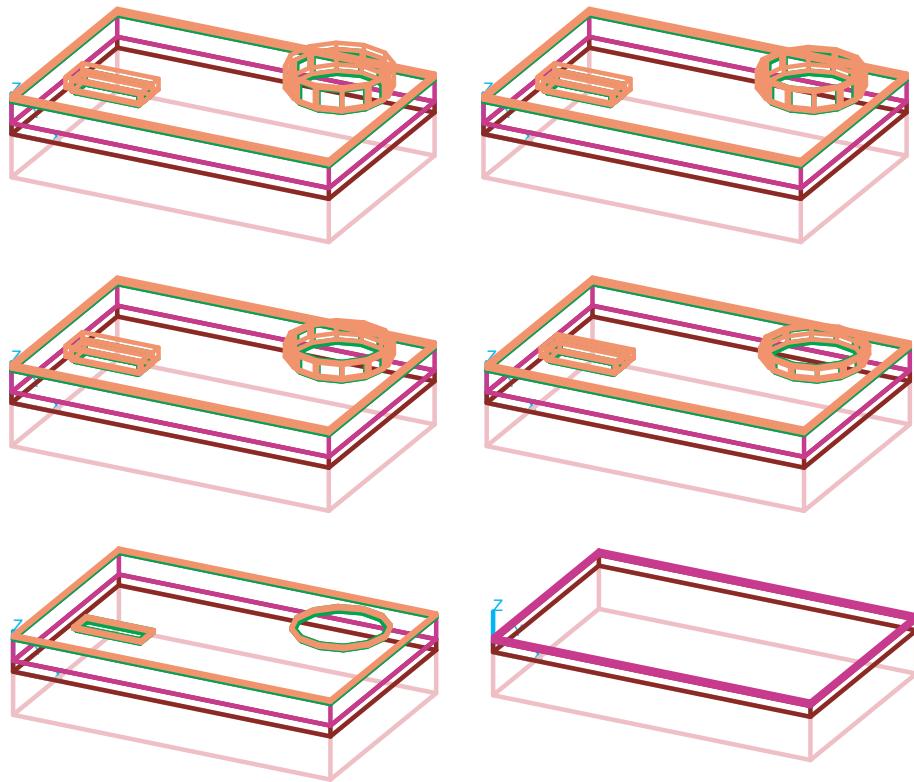
Polishing

The `sdepe:polish-device` Scheme extension can be used to perform a polishing process-emulation step. To perform selective polishing, the optional "material" parameter can be used to specify the bodies to be polished.

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Figure 97 Polishing steps starting from the upper-left corner and performing polish steps (left to right)



Interconnect Layer Generation

Complex interconnect structures can be created easily with the `sdepe:icon_layer` command. The basic assumption is that the initial structure is flat.

The `sdepe:icon_layer` command always creates a flat layer on top of the existing structure. The z-coordinates of the bottom and top of each layer are constant. Each layer consists of two materials: the interconnect material and the surrounding material. The interconnect body is created using the specified mask and mask polarity. The mask is swept from the top of the previous layer, and the sidewalls of the swept body can be tapered by specifying a taper angle in the arguments list.

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

[Table 30](#) lists the keywords for interconnect layer generation and keyword combinations.

Table 30 Keywords for interconnect layer generation and keyword combinations

Type	Keyword	Functionality
Basic parameters	"mask"	Name of the mask
	"thickness"	Layer thickness
	"polarity"	Mask polarity (default is "light")
	"ic-material"	Interconnect material (default is "Metal")
	"env-material"	Fill material (default is "Gas")
Basic control	"taper-angle"	Taper angle (default is 0)

Example: Generating an Interconnect Structure

The following script demonstrates the use of `sdepe:icon_layer` and [Figure 98](#) shows the generated structure:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdepe:define-pe-domain (list 1 1 9 9))
(sdepe:add-substrate "Silicon" 1.0)

;# Mask definition #
(sdepe:generate-mask "Vial" (list (list 2 6 3 7) (list 7 6 8 7)))
(sdepe:generate-mask "Metall1" (list (list 2 0 3 10) (list 7 0 8 10)))
(sdepe:generate-mask "Via2" (list (list 2 3 3 4) (list 7 3 8 4)))
(sdepe:generate-mask "Metal2" (list (list 0 2.5 10 4.5)))
(sdepe:icon_layer "mask" "Vial" "polarity" "light" "thickness" 0.4
                  "ic-material" "Aluminum" "env-material" "Oxide" "taper-angle" 20)
(sdepe:icon_layer "mask" "Metall1" "polarity" "light" "thickness" 0.6
                  "ic-material" "Aluminum" "env-material" "Oxide" "taper-angle" 10)
(sdepe:icon_layer "mask" "Via2" "polarity" "light" "thickness" 0.8
                  "ic-material" "Aluminum" "env-material" "Oxide" "taper-angle" 20)
(sdepe:icon_layer "mask" "Metal2" "polarity" "light" "thickness" 1.2
                  "ic-material" "Aluminum" "env-material" "Oxide" "taper-angle" 10)
```

Figure 98 Interconnect structure: (left) mask layout, (middle) final interconnect structure with oxide hidden, and (right) with oxide shown

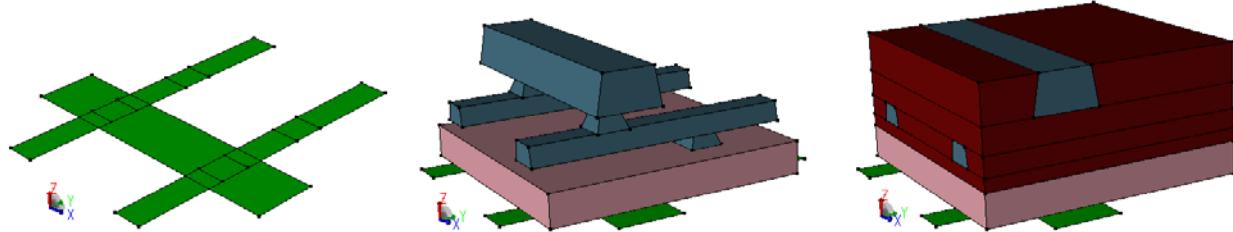
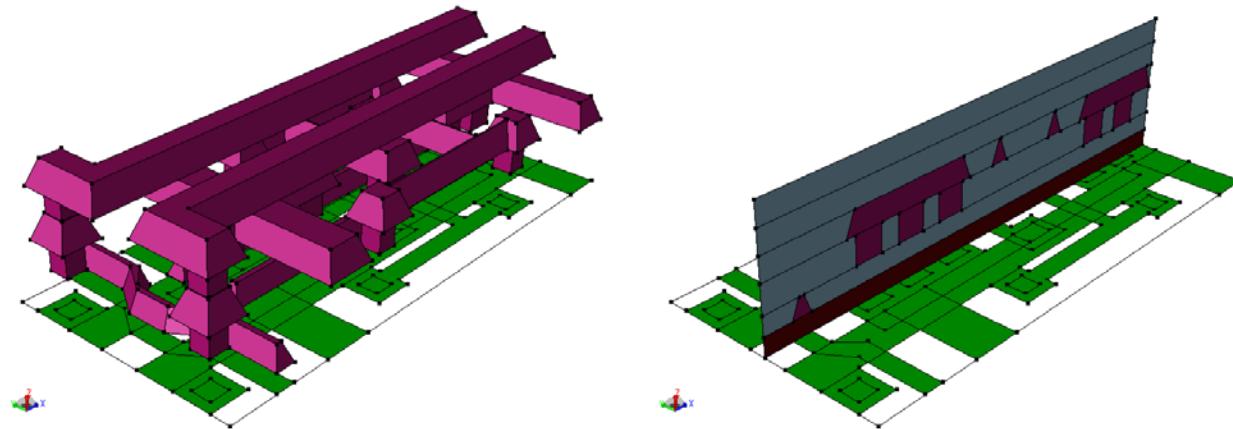


Figure 99 (Left) Example of an interconnect structure created using an external layout file and (right) 2D cut of interconnect structure showing tapered sidewalls



Shape Library

Sentaurus Structure Editor provides a set of commands to create special shapes (2D and 3D solid regions) that can be inserted into a 3D model:

- `PolygonSTI` creates a shallow trench isolation (STI) shape (2D).
- `PolygonWaferMask` creates a wafer shape (2D).
- `PolyHedronCylinder` creates a (tessellated) cylinder shape (3D).
- `PolyHedronEllipticCylinder` creates a (tessellated) elliptic cylinder shape (3D).
- `PolyHedronEpiDiamond` creates an epi diamond shape (3D).
- `PolyHedronSTI` creates an STI shape (3D).
- `PolyHedronSTIaccc` creates an STI shape for a concave active corner (3D).
- `PolyHedronSTIaccv` creates an STI shape for a convex active corner (3D).

You can define additional commands that create parameterized custom shapes using the scripting capabilities of Sentaurus Structure Editor.

PolygonSTI

This command creates a 2D STI-shaped polygon. The syntax is the same as for the PolyHedronSTI command, except it does not have the zmin, zmax, and material parameters (see [PolyHedronSTI on page 217](#)).

Syntax

```
(PolygonSTI name direction X0 Y0 Depth Tsti Asti Hsti Rd Rb Ru)
```

where:

- name is the name of the polygon.
- direction can be left or right only. This sets the facing direction of the STI polygon.
- [Figure 104 on page 218](#) describes the other parameters.

PolygonWaferMask

This command creates a 2D wafer mask polygon.

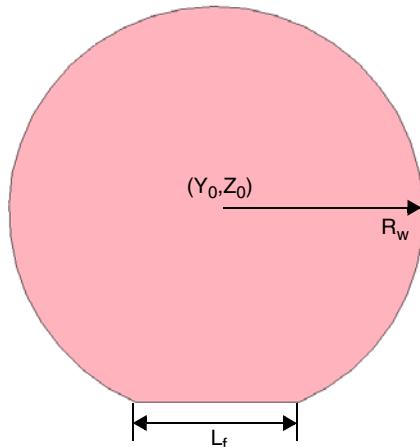
Syntax

```
(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 100](#)).

Figure 100 Wafer mask-shaped polygon



PolyHedronCylinder

This command creates a 3D cylinder-shaped polyhedron.

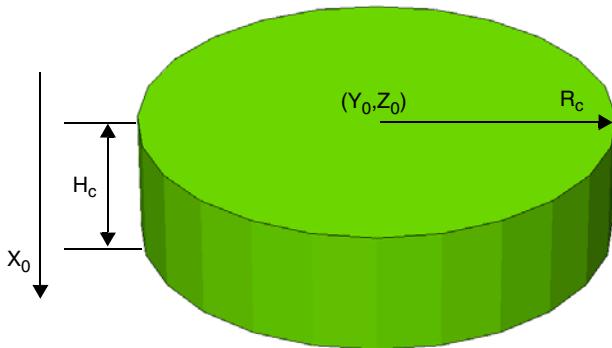
Syntax

```
(PolyHedronCylinder name X0 Y0 Z0 Rc Hc  
[material] [Rotate.Y | Rotate.Z] [angle])
```

where:

- **name** is the name of the polyhedron.
- **material** is optional and specifies the material of the inserted shape. The default is Oxide.
- 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 101](#)).

Figure 101 Cylinder-shaped polyhedron



PolyHedronEllipticCylinder

This command creates a 3D elliptic cylinder-shaped polyhedron.

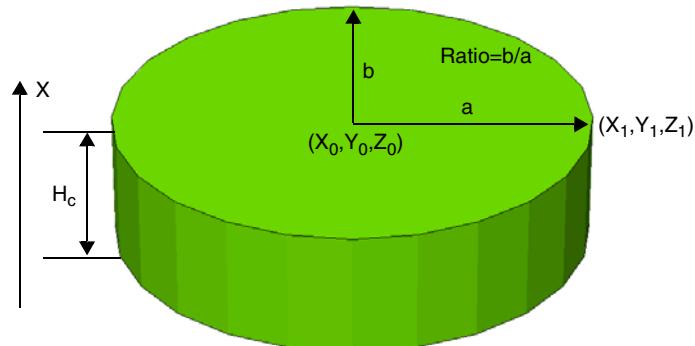
Syntax

```
(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.
- `x0, y0, z0` are the center coordinates of the base ellipse.
- `y1` and `z1` 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.
- `material` is optional and specifies the material of the inserted shape. The default is Oxide.
- If `Rotate.x` is specified, the cylinder is rotated at position $(x0, y0, z0)$ along the x-axis for angle `degrees`.
If `Rotate.y` is specified, the cylinder is rotated at position $(x0, y0, z0)$ 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 degrees (see [Figure 102](#)).

Figure 102 Elliptic cylinder-shaped polyhedron



PolyHedronEpiDiamond

This command creates a 3D epitaxial diamond-shaped polyhedron.

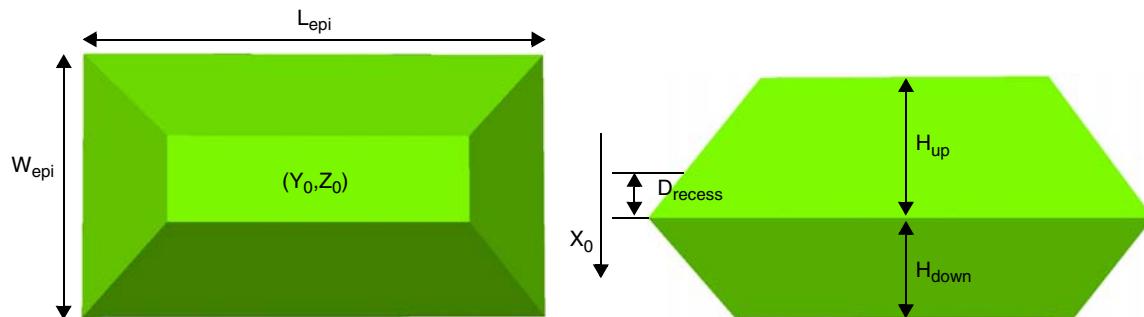
Syntax

```
(PolyHedronEpiDiamond name X0 Y0 Z0 Wepi Lepi Hup Hdown Drecess  
[material])
```

where:

- `name` is the name of the polyhedron.
- `material` is optional and specifies the material of the inserted shape. The default is Oxide.
- [Figure 103](#) describes the other parameters.

Figure 103 Parameters for generating epitaxial diamond-shaped polyhedron



PolyHedronSTI

This command creates a 3D STI-shaped polyhedron.

Syntax

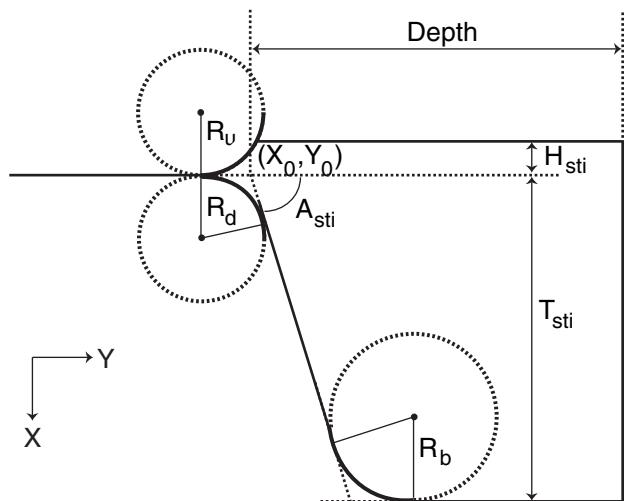
```
(PolyHedronSTI name direction X0 Y0 Depth Zmin Zmax Tsti Asti Hsti  
Rd Rb Ru [material])
```

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. The default is Oxide.
- [Figure 104](#) describes the other parameters.

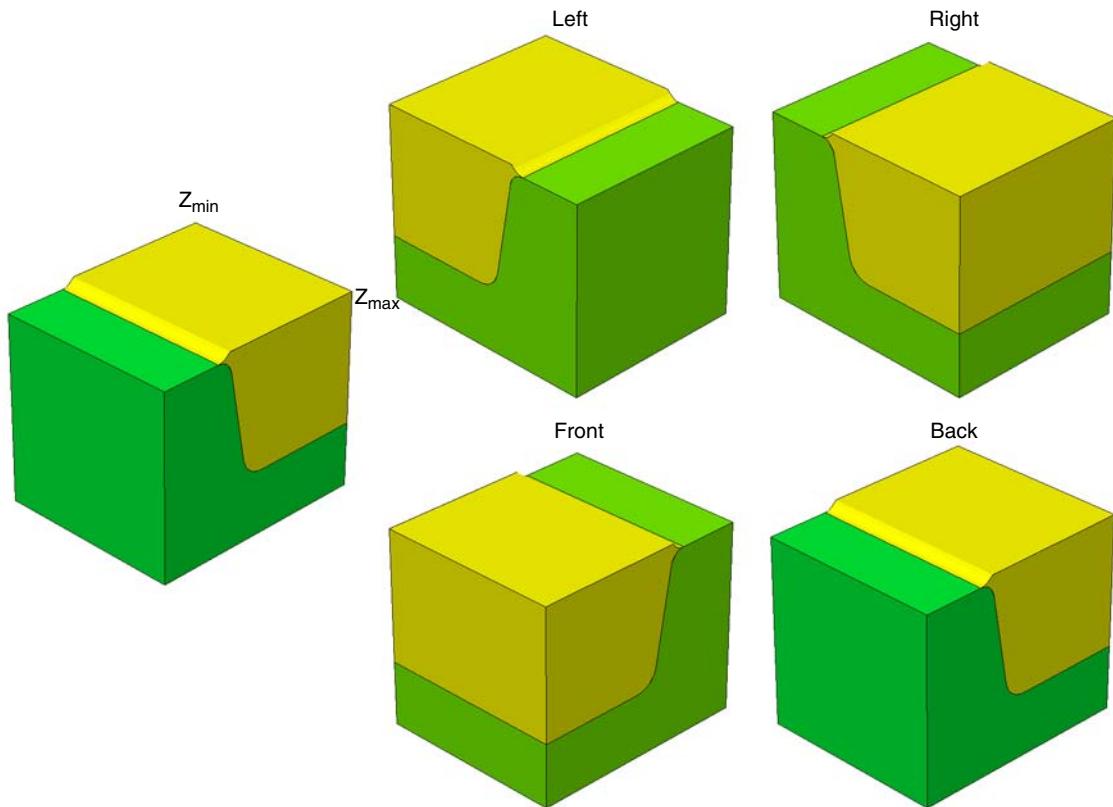
Chapter 5: Structure Generation Using Etching and Deposition
Process Emulation Steps

Figure 104 Parameters for generating STI-shaped polyhedron



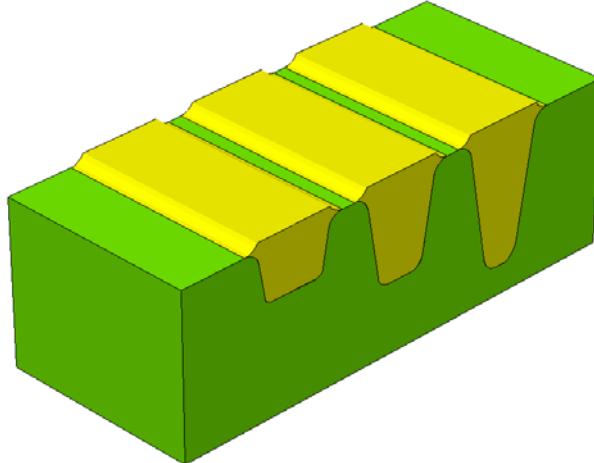
[Figure 105](#) shows some generated STI shapes in different directions.

Figure 105 STI-shaped polyhedra in different directions



[Figure 106](#) shows STI shapes with different T_{sti} and R_b values.

Figure 106 STI-shaped polyhedra with different T_{sti} and R_b



PolyHedronSTIacc

This command creates a 3D STI concave active corner-shaped polyhedron.

Syntax

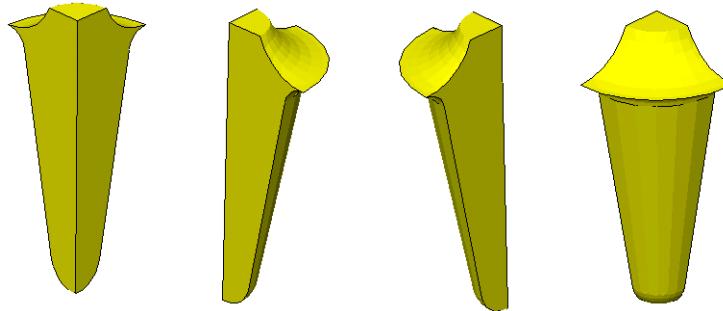
```
(PolyHedronSTIacc name direction x0 y0 z0 Tsti Asti Hsti Rd Rb Ru  
Rac [material])
```

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.
- `material` is optional and specifies the material of the inserted shape. The default is Oxide.
- [Figure 104 on page 218](#) describes the other parameters.

[Figure 107](#) shows STI concave corner-shaped polyhedra in different directions.

Figure 107 STI concave corner-shaped polyhedra in different directions: (from left to right) left back, right back, left front, and right front



PolyHedronSTIaccv

This command creates a 3D STI convex active corner-shaped polyhedron.

Syntax

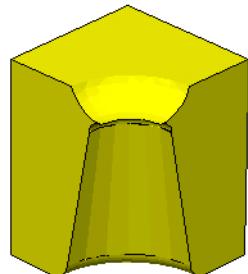
```
(PolyHedronSTIaccv name direction X0 Y0 Z0 Depth Tsti Asti Hsti  
Rd Rb Ru Rac [material])
```

where:

- `name` is the name of the polyhedron.
- `direction` can be `rb`, `lb`, `lf`, or `rf` (as for the `PolyHedronSTIaccc` command).
- `Rac` is the radius of the convex corner.
- `material` is optional and specifies the material of the inserted shape. The default is Oxide.
- [Figure 104 on page 218](#) describes the other parameters.

[Figure 108](#) shows a generated STI convex corner-shaped polyhedron.

Figure 108 STI convex corner-shaped polyhedron

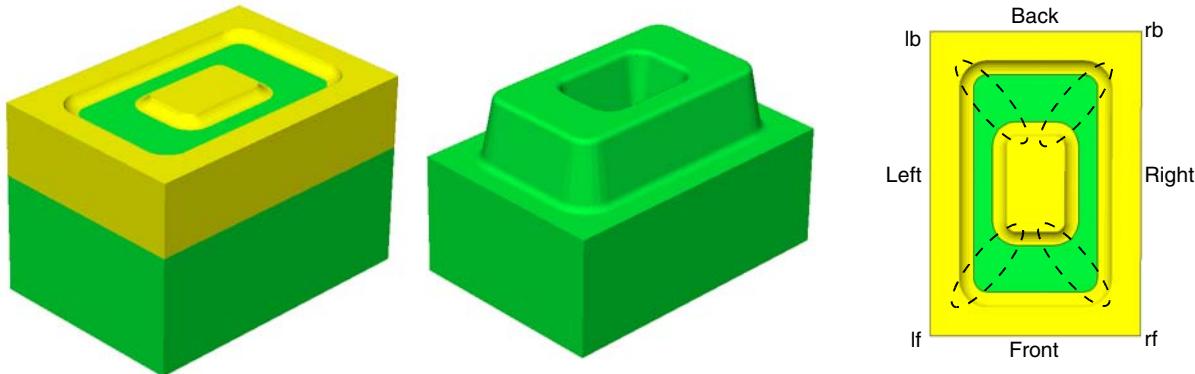


Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

Figure 109 (left) shows a structure generated by combining the above three STI commands.
Figure 109 (right) illustrates the directions of the STI shapes.

Figure 109 (Left) STI structures and (right) polyhedron directions



Removing Material

The command `sdepe:remove` can be used to remove a top layer (with a given "material" or "region" attribute). If there are multiple exposed bodies with the given material or region name, all these bodies are removed.

Doping and Implantation

For constant doping, see [sdepe:doping-constant-placement on page 754](#).

For a Gaussian doping profile, see [sdedr:define-gaussian-profile on page 548](#).

The following example uses doping and implantation commands:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 1.0 1.0)
    "Silicon" "region_1")
(sdegeo:create-cuboid (position -0.2 0.0 0.0) (position 0.0 0.3 0.3)
    "PolySilicon" "region_2")
(sdegeo:create-cuboid (position -0.2 0.4 0.7) (position 0.0 1.0 1.0)
    "Resist" "region_3")
(sdepe:doping-constant-placement "BackGround"
    "BoronActiveConcentration" 1e16 "region_1")
(sdedr:define-gaussian-profile "Implant_As"
    "ArsenicActiveConcentration" "PeakPos" 0.0 "PeakVal" 1.3e+18
    "ValueAtDepth" 1.0e16 "Depth" 0.5 "Erf" "Factor" 0.7)
(sdepe:implant "Implant_As")
(sdedr:define-refinement-size "RefDef.BG" 0.1 0.1 0.1 0.0125 0.0125
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Steps

```
0.0125)
(sdedr:define-refinement-function "RefDef.BG" "DopingConcentration"
  "MaxTransDiff" 1)
(sdedr:define-refinement-region "RefPlace.Si" "RefDef.BG" "region_1")
(sdedr:define-refinement-region "RefPlace.Po" "RefDef.BG" "region_2")
(sde:build-mesh "" "Implant_example")
```

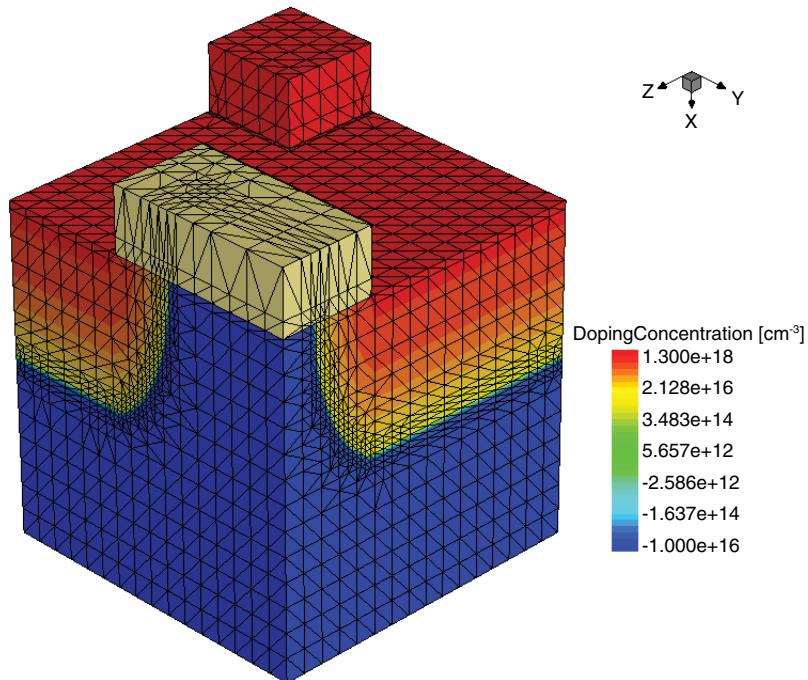
Note:

By default, the `sdepe:implant` command places the doping profile at all surfaces exposed to gas, including the vertical sidewalls. If implantation should be performed only on the inclined (nonvertical) faces, you must add the `"flat"` option to the `sdepe:implant` argument list.

A refinement window defined by using the `sdepe:implant` command, which is used for the doping, extends to the domain boundary by default. You can change this behavior by setting the global Scheme variable `extend-implant-window` to `#t`. Its default value is `#f`, but you can use the

`(set! extend-implant-window #t)` Scheme command to change it. If you set `extend-implant-window #t`, then the refinement window extends from the domain boundary by $3 \times Length$, where the *Length* value is taken from the doping profile definition. If the `Factor` parameter was used in the doping definition, then the extension distance will be $3 \times Factor$.

Figure 110 Gaussian implantation on a silicon example with a rectangle of resist on the surface; the lateral penetration of the higher doping is red underneath the resist



Process Emulation Example

This example illustrates some process emulation commands introduced in this chapter. The example is a complete Scheme script with variable definitions and process emulation commands. It contains commands for mask generation, domain definition, and substrate generation. Isotropic depositions, patterning steps, and etching operations are used to illustrate process emulation steps with Procem.

The example shows a backend (interconnect) structure generation using a dual-damascene process emulation. First, two oxide layers (separated by a nitride layer) are deposited on the silicon substrate. Second, vias are etched in the bottom oxide layer followed by etching of lines in the top oxide layer. A thin barrier layer is deposited in the etched vias and lines. Subsequently, they are filled with copper, which is finally polished to form the metal lines.

The Scheme code for generating an interconnect structure is:

```
; Clear the database
(sde:clear)
(sde:set-process-up-direction "+z")
(define W 2.2)           ; Total width of the simulation domain.
(define H 2.8)           ; Total length of the simulation domain.
(define HD 3.0)          ; Total domain length.

; Metal line mask creation
(sdepe:generate-mask "MASK0" (list
    (list 0.2 2.8 0.2 -0.6 2 -0.6 2 -0.3 0.5 -0.3 0.5 2.8)
    (list 0.7 0 1 2.8)
    (list 1.2 0 1.5 2.8)
    (list 1.7 0 2 2.8)))

; Via mask creation
(sdepe:generate-mask "MASK1" (list
    (list 0.25 1.2 0.45 1.4)
    (list 0.75 0.8 0.95 1)
    (list 1.25 0.4 1.45 0.6)))

; Process Emulation Steps:
; Define domain
(sdepe:define-pe-domain (list 0.0 -1.0 W HD))

; Create substrate
(sdepe:add-substrate "material" "Silicon" "thickness" 0.5)

; First layer of dielectric deposition for vias/contacts
(sdepe:depo "material" "Oxide" "thickness" 0.3)

; Nitride deposition
(sdepe:depo "material" "Nitride" "thickness" 0.05)

; Second layer of dielectric deposition for metal lines
(sdepe:depo "material" "Oxide" "thickness" 0.3)
```

Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Example

```
; Vias pattern and etching
(sdepe:pattern "mask" "MASK1" "polarity" "dark" "material" "Resist"
  "thickness" 0.1 "type" "iso")
(sdepe:etch-material "material" "Oxide" "depth" 0.65 "type" "aniso")
(sdepe:etch-material "material" "Nitride" "depth" 0.05 "type" "aniso")
(sdepe:remove "material" "Resist")

; Metal lines pattern and etching
(sdepe:pattern "mask" "MASK0" "polarity" "dark" "material" "Resist"
  "thickness" 0.1)
(sdepe:etch-material "material" "Oxide" "depth" 1.0 "type" "aniso")
(sdepe:remove "material" "Resist")

; Barrier layer deposition
(sdepe:depo "material" "Gold" "thickness" 0.02)

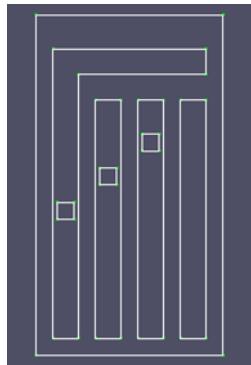
; Metal (copper) deposition
(sdepe:depo "Copper" 0.2)

; Metal CMP
(sdepe:polish-device "thickness" 0.2)

; Fill with copper
(sdepe:fill-device "material" "Copper")
```

The following figures illustrate the process steps that were performed by the script.

Figure 111 Generated masks and domain boundary; the external rectangle wire shows the simulation domain



Chapter 5: Structure Generation Using Etching and Deposition
Process Emulation Example

Figure 112 State of process emulation after the first pattern step (MASK1 as a 'dark' mask was used); the patterned resist layer is translucent

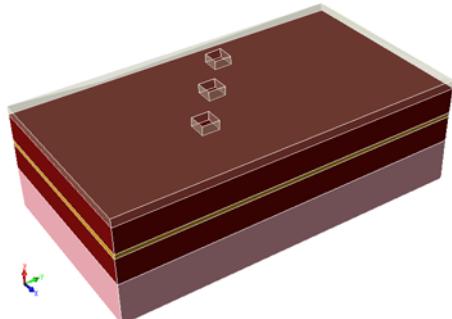


Figure 113 A 2D cross section of the device after the first two etching steps (the vias mask is removed)

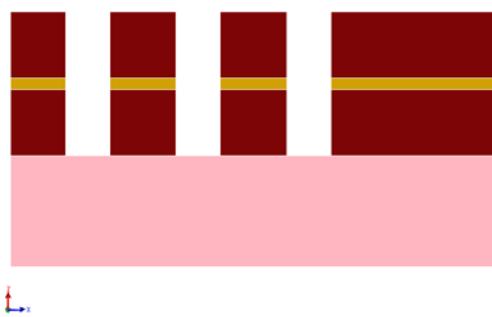
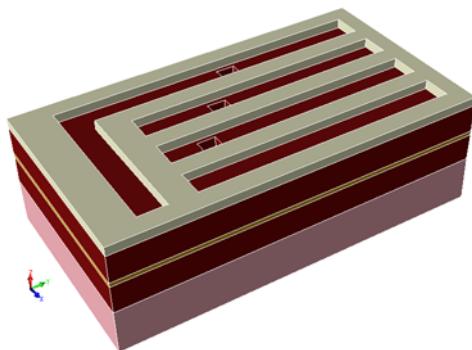


Figure 114 State of process emulation after metal lines are patterned (MASK1 with 'dark' polarity)



Chapter 5: Structure Generation Using Etching and Deposition

Process Emulation Example

Figure 115 Deposited barrier layer is shown separately from two different angles

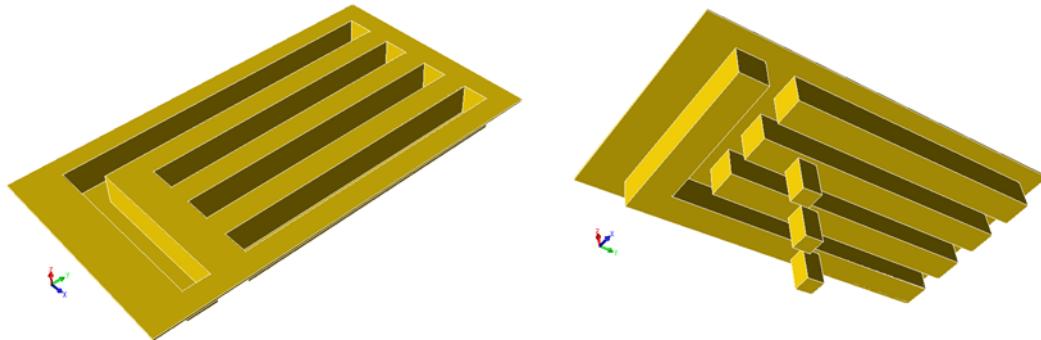
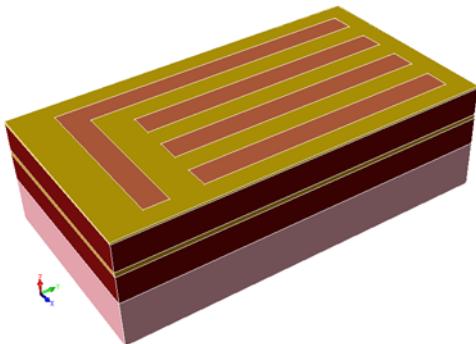


Figure 116 Final structure after final metal layer (copper) deposition and after last polish step is performed



6

Electrical and Thermal Contacts

This chapter describes how to define contacts in Sentaurus Structure Editor.

Overview

Contacts are interfacial regions where subsequent TCAD tools, such as Sentaurus Device, apply electrical, thermal, or other boundary conditions. They correspond to physical contacts in the real device. Contacts are edges in two dimensions and faces in three dimensions. The contact edges and faces are part of the geometry.

Before an edge or a face can be assigned to a contact, the contact itself must be declared and activated first.

There are distinct methods to assign edges or faces to contacts. You can either use an explicit edge or face list, or select a region. In the latter case, all boundary edges or boundary faces will be assigned to the contact.

Contacts are respected by all geometry-modifying operations. For example:

- Contacts are maintained or updated as needed in transformations (scaling, translation, rotation, and reflection).
- Two-dimensional contact edges are transformed to 3D contact faces in operations such as extrusion and sweeping.
- Three-dimensional contact faces are reduced to 2D contact edges in operations such as slicing.

A contact can consist of several disjoint edges or faces.

Chapter 6: Electrical and Thermal Contacts

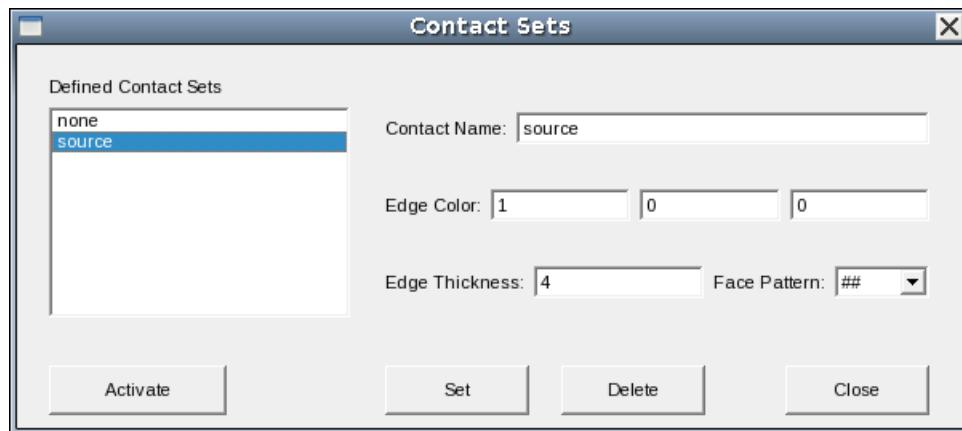
Defining and Activating a Contact

Defining and Activating a Contact

To define and activate a contact:

1. Choose **Contacts > Contact Sets**.

The Contact Sets dialog box opens.



2. Enter a contact name.
3. Edit the color and line thickness for drawing the 2D contact edges if required.
4. Select the pattern to mark 3D contact faces if required.
5. Click **Set** to complete the contact declaration.

The new contact name is displayed the Defined Contact Sets list.

6. Select the newly defined contact from the Defined Contact Sets list, and click **Activate**.

The active contact is now displayed in the Contact list in the main window of Sentaurus Structure Editor.

An already defined contact can be activated directly by selecting it from the Contact list of the main window.

The corresponding Scheme command is:

```
(sdegeo:define-contact-set contact-name edge-thickness  
(color:rgb red green blue) pattern)
```

For example:

```
(sdegeo:define-contact-set "substrate" 4 (color:rgb 1 0 0) "##")  
(sdegeo:set-current-contact-set "substrate")
```

Chapter 6: Electrical and Thermal Contacts

Deleting a Contact

The contact edge color is defined using three values, each ranging from 0 to 1. These values give the relative intensity of the three basic colors: red, green, and blue. For example:

- `red=(color:rgb 1 0 0)`
- `green=(color:rgb 0 1 0)`
- `blue=(color:rgb 0 0 1)`
- `yellow=(color:rgb 1 1 0)`
- `cyan=(color:rgb 0 1 1)`
- `purple=(color:rgb 1 0 1)`
- `gray=(color:rgb 0.5 0.5 0.5)`

Deleting a Contact

To delete a contact definition:

1. Choose **Contacts > Contact Sets**.
2. Click the respective contact in the Defined Contact Sets list.
3. Click **Delete**.

The corresponding Scheme command is:

```
(sdegeo:delete-contact-set contact-name)
```

Assigning Edges or Faces to a Contact

To assign edges (2D) or faces (3D) to the currently active contact:

1. Click the **Select** button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Edge** (for 2D) or **Select Face** (for 3D).
3. Select an edge (or a face) and hold the Shift key to select additional edges (or faces), or drag a box around a set of edges (or faces).
4. Choose **Contacts > Set Contact**.

This command checks the type of the selected entities. If only edges are selected, it assigns a 2D contact to the selected entities (same as choosing **Contacts > Set Edges**). If only faces are selected (and the faces belong to 3D bodies), it assigns a 3D contact to the selected entities (same as choosing **Contacts > Set Faces**).

Chapter 6: Electrical and Thermal Contacts

Assigning a Region Boundary to a Contact

Alternatively, right-click anywhere and choose **Contacts**. From the submenu, select an already defined contact to assign all selected edges or faces to this contact, or select **Contact Sets** to assign the edges or faces to a new contact. Define and activate this contact as described in [Defining and Activating a Contact on page 228](#).

5. Choose **Contacts > Set Edges** (for 2D) or **Contacts > Set Faces** (for 3D).

After closing the Contact Sets dialog box, all selected edges or faces are assigned to the newly defined contact.

For example, for 2D structures, the corresponding Scheme commands are:

```
(sdegeo:create-rectangle (position 0.0 0.0 0) (position 1.0 1.0 0)
    "Silicon" "region_1")
(sdegeo:set-contact (find-edge-id (position 0.5 0.0 0)) "cont")
```

For example, for 3D structures, the corresponding Scheme commands are:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 1.0 1.0)
    "Silicon" "region_1")
(sdegeo:set-contact (find-face-id (position 0.5 0.5 1.0)) "cont")
```

Assigning a Region Boundary to a Contact

To assign all region boundary edges (2D) or all region boundary faces (3D) to the currently active contact:

1. Click the **Select** button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Body**.
3. Select a body and hold the Shift key to select additional bodies, or drag a box around a set of bodies.
4. Choose **Contacts > Set Region Boundary Edges**.

Choose **Contacts > Set Region Boundary Faces**.

The corresponding Scheme command is:

```
(sdegeo:set-contact entity | entity-list
[contact-name ["remove"] ["imprint" BOOLEAN]])
```

Specifying the "remove" option deletes the region and converts its boundaries into a contact. If you specify "imprint" #f, the 3D contact is not assigned to the touching faces. The default behavior is "imprint" #t, in which case, the 3D contact is imprinted to the touching neighbor faces.

Chapter 6: Electrical and Thermal Contacts

Removing Edges or Faces From a Contact

For example, for 2D structures, the corresponding Scheme commands are:

```
(sde:clear)
(sdegeo:create-rectangle (position 0.0 0.0 0) (position 1.0 1.0 0)
    "Silicon" "region_1")
(sdegeo:set-default-boolean "ABA")
(define VIA (sdegeo:create-rectangle
    (position 0.25 0.75 0) (position 0.75 1.5 0) "Metal" "region_2"))
(sdegeo:set-contact VIA "cont" "remove")
```

For example, for 3D structures, the corresponding Scheme commands are:

```
(sde:clear)
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 1.0 1.0)
    "Silicon" "region_1")
(sdegeo:set-default-boolean "ABA")
(define VIA (sdegeo:create-cuboid (position 0.25 0.25 0.75)
    (position 0.75 0.75 1.5) "Metal" "region_2"))
(sdegeo:set-contact VIA "cont" "remove")
```

You can use the `sdegeo:set-contact` function to assign contacts to geometric edges or faces depending on model dimensionality. In three dimensions, this function attaches the "3d-contact" attributes and the *no merge* attributes to the specified faces.

Removing Edges or Faces From a Contact

The procedures to remove edges or faces from a contact are similar to the procedures to assign edges or faces to a contact (see [Assigning Edges or Faces to a Contact on page 229](#) and [Assigning a Region Boundary to a Contact on page 230](#)). After selecting the edges, faces, or bodies, select the respective option by choosing **Contacts > Unset**.

The corresponding Scheme commands are:

```
(sdegeo:delete-contact-boundary-edges [body | body-list])
(sdegeo:delete-contact-boundary-faces [body | body-list])
(sdegeo:delete-contact-edges [edge | edge-list])
(sdegeo:delete-contact-faces [face | face-list])
```

As an alternative to the above procedures:

1. Click the **Select** button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Edge** (2D) or **Select Face** (3D).
3. Select an edge (or a face) and hold the Shift key to select additional edges (or faces), or drag a box around a set of edges (or faces).

Chapter 6: Electrical and Thermal Contacts

Creating Edges or Faces for Use in Contacts

4. Right-click anywhere and choose **Contacts**. From the submenu, view the list of contact sets. A check mark shows which contact sets have already been placed at the selected entity or entities.
5. Deselect the required contact to unset the selected edges or faces as that contact set.

Creating Edges or Faces for Use in Contacts

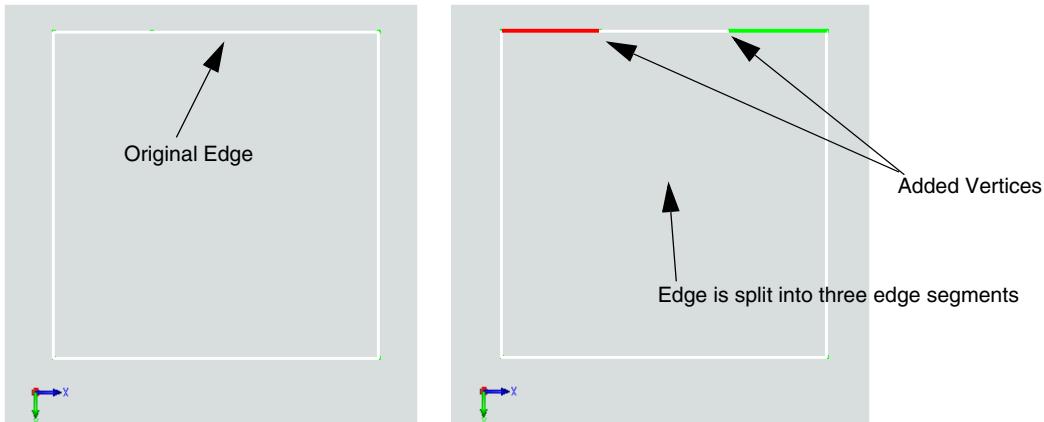
Only an entire edge or face can be assigned to a contact. If the contact should cover only part of an edge or a face, the edge or face must be split explicitly.

In two dimensions, a convenient way to accomplish this task is to add vertices (see [Adding a Vertex on page 85](#)).

For example, the following Scheme commands create a rectangle, split one of its edges into three segments by inserting two new vertices, and assign the contact "cont1" to the side edge segments (see [Figure 117](#)):

```
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silicon"
    "region_1")
(sdegeo:insert-vertex (position 0.3 0.0 0.0))
(sdegeo:insert-vertex (position 0.7 0.0 0.0))
(sdegeo:set-contact (find-edge-id (position 0.15 0 0)) "cont1")
(sdegeo:set-contact (find-edge-id (position 0.85 0 0)) "cont2")
```

Figure 117 Adding contacts to 2D edge segments



In three dimensions, Sentaurus Structure Editor provides dedicated functions to imprint a shape on a surface to split existing faces.

Chapter 6: Electrical and Thermal Contacts

Protecting Contacts

To split faces in three dimensions by imprinting a rectangle:

1. Choose **Contacts > Imprint Rectangle**.
2. Drag to draw the diagonal of the rectangle in the view window.

To split faces in three dimensions by imprinting a polygon:

1. Choose **Contacts > Imprint Polygon**.
2. Click at each vertex of the polygon in the view window. To finish drawing the polygon, click the middle mouse button. (Press both buttons when using a two-button mouse.)

The polygon is closed automatically.

To split faces in three dimensions by imprinting a circle:

1. Choose **Contacts > Imprint Circle**.
2. Drag to draw the radius of the circle in the view window.

The corresponding Scheme commands are:

```
(sdegeo:imprint-rectangular-wire position position)
(sdegeo:imprint-polygonal-wire position-list)
(sdegeo:imprint-circular-wire position radius)
```

For example:

```
(sdegeo:create-cuboid (position 0.0 0.0 -1.0) (position 3.0 1.0 0.0)
  "Silicon" "region_1")
(define RECT (sdegeo:imprint-rectangular-wire
  (position 0.2 0.2 0) (position 0.8 0.8 0)))
(define POLY (sdegeo:imprint-polygonal-wire (list (position 1.2 0.2 0)
  (position 1.8 0.2 0) (position 1.8 0.8 0) (position 1.5 0.8 0)
  (position 1.5 0.5 0) (position 1.2 0.5 0) (position 1.2 0.2 0))))
(define CIRC (sdegeo:imprint-circular-wire (position 2.5 0.5 0) 0.3))
```

The imprint commands return a list containing the newly generated faces. This list can be used in `sdegeo:set-contact`. After the new faces are created, the contact generation procedure is the same as that discussed in [Assigning Edges or Faces to a Contact on page 229](#).

When used from the GUI, the imprint commands place the rectangle, polygon, or circle, in the current work plane. Change the work plane as discussed in [Work Planes on page 152](#) to place imprint shapes on a general face.

Protecting Contacts

If contacts are assigned to edges and faces using any of the `sdegeo` contact assignment commands, these contacts are protected.

Chapter 6: Electrical and Thermal Contacts

Examples of Contact Assignments

The 2D boundary simplification commands do not remove or merge contact edges, and also the 3D boundary regularization command `sde:bool-regularise` does not merge coplanar contact faces with adjoining faces.

If an application defines the edge or face contacts directly, by attaching the "2d-contact" or "3d-contact" attributes to the contact edges or faces, these contacts are not protected. In this case, the `(protect-all-contacts)` command can be used to add the necessary *no merge* attribute to the contact edges or faces.

Examples of Contact Assignments

These examples illustrate how to create different 2D and 3D contacts.

Creating Different 2D Contacts

Create the device geometry:

```
(sdegeo:create-rectangle (position -1.0 0.0 0) (position 1.0 1.0 0)
    "Silicon" "R.Substrate")
(sdegeo:create-rectangle (position -0.4 0.0 0) (position 0.4 -0.05 0)
    "Oxide" "R.Gox")
(sdegeo:create-rectangle (position -0.4 -0.05 0) (position 0.4 -0.5 0)
    "Nitride" "R.Spacer")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position -0.2 -0.05 0) (position 0.2 -0.5 0)
    "PolySi" "R.Poly")
(define TMP (sdegeo:create-ellipse (position -1.0 0.0 0)
    (position -0.5 0.00 0) 0.2 "Silicide" "R.Silicide"))
(sdegeo:delete-region TMP)
```

Assign an existing edge to the "substrate" contact:

```
(sdegeo:set-contact (find-edge-id (position 0.0 1.0 0)) "substrate")
```

Split an edge and assign part of the original edge to the "drain" contact:

```
(sdegeo:insert-vertex (position 0.5 0.0 0))
(sdegeo:set-contact (find-edge-id (position 0.75 0.0 0)) "drain")
```

Assign all region boundary edges to the "gate" contact and remove the region:

```
(sdegeo:set-contact (find-body-id (position 0 -0.275 0)) "gate"
    "remove")
```

Assign edges along a complex topology to the "source" contact as follows.

In the Old Replaces New (BAB) overlap resolution mode (see [Overlap Behavior on page 79](#)), a dummy body is created that includes all boundary edges, which should be assigned to the

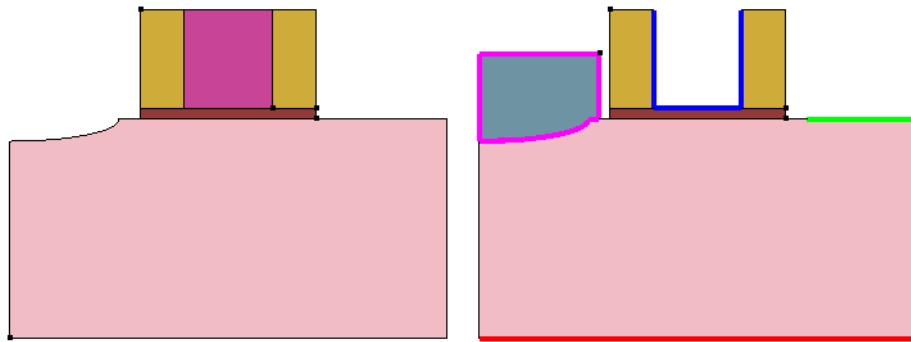
Chapter 6: Electrical and Thermal Contacts

Examples of Contact Assignments

"source" contact. Then, the region boundary edges of this dummy body are assigned to the "source" contact, and the dummy body itself is removed:

```
(sdegeo:set-default-boolean "BAB")
(define DUMMY (sdegeo:create-rectangle (position -1.0 -0.3 0)
                                         (position -0.45 0.2 0) "Aluminum" "R.Dummy"))
(sdegeo:set-contact DUMMY "source" "remove")
```

Figure 118 Two-dimensional contact creation example: (left) initial device geometry structure with "substrate" (red), "drain" (green), "gate" (blue), and "source" (purple) contacts; and (right) immediately before deleting the dummy body used to create the "source" contact



Creating Different 3D Contacts

Create the device geometry:

```
(sdegeo:create-cuboid (position -1.0 0.0 0.0) (position 1.0 1.0 -1.0)
                      "Silicon" "R.Substrate")
(sdegeo:create-cuboid (position -0.4 0.0 0.0) (position 0.4 1.0 0.05)
                      "Oxide" "R.Gox")
(sdegeo:create-cuboid (position -0.4 0.0 0.05) (position 0.4 1.0 0.5)
                      "Nitride" "R.Spacer")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-cuboid (position -0.2 0.0 0.05) (position 0.2 1.0 0.5)
                      "PolySi" "R.Poly")
(sdegeo:create-cuboid (position -1.0 0.5 0.0) (position 1.0 1.0 -0.5)
                      "Oxide" "R.STI")
(sdegeo:set-default-boolean "ABA")
(define TMP (sdegeo:create-sphere (position -0.75 0.25 0.0) 0.2
                                    "Silicide" "R.Silicide"))
(sdegeo:delete-region TMP)
```

Assign an existing face to the "substrate" contact:

```
(sdegeo:set-contact (find-face-id (position 0.0 0.5 -1.0))
                     "substrate")
```

Chapter 6: Electrical and Thermal Contacts

Examples of Contact Assignments

Imprint a rectangle on a face in the base work plane, and assign the new face to the "drain" contact:

```
(define DRAINFACE (sdegeo:imprint-rectangular-wire  
    (position 0.5 0.45 0) (position 0.95 0.05 0)))  
(sdegeo:set-contact DRAINFACE "drain")
```

Imprint a rectangle on a face in a custom work plane ("Sidewall"), and assign the new face to the "thermode" contact:

```
(sdegeo:define-work-plane "Sidewall" (position -1 1 -1)  
    (position -1 0 -1) (position -1 1 0))  
(sdegeo:set-active-work-plane "Sidewall")  
(define THERMFACE (sdegeo:imprint-rectangular-wire  
    (position 0.05 0.05 0) (position 0.95 0.45 0)))  
(sdegeo:set-active-work-plane "base")  
(sdegeo:set-contact THERMFACE "thermode")
```

Imprint a polygon on a general face using exploited 3D coordinates, and add the new face to the "thermode" contact:

```
(define THERMFACE_2 (sdegeo:imprint-polygonal-wire (list  
    (position 1.0 0.05 -0.95) (position 1.0 0.95 -0.95)  
    (position 1.0 0.95 -0.55) (position 1.0 0.45 -0.55)  
    (position 1.0 0.45 -0.05) (position 1.0 0.05 -0.05)  
    (position 1.0 0.05 -0.95)))  
(sdegeo:set-contact THERMFACE_2 "thermode")
```

Assign all region boundary faces to the "gate" contact and remove the region:

```
(sdegeo:set-contact (find-body-id (position 0.0 0.5 0.3)) "gate"  
    "remove")
```

Assign faces along a complex topology to the "source" contact as follows.

In the Old Replaces New (**BAB**) overlap resolution mode (see [Overlap Behavior on page 79](#)), a dummy body is created that includes all boundary faces, which should be assigned to the "source" contact.

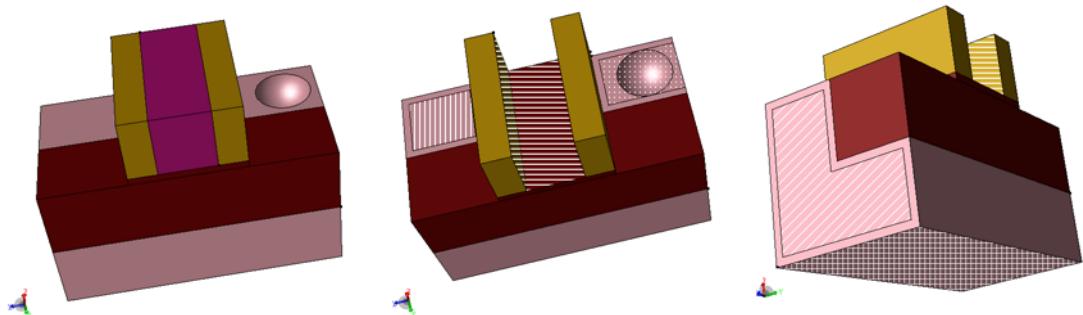
Then, the region boundary faces of this dummy body are assigned to the "source" contact, and the dummy body itself is removed:

```
(sdegeo:set-default-boolean "BAB")  
(define DUMMY (sdegeo:create-cuboid (position -1.0 0.05 -0.3)  
    (position -0.45 0.45 0.1) "Aluminum" "R.Dummy"))  
(sdegeo:set-contact DUMMY "source" "remove")
```

Chapter 6: Electrical and Thermal Contacts

Examples of Contact Assignments

Figure 119 Three-dimensional contact creation example: (left) initial device geometry, (middle) structure top view with “drain”, “gate”, and “source” contacts, and (right) structure bottom view with “thermode” and “substrate” contacts



7

Generating Mesh and Doping Profiles

This chapter presents the meshing operations available in Sentaurus Structure Editor.

Overview of Meshing Operations

In addition to defining the geometry of a structure, doping profiles and refinement parameters can be defined for different parts of the structure. The placement of these profiles and refinements as well as the areas to which the use of these profiles and refinements are restricted can be specified by user-defined refinement/evaluation (Ref/Eval) windows.

Ref/Eval windows are implemented as geometric objects and can be manipulated in a similar manner as other geometric objects. The difference between a regular geometric body and a Ref/Eval window is that material and region properties are attached to a regular geometric body, while a Ref/Eval window name is attached to a Ref/Eval window. Certain operations affect only regular geometric bodies. For example, only regular geometric bodies can be written to a tessellated boundary file. Some query functions, such as `(get-body-list)`, handle only regular geometric bodies. On the other hand, only Ref/Eval windows can be used in refinement and doping specifications as evaluation windows, for example. Ref/Eval windows have their own query functions, such as `(get-drs-list)`, which returns the entity IDs of all defined Ref/Eval windows.

In some cases, doping profiles and refinements can also be restricted to a specific region or material (materialwise and regionwise refinements and dopings).

Unlike regular geometric objects, Ref/Eval windows can overlap. For example, the refinement in a given area of the device can be determined by the tightest requirements among several refinement placements active in that area. For overlapping doping placements, the resulting profile is the sum of all profiles.

Sentaurus Structure Editor allows the full flexibility of definitions and placements available in the input syntax of Sentaurus Mesh. Profiles and refinements are first *defined* by specifying all the necessary parameters, but their use is not restricted to a specific location or an area. In a second step, profiles and refinements are *placed*. The placement step links a given profile or refinement definition with an area of validity, which can be a Ref/Eval window, a region name, or a material.

Chapter 7: Generating Mesh and Doping Profiles

Defining Areas for Mesh Refinement or Doping

Both profile and refinement definitions as well as Ref/Eval windows can be reused in various placements for maximum flexibility.

Defining Areas for Mesh Refinement or Doping

Ref/Eval windows are areas in which a certain mesh refinement or doping profile is applied. These reference areas can take the form of a line segment, a rectangle, a polygon, a cuboid, or a polygon automatically extracted from a face.

Line-Segment Ref/Eval Windows

Line-segment Ref/Eval windows are used as reference edges for placing implant-like doping profiles in two dimensions.

To define a line-segment Ref/Eval window:

1. Choose **Mesh > Define Ref/Eval Window > Line**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).
2. Click the starting point and the endpoint of the line segment in the view window.

The corresponding Scheme command is:

```
(sdedr:define-refeval-window RefEval-name "Line" position position)
```

For example:

```
(sdedr:define-refeval-window "RefEvalWin_1" "Line"  
    (position 0.0 0.0 0) (position 1.0 0.0 0))
```

Rectangular and Polygonal Ref/Eval Windows

Rectangular and polygonal Ref/Eval windows are used to place mesh refinement or constant profiles in two dimensions, and are used as reference faces to place implantation-like doping profiles in three dimensions.

To define a rectangular or polygonal Ref/Eval window:

1. Choose **Mesh > Define Ref/Eval Window > Rectangle (or Polygon)**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).
2. For a rectangle, drag to draw the diagonal of the rectangle in the view window.

For a polygon, click at each vertex of the polygon in the view window. To finish drawing the polygon, click the middle mouse button. (Press both buttons when using a two-button mouse.) The polygon is closed automatically.

Chapter 7: Generating Mesh and Doping Profiles

Defining Areas for Mesh Refinement or Doping

Note:

Ref/Eval windows are oriented. The face normals of rectangular Ref/Eval windows point in the positive third-axis direction (right-hand rule). For polygonal Ref/Eval windows, if the vertex position list is counterclockwise, then the normals point in the positive third-axis direction. The orientation is important when Ref/Eval windows are used in `sdedr:define-analytical-profile-placement` for reference. The "Positive" and "Negative" arguments of this Scheme extension must be adjusted to the Ref/Eval window normal direction.

The corresponding Scheme commands are:

```
(sdedr:define-refeval-window RefEval-name "Rectangle"
    position position)

(sdedr:define-refeval-window RefEval-name "Polygon" position-list)
```

For example:

```
(sdedr:define-refeval-window "RefEvalWin_2" "Rectangle"
    (position 0.0 0.5 0) (position 1.0 1.5 0))

(sdedr:define-refeval-window "RefEvalWin_3" "Polygon"
    (list (position 0 0 0) (position 0.5 -1 0)
        (position 1 -1 0) (position 1.5 0 0) (position 0 0 0)))
```

Rectangles and polygons defined by using the graphical user interface (GUI) are placed in the current work plane.

To create Ref/Eval windows in areas outside the current work plane, change the work plane first as discussed in [Work Planes on page 152](#).

In the 2D GUI mode, a rectangular Ref/Eval window can also be defined from certain dialog boxes by choosing, for example:

- **Mesh > Refinement Placement**
- **Mesh > Multibox Placement**
- **Device > Constant Profile Placement**
- **Device > Analytical Profile Placement**
- **Device > External Profile Placement**

Note:

When you use the GUI to define Ref/Eval windows, the name of the Ref/Eval window is assigned automatically and cannot be changed later.

Chapter 7: Generating Mesh and Doping Profiles

Defining Areas for Mesh Refinement or Doping

Cuboidal Ref/Eval Windows

Cuboidal Ref/Eval windows are used to place mesh refinement or constant profiles in three dimensions.

To define a cuboidal Ref/Eval window:

1. Choose **Mesh > Define Ref/Eval Window > Cuboid**.

The Cuboid Coordinates dialog box opens.

2. Enter the coordinates of the start corner and the end corner.

Note:

The face normals of cuboidal Ref/Eval windows point outwards.

3. Click **OK**.

The corresponding Scheme command is:

```
(sdedr:define-refeval-window RefEval-name "Cuboid" position position)
```

For example:

```
(sdedr:define-refeval-window "RefEvalWin_4" "Cuboid"  
    (position 0 0 0) (position 1 1 1))
```

In the 3D GUI mode, a cuboidal Ref/Eval window can also be defined from certain dialog boxes by choosing, for example:

- **Mesh > Refinement Placement**
- **Mesh > Multibox Placement**
- **Device > Constant Profile Placement**
- **Device > Analytical Profile Placement**
- **Device > External Profile Placement**

Note:

When you use the GUI to define Ref/Eval windows, the name of the Ref/Eval window is assigned automatically and cannot be changed later.

Extracting Ref/Eval Window From Face

You can create a Ref/Eval window that coincides with one or more faces of a 2D body for placing mesh refinement or constant profiles, or faces of a 3D body for placing implant-like doping profiles in three dimensions.

Chapter 7: Generating Mesh and Doping Profiles

Defining Areas for Mesh Refinement or Doping

To extract a Ref/Eval window from faces:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Select Face**.
3. Select a face and hold the Shift key to select additional faces, or drag a box around a set of faces.
4. Choose **Mesh > Define Ref/Eval Window > From Face**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).

The corresponding Scheme command is:

```
(extract-refwindow face | face-list RefEval-name)
```

For example:

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon"
  "region_1")
(extract-refwindow (find-face-id (position 0.5 0.5 1.0))
  "RefEval_TopFace")
```

Extracting Ref/Eval Window From Body Interface

You can use the Scheme extension `sdedr:define-body-interface-refwin` to define a refinement window between the interfaces of the specified geometric bodies (see [sdedr:define-body-interface-refwin on page 542](#)):

```
(sdedr:define-body-interface-refwin body-list ref-eval-window)
```

For example:

```
(sde:clear)
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 4)
  "Silicon" "x1"))
(define mb2 (sdegeo:create-cuboid (position 2 2 3) (position 8 8 5)
  "PolySilicon" "x2"))
(sdedr:define-body-interface-refwin (list mb1 mb2) "RW")
(sdedr:define-refinement-size "RDef_1" 1 1 1 0.1 0.1 0.1)
(sdedr:define-refinement-placement "RPl_1" "RDef_1" "RW")
(sdedr:write-cmd-file "rwext.cmd")
```

Extracting Ref/Eval Window From 3D Solid Body

You can use the Scheme extension `extract-refpolyhedron` to convert a 3D geometric body to a polyhedral refinement window. The ACIS surface mesher of Sentaurus Structure Editor, which saves the tessellated polyhedral boundary for meshing, converts the 3D geometric body (possibly with nonplanar faces) to a polyhedral Ref/Eval window.

Chapter 7: Generating Mesh and Doping Profiles

Defining Areas for Mesh Refinement or Doping

For example:

```
(sde:clear)
(define mb (sdegeo:create-cuboid (position 0 0 0) (position 1 1 1)
    "Silicon" "x1"))
(extract-refpolyhedron mb "ref1")
```

or:

```
(sde:clear)
(define mb (sdegeo:create-cylinder (position 0 0 0) (position 0 0 1)
    0.1 "Silicon" "x1"))
(extract-refpolyhedron mb "ref1")
(sdedr:define-refinement-size "RD_1" .2 .2 .1 .1)
(sdedr:define-refinement-placement "RP_1" "RD_1" "ref1")
(sdedr:write-cmd-file "xx.cmd")
```

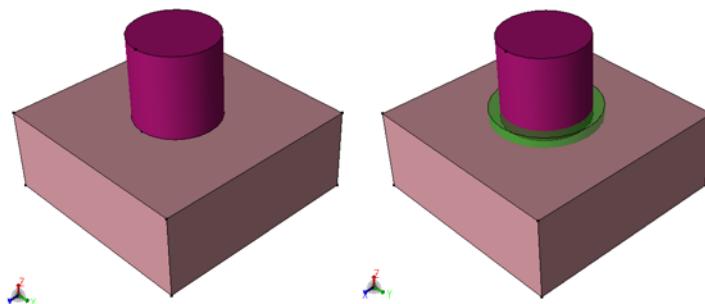
Extracting Ref/Eval Windows From 3D Solid Body Interfaces

You can use the Scheme extensions `extract-interface-offset-refwindow` and `extract-interface-normal-offset-refwindow` to extract the *non-regularized* intersection between two solid bodies, and to create a 3D offset body from the non-regularized intersection and to assign it as a doping/refinement/submesh (DRS) body. The `extract-interface-normal-offset-refwindow` extension offsets the faces of the interface body in the normal direction.

The following example creates the extracted DRS body shown in [Figure 120](#):

```
(sde:clear)
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 4)
    "Silicon" "x1"))
(define mb2 (sdegeo:create-cylinder (position 5 5 4) (position 5 5 8)
    2 "PolySilicon" "x2"))
(extract-interface-offset-refwindow mb1 mb2 1 "rwl")
```

Figure 120 Extraction of interface offset body using extract-interface-offset-refwindow: (left) initial model and (right) extracted DRS body



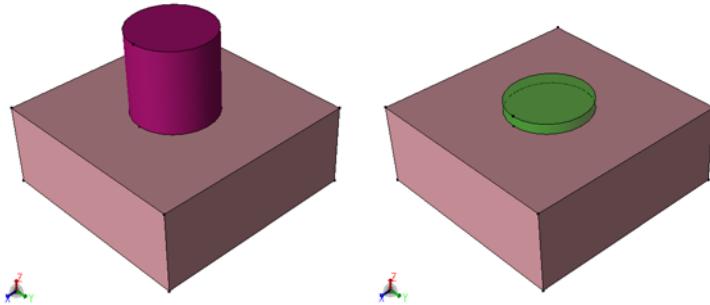
Chapter 7: Generating Mesh and Doping Profiles

Defining Areas for Mesh Refinement or Doping

The following example creates the extracted DRS body shown in [Figure 121](#):

```
(sde:clear)
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 4)
    "Silicon" "x1"))
(define mb2 (sdegeo:create-cylinder (position 5 5 4) (position 5 5 8)
    2 "PolySilicon" "x2"))
(extract-interface-normal-offset-refwindow mb1 mb2 1 "rw1")
```

Figure 121 Extraction of interface offset body using extract-interface-normal-offset-refwindow: (left) initial model and (right) extracted DRS body



Deleting Ref/Eval Windows

To delete a Ref/Eval window:

1. Click the Select button (see [Table 7 on page 46](#)).
2. From the Selection Level list, select **Ref/Eval Window**.

Alternatively, right-click in the view window and choose **Ref/Eval Window**.

3. Click inside the Ref/Eval window to be deleted or drag a box around it.
4. Choose **Mesh > Delete Ref/Eval Window**.

The corresponding Scheme command is:

```
(sdedr:delete-refeval-window RefEval-name)
```

For example:

```
(sdedr:define-refeval-window "RefEvalWin_1" "Line"
    (position 0.0 0.0 0) (position 1.0 0.0 0))
(sdedr:delete-refeval-window "RefEvalWin_1")
```

Defining Mesh Refinements

This section describes how to define mesh refinements.

Regular Mesh Refinement Boxes

A mesh refinement box allows you to request from the mesher that, in the specified area, the mesh spacing should not exceed a given value and that mesh refinement should stop if the mesh size becomes smaller than a given value. Further, you can instruct the mesher to refine the mesh in areas of steep doping gradients or near interfaces. The mesher can create mesh spacings that are smaller than the requested minimum if needed to satisfy other constraints such as the resolution at an interface or to fulfill the Delaunay mesh quality criterion.

Sentaurus Mesh uses regular mesh refinement boxes.

To create a regular mesh refinement box:

1. Choose **Mesh > Refinement Placement**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).

The Refinement Specification dialog box opens.

2. Edit the **Placement Name** field as needed (`RefinementPlacement_<index>` is the default name), or select the name of a previously defined refinement specification for editing.

3. Under Placement Type, select from the following:

- a. To place the mesh refinement into a previously defined Ref/Eval window, select the required Ref/Eval window from the **Ref/Eval Window** list.
- b. To place the mesh refinement into all regions with a given material, select a material name from the **Materials** list.
- c. To place the mesh refinement into a specific region, select a region name from the **Regions** list.
- d. Select **Define Ref/Eval Window** if a required Ref/Eval window has not yet been defined. Enter the coordinates of the start corner and end corner of the Ref/Eval window, and click **Create**.

The name of the new Ref/Eval window is assigned automatically, and the new Ref/Eval window is selected automatically in the **Ref/Eval Window** field.

A cuboidal Ref/Eval window is created if the 3D GUI mode is set (choose **View > GUI Mode > 3D Mode**). In 2D GUI mode, a rectangular Ref/Eval window is created. See [GUI Modes on page 43](#).

Chapter 7: Generating Mesh and Doping Profiles

Defining Mesh Refinements

- e. To edit a previously defined Ref/Eval window, select the Ref/Eval window from the **Ref/Eval Window** field.

Select **Define Ref/Eval Window** and edit the window coordinates as needed. Click **Modify**. (Only Ref/Eval windows consistent with the current GUI mode can be edited from this dialog box.)

4. Under Refinement Definition, edit the **Name** field as needed (the default name is `RefinementDefinition_<index>`).

You can link an existing mesh refinement definition to the current refinement placement by selecting it from the corresponding list.

5. Enter values for the required maximum and minimum element sizes in each axis direction. (Omit this step when reusing an existing refinement definition.)
6. Optionally, add refinement functions to the refinement definition by clicking **More**. In the expanded dialog box, under Refinement Functions, do any of the following:
 - To add automatic doping refinement based on the value difference or gradient of a doping profile, select **Value Difference** or **Gradient**, and select the required doping profile from the corresponding list.

Change the **Value** field to control the sensitivity of the automatic refinement if needed. Click **Add** to append the new function to the current list of refinement functions associated with the current refinement definition.

- To add automatic refinement at selected interfaces, select **Interface Length** and select the material in which the refinement should be applied from the first material list. Enter the material on the other side of the interface in the second material list.

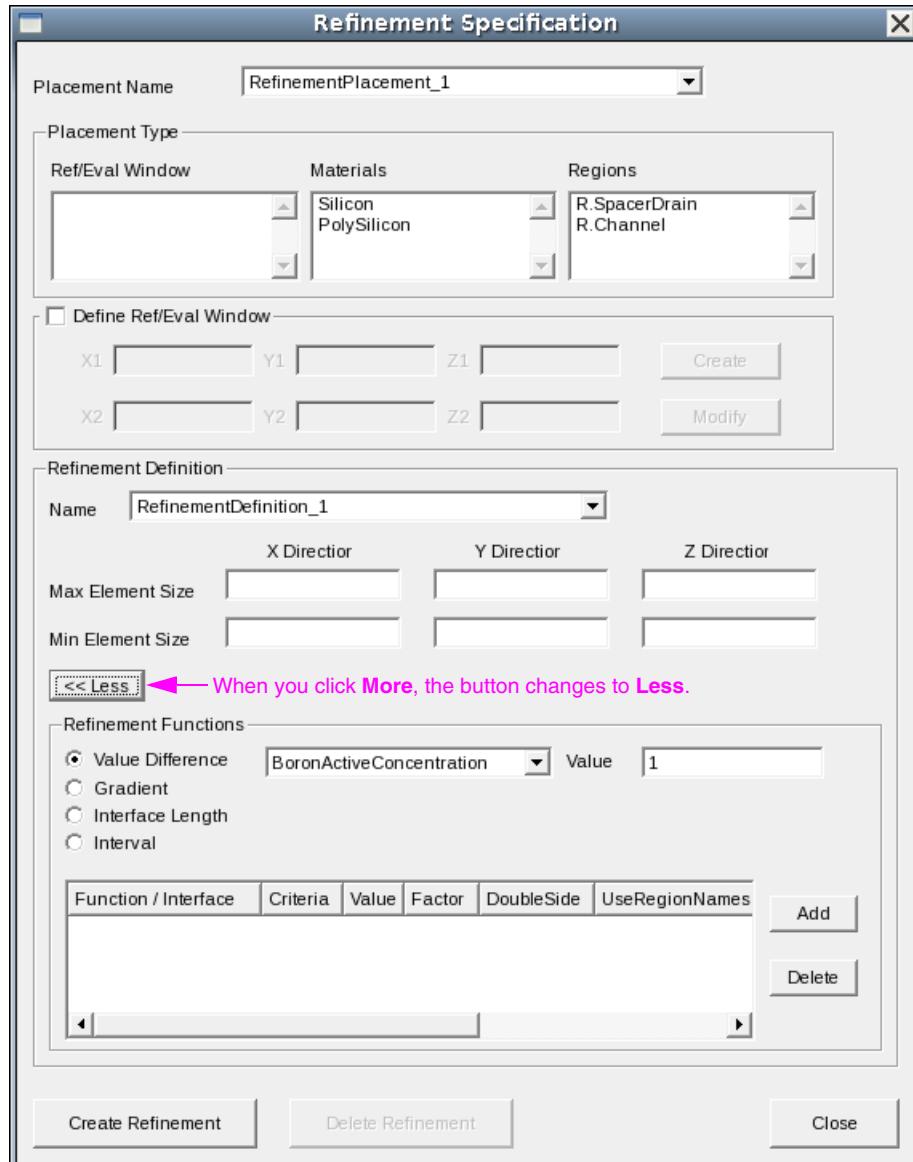
Enter the first requested mesh spacing at the interface in the **Value** field and the ratio between the second and the first requested mesh spacing in the **Factor** field. Select **DoubleSide** to apply the interface refinement to both sides of the interface.

Select **UseRegionNames** to interpret the interface as a regionwise specification.

Click **Add** to append the new function to the current list of refinement functions associated with the current refinement definition.

Chapter 7: Generating Mesh and Doping Profiles

Defining Mesh Refinements



- To add automatic refinement based on profile values with a certain interval, select **Interval** and select the profile from the list.

Enter values for **Scaling**, **Cmin**, **Cmax**, and **TargetLength**. Optionally, select **Rolloff** for the interval refinement of Sentaurus Mesh.

Note:

You can add vector and tensor values for **Cmin** and **Cmax** by entering a space-separated list of the component values.

Chapter 7: Generating Mesh and Doping Profiles

Defining Mesh Refinements

- To remove a refinement function, select it and click **Delete**.
7. When you have completed the refinement specification, click **Create Refinement**.

After a refinement is created, the **Create Refinement** button changes to **Change Refinement**, and the **Delete Refinement** button becomes available.

See *Sentaurus™ Mesh User Guide*, Defining Refinement Region, as well as *Sentaurus™ Mesh User Guide*, Placing Refinement Regions, for details about refinement functions.

The related Scheme commands for creating Ref/Eval windows are:

```
(sdedr:define-refeval-window RefEval-name "Line" | "Rectangle" | "Cuboid"
    position position)
(sdedr:define-refeval-window RefEval-name "Polygon" position-list)
```

The corresponding Scheme command for creating a refinement definition is:

```
(sdedr:define-refinement-size RefDef-name dxmax dymin [dzmax]
    dxmin dymin [dzmin])
```

where **dxmax**, **dymin**, and **dzmax** are the maximum required mesh spacings, and **dxmin**, **dymin**, and **dzmin** are the minimum required mesh spacings.

The corresponding Scheme commands for adding adaptive refinement functions are:

```
(sdedr:define-refinement-function RefDef-name dopant-name
    "MaxTransDiff" | "MaxGradient" value)

(sdedr:define-refinement-function "RefDef-name"
    "MaxLenInt" material-1 material-2 first-step ratio)

(sdedr:define-refinement-function "RefDef-name"
    "MaxInterval" "Variable" dopant-name "Cmin" cmin "Cmax" cmax
    "Scaling" scaling "TargetLength" targetLength "Rolloff")
```

The corresponding Scheme commands for linking a refinement definition to a Ref/Eval window, a region, or a material are:

```
(sdedr:define-refinement-placement RefPlace-name RefDef-name
    RefEval-name)
(sdedr:define-refinement-region RefPlace-name RefDef-name
    region-name)
(sdedr:define-refinement-material RefPlace-name RefDef-name
    material-name)
```

For example:

```
(sdegeo:create-rectangle (position -0.5 0.0 0) (position 0.5 0.5 0)
    "Silicon" "region_1")
(sdedr:define-refeval-window "RefEvalWin.Channel" "Rectangle"
    (position -0.025 0.00 0) (position 0.025 0.01 0))
(sdedr:define-refinement-size "RefDef.Channel" 5e-3 2e-3 2e-3 1e-3)
(sdedr:define-refinement-function "RefDef.Channel" "MaxLenInt")
```

Chapter 7: Generating Mesh and Doping Profiles

Defining Mesh Refinements

```
"Silicon" "Oxide" 1e-4 1.2)
(sdedr:define-refinement-function "RefDef.Channel" "MaxInterval"
    "Variable" "DopingConcentration" "Cmin" 1e-4 "Cmax" 1.2 "Scaling" 2
    "TargetLength" 1e-3 "Rolloff")
(sdedr:define-refinement-function "RefDef.Channel"
    "DopingConcentration" "MaxTransDiff" 1)
(sdedr:define-refinement-placement "RefPlace.Channel" "RefDef.Channel"
    "RefEvalWin.Channel")
```

Note:

When using interface refinement from a Scheme command, you can set the materials to "All", or to an empty string to refer to all materials, or to the ambient. This option is not available from the GUI.

Interface refinement is specified in a similar way to the refinement on analytic functions. To perform interface refinement, define a `RefineFunction` of type `MaxLengthInterface` and specify the pair of materials defining the interface, the initial thickness, and a factor used to define how this thickness should grow into the material.

The following examples illustrate the use of this function:

- Refine silicon at the oxide interface, starting with a layer of 0.02 μm and gradually increasing the thickness by 1.4 times:

```
(sdedr:define-refinement-function "RefinementDefinition_1"
    "MaxLenInt" "Silicon" "Oxide" 0.02 1.4)
```

- Refine all interfaces, creating a single layer of 0.01 μm:

```
(sdedr:define-refinement-function "RefinementDefinition_1"
    "MaxLenInt" "All" "All" 0.01)
```

- Refine all contacts in the mesh:

```
(sdedr:define-refinement-function "RefinementDefinition_1"
    "MaxLenInt" "All" "Contact" 0.01)
```

- Refine around a single contact:

```
(sdedr:define-refinement-function "RefinementDefinition_1"
    "MaxLenInt" "All" "Gate" 0.01 "UseRegionNames")
```

By default, interface refinement is performed only on the first material of the specified pair of materials describing the interface. To refine on both sides of the interface, specify the "DoubleSide" keyword:

```
(sdedr:define-refinement-function "RefinementDefinition_1" "MaxLenInt"
    "Silicon" "Oxide" 0.02 1.4 "DoubleSide")
```

Multibox Refinements

Multibox refinements are similar to regular mesh refinement boxes, but the requested minimum mesh spacing can be graded. It starts with the minimum value given at a specified side of the refinement window and is expanded by a given factor from one mesh line to the next until the given maximum is reached. Sentaurus Mesh uses multibox refinement boxes.

Note:

Consider using interface refinement in Sentaurus Mesh as a more flexible alternative to multibox refinement.

Multibox refinement applies only to rectangular (2D) and cuboidal (3D) Ref/Eval windows. Polygonal Ref/Eval windows and regionwise or materialwise placements are not supported. Moreover, refinement functions cannot be added to multibox refinements. Use additional regular mesh refinement boxes to obtain, for example, refinement at doping gradients in addition to multibox refinement.

To define a multibox refinement:

1. Choose **Mesh > Multibox Placement**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).
- The Multibox Specification dialog box opens.
2. Edit the **Placement Name** field as needed (`MultiboxPlacement_<index>` is the default name), or select the name of an existing multibox refinement specification for editing.
3. Under Placement Type, select a Ref/Eval window.

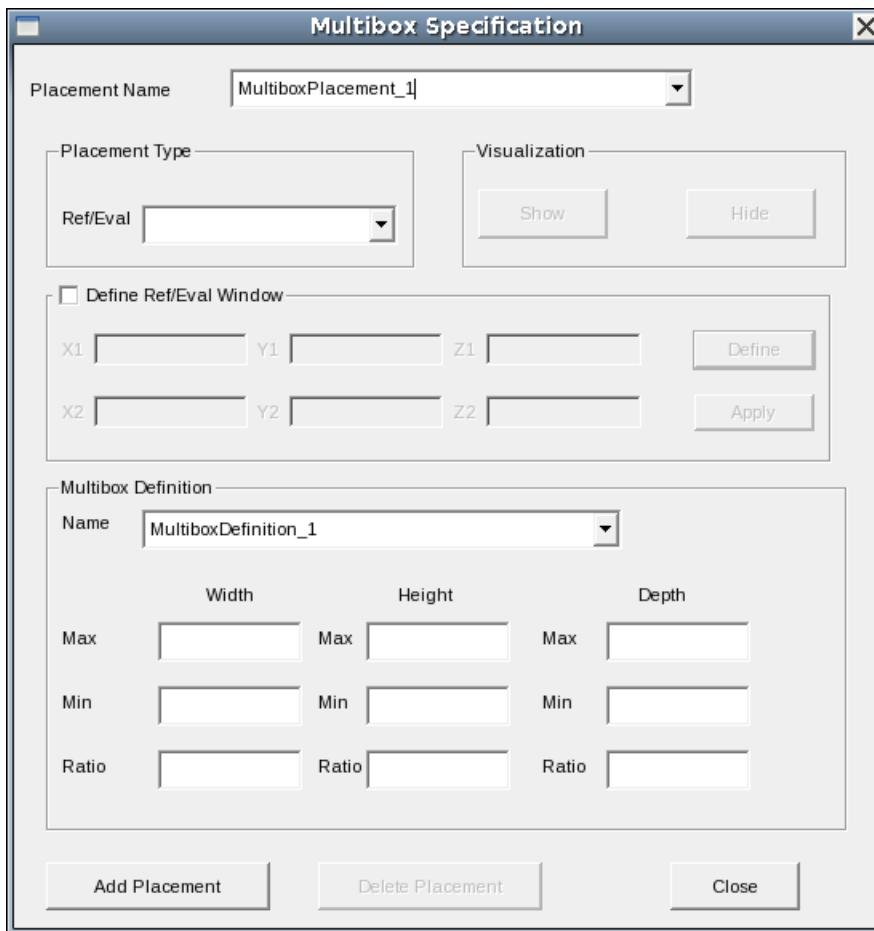
To define a new Ref/Eval window or to edit an existing Ref/Eval window, proceed as discussed in [Regular Mesh Refinement Boxes on page 245](#) in Steps 3d and 3e.

4. Under Multibox Definition, edit the **Name** field as needed (the default name is `MultiboxDefinition_<index>`).
You can link an existing refinement definition to the current multibox placement by selecting it from the corresponding list.
5. Under Multibox Definition, enter values for the required maximum and minimum element sizes in each axis direction as well as for the expansion ratio:
 - Use a positive expansion ratio to apply the minimum element size at the respective boundary with the lower coordinate value, that is, the left, front, or bottom side.
 - Use a negative expansion ratio to apply the minimum element size at the respective boundary with the higher coordinate value, that is, the right, back, or top side.
 - A ratio of 1 means the minimum element size is used throughout the multibox.
 - A ratio of 0 deactivates refinement along the corresponding axis.

Chapter 7: Generating Mesh and Doping Profiles

Defining Mesh Refinements

Omit this step when reusing an existing refinement definition.



- When you have completed the specification, click **Add Placement**.

After the new specification is added, the **Add Placement** button changes to **Change Placement**, and the **Delete Placement** button becomes available.

See *Sentaurus™ Mesh User Guide*, Defining Multibox Region, as well as *Sentaurus™ Mesh User Guide*, Placing Multibox Regions.

The Scheme extensions for creating a multibox refinement are:

```
(sdedr:define-refeval-window rfwin-name "Rectangle" | "Cuboid"
  position position)
(sdedr:define-multibox-size mbox-name max-x max-y max-z min-x min-y
  min-z ratio-x ratio-y ratio-z)
```

For example:

```
(sdegeo:create-cuboid (position 1 3 5) (position 2 4 6) "Silicon"
  "region_1")
```

Chapter 7: Generating Mesh and Doping Profiles

Defining Mesh Refinements

```
(sdedr:define-refeval-window "RefEvalWin_1" "Cuboid"
    (position 1 3 5) (position 2 4 6))
(sdedr:define-multibox-size "MultiboxDefinition_1"
    0.25 0.25 0.25
    0.05 0.05 0.05
    -2 0 1)
(sdedr:define-multibox-placement "MultiboxPlacement_1"
    "MultiboxDefinition_1" "RefEvalWin_1")
```

Offsetting Refinements

The main algorithm in Sentaurus Mesh produces meshes that are mostly axis aligned. Even when refining curved interfaces, an adapted axis-aligned mesh is used by default.

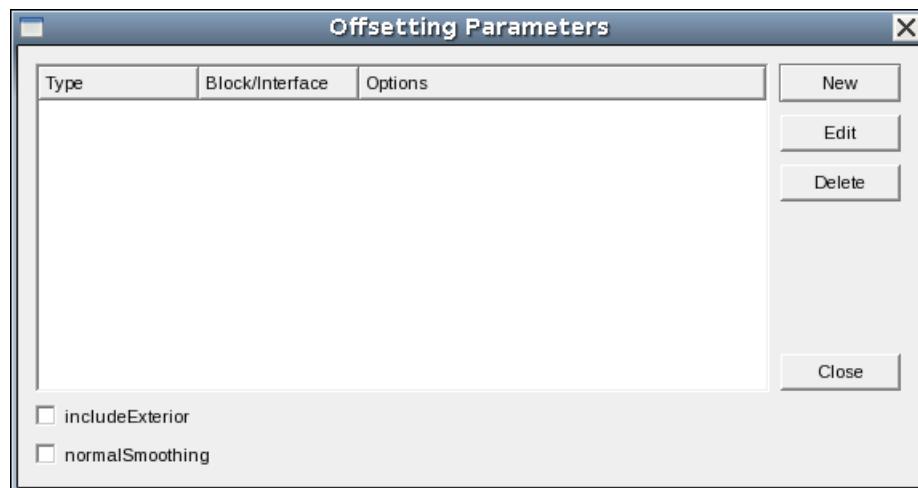
Sentaurus Mesh also provides an offsetting algorithm that resolves interfaces by introducing mesh layers truly parallel to the curved surfaces.

For devices where the main surfaces are nonaxis-aligned or curved (for example, a MOS-type structure where the channel is nonplanar), the offsetting algorithm is an attractive alternative to purely axis-aligned meshing.

To define offsetting refinements globally or for a region, a material, or an interface:

1. Choose **Mesh > Offsetting Parameters**.

The Offsetting Parameters dialog box opens.

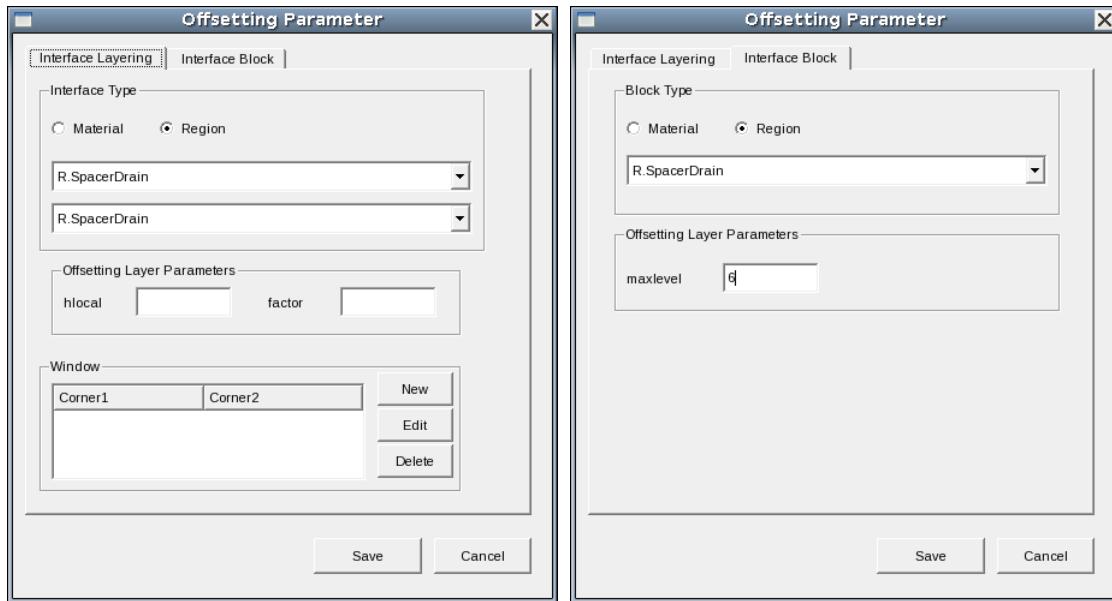


2. Click **New** to add a new offsetting refinement parameter, or select a defined parameter from the list and click **Edit**.

Chapter 7: Generating Mesh and Doping Profiles

Defining Mesh Refinements

The Offsetting Parameter dialog box opens.



3. Specify the fields on the required tabs, and then click **Save**.
4. (Optional) Select **includeExterior** to include the exterior material or region.
5. (Optional) Select **normalSmoothing** to reorient the surface normals around high curvature areas to avoid tangled surfaces during the construction of analytic layers.
6. Click **Close**.

The corresponding Scheme extensions are:

```
(sdedr:offset-block {"region" region | "material" material}
                     "maxlevel" maxlevel)

(sdedr:offset-interface
  {"region" region1 region2 | "material" material1 material2}
  "hlocal" hlocal "factor" factor "window" x1 y1 z1 x2 y2 z2)

(sdesnmesh:offsetting ["includeExterior" includeExterior]
                      ["normalSmoothing" normalSmoothing])
```

For example:

```
(sdedr:offset-block "region" "R.SiEpi" "maxlevel" 10)

(sdedr:offset-interface "region" "R.SiEpi" "R.Oxide"
                       "hlocal" 2e-4 "factor" 1.5 "window" 0.0 0.0 0.0 0.1 0.2 0.5)

(sdesnmesh:offsetting "includeExterior" #t "normalSmoothing" #t)
```

Defining Doping Profiles

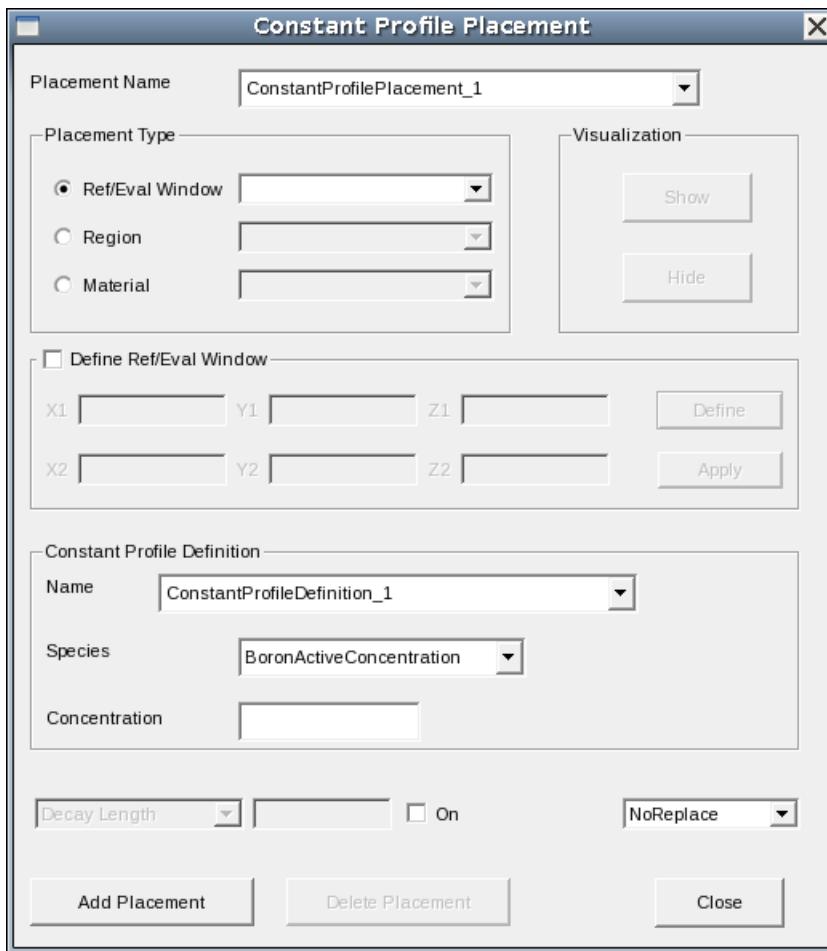
This section describes how to define doping profiles.

Defining Constant Doping Profiles

To define a constant doping profile:

1. Choose **Device > Constant Profile Placement**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).

The Constant Profile Placement dialog box opens.



2. Edit the **Placement Name** field as needed (`ConstantProfilePlacement_<index>` is the default name), or select the name of a previously defined constant profile for editing.

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

3. Under Placement Type, select a Ref/Eval window as discussed in Step 3 of [Regular Mesh Refinement Boxes on page 245](#).

4. Under Constant Profile Definition, edit the **Name** field as needed (the default name is `ConstantProfileDefinition_<index>`).

You can link an existing constant profile definition to the current constant profile placement by selecting it from the corresponding list.

5. Select a species from the list, which contains the most common dopant species. Select **Other** to access the complete list of DATEX variables. Enter a value for the concentration of the dopant.

6. To activate smoothing of the otherwise abrupt doping profile at Ref/Eval boundaries, enter a nonzero decay length in the **Decay Length** field and select **On**.

In this case, doping profiles are smoothed using an error function with an inflection length given by the specified decay length. Note that, in this case, the doping value at Ref/Eval boundaries is half of the specified concentration

For details, see *Sentaurus™ Mesh User Guide*, Placing Constant Profiles, and *Sentaurus™ Mesh User Guide*, Lateral Error Function.

7. Select the replacement option:

- Select **Replace** to replace all other previously defined profiles with this doping profile.
- Select **LocalReplace** to replace only the doping species being defined.
- Select **NoReplace** to add the new profile to all previously defined profiles.

Omit this step when reusing an existing refinement definition.

8. When you have completed the constant profile specification, click **Add Placement**.

After the specification is added, the **Add Placement** button changes to **Change Placement**, and the **Delete Placement** button becomes available.

The Scheme extensions for creating constant doping profiles are:

```
(sdedr:define-refeval-window rfwin-name "Rectangle" | "Cuboid"
    position position)

(sdeder:define-constant-profile definition-name species concentration)

(sdeder:define-constant-profile-placement placement-name
    definition-name rfwin-name [decay-length]
    ["Replace" | "LocalReplace"])

(sdeder:define-constant-profile-material placement-name definition-name
    material-name [decay-length] ["Replace" | "LocalReplace"])
```

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

```
(sdedr:define-constant-profile-region placement-name definition-name  
region-name [decay-length] ["Replace" | "LocalReplace"])
```

For example:

```
(sdedr:define-constant-profile "Const.BG" "BoronActiveConcentration"  
1e7)  
(sdedr:define-constant-profile-material "PlaceCD.BG" "Const.BG" "GaAs")  
(sdedr:define-constant-profile "Const.GaAsCap"  
"ArsenicActiveConcentration" 3e18)  
(sdedr:define-constant-profile-region "PlaceCD.GaAsCap" "Const.GaAsCap"  
"R.GaAsCap" 2 "Replace")
```

Defining Analytic Doping Profiles

To define an analytic doping profile:

1. Choose **Device > Analytic Profile Placement**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).
- The Analytical Profiles dialog box opens.
2. Edit the **Placement Name** field as needed (`AnalyticalProfilePlacement_<index>` is the default name), or select the name of a previously defined analytic profile for editing.
3. Select a previously defined Ref/Eval window from the **Ref/Eval Window** list.
4. Under Profile Definition, edit the **Name** field as needed (the default name is `AnalyticalProfileDefinition_<index>`).

You can link an existing analytic profile definition to the current analytic profile placement by selecting it from the corresponding list.

5. Under Profile Definition, select a species from the list, which contains the most common dopant species. Select **Other** to access the complete list of DATEX variables.
6. Under Profile Definition, from the **Profile Type** list, select the profile type along the primary direction (orthogonal to the baseline, that is, a Ref/Eval window) from the following options:
 - **Gaussian:** Under Primary Direction Profile (Gauss), select **Peak Concentration** or **Dose**, and enter the value of the peak concentration in units of cm^{-3} or the dose in units of cm^{-2} .

Enter the distance of the peak position to the baseline (Ref/Eval window) in μm .

Define the broadening of the profile by selecting **Standard Deviation** or **Diffusion Length** from the list, and entering the corresponding distance in μm .

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

Alternatively, define the broadening by selecting **Junction** and entering the concentration at junctions in units of cm^{-3} and the requested distance from the peak position to the junction in μm . In this case, the standard deviation of the Gaussian profile is computed automatically.

- **Error Function:** Under Primary Direction Profile (Gauss), select **Max Concentration** or **Dose**, and enter the value of the maximum concentration in units of cm^{-3} or the dose in units of cm^{-2} .

Enter the distance of the inflection point to the baseline (Ref/Eval window) in μm .

Define the broadening of the profile by selecting **Standard Deviation** or **Diffusion Length** from the list, and entering the corresponding distance in μm .

Alternatively, define the broadening by selecting **Junction** and entering the concentration at junctions in units of cm^{-3} and the requested distance from the peak position to the junction in units of μm . In this case, the broadening parameter of the error function profile is computed automatically.

- **1D Profile:** Enter the relative path and the file name of an external 1D profile in **.plx** format. Click **Browser** to open a file browser.

You can restrict the external 1D profile to a certain range. Enter the starting 1D coordinate of the external 1D profile in the **From** field and the ending 1D coordinate in the **To** field. In the **Units** field, declare the unit of length in which the external 1D profile coordinates are given. (For example, if the coordinate range in the external 1D profile is -300 nm to 10000 nm , and the baseline should be aligned with a coordinate of 500 nm , and noisy tails beyond 3000 nm should be ignored, enter 500 in the **From** field and 3000 in the **To** field, and select **Nanometer** in the **Units** field.) In the **Data Scale** field, enter the data scaling factor.

For a 1D profile, the species is read from the **.plx** data file and the selection in the **Species** field is ignored.

- **Analytic Function:** Enter a formula in the **Function** field. For example:
`a*sin(x)*cos(y)`

Enter the initialization definition of any constants used in the formula in the **Initialization** field. For example: `a=2*10^18`

Select the coordinate system to use for evaluating the analytic formula. Select **General** if the x-, y-, and z-coordinates of the formula are to be interpreted in the coordinate system of the device. Select **Eval** if x refers to the primary direction, and y and z refer to the lateral directions.

7. Under Lateral Direction Diffusion, select the profile type along the lateral direction (parallel to the baseline, that is, the Ref/Eval window) as **Gaussian** or **Error Function**.

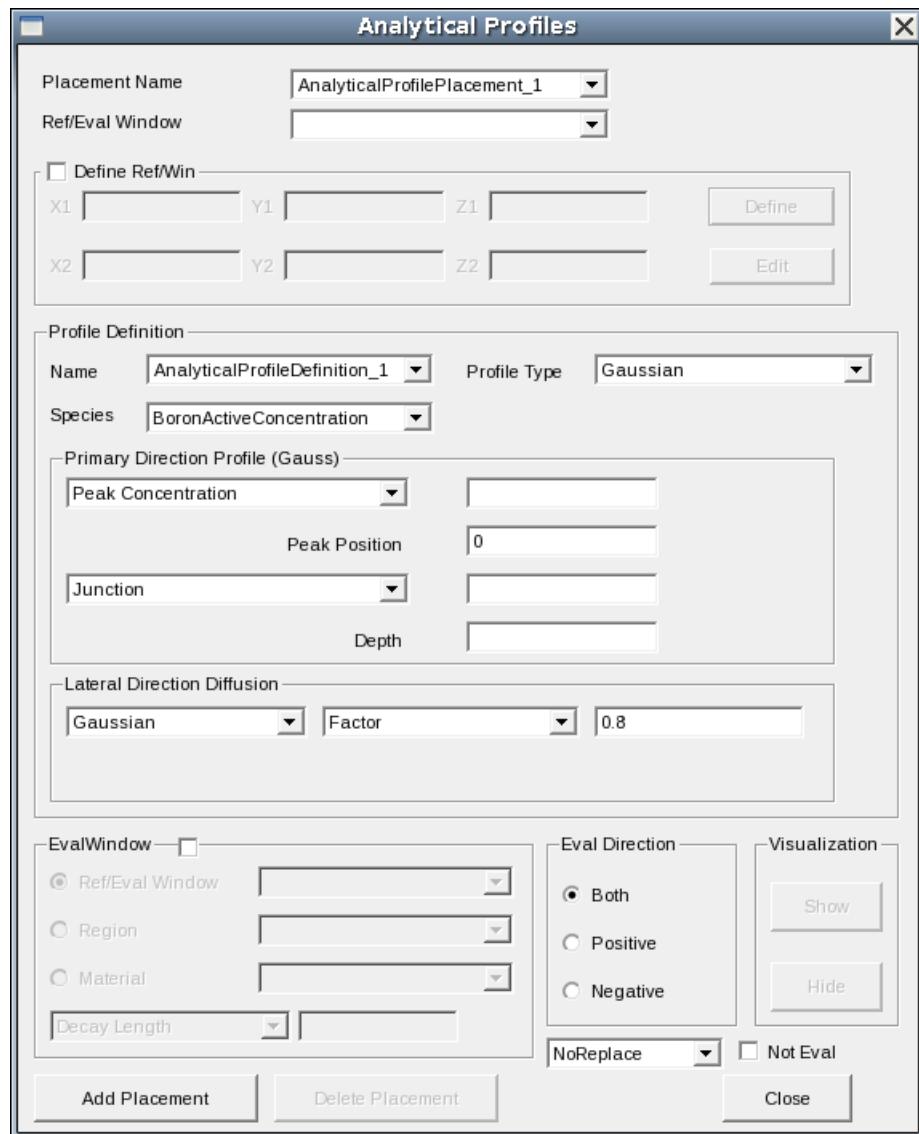
Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

Select from the list how the lateral-broadening parameter is defined:

- **Factor** computes the lateral-broadening parameter by multiplying the primary-broadening parameter with a given factor.
- **Standard Deviation or Diffusion Length** defines the broadening parameter explicitly. Enter the respective value in the corresponding field.

8. To restrict the evaluation of the analytic profile to only one side of the baseline, under Eval Direction, select **Positive** or **Negative**.



In two dimensions, for a positive evaluation direction, the profile is evaluated only to the right side of the baseline vector. In three dimensions, the profile is evaluated only along

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

the positive direction of the baseline face normal. Select **Both** to evaluate the profile on both sides.

9. If needed, select the **EvalWindow** option to restrict the evaluation of the analytic profile further to a certain area. Profiles can be restricted to another Ref/Eval window by selecting **Ref/Win** and then the name of the Ref/Eval window, or to a region or a material, by selecting **Region** or **Material**, and selecting the region or material.
10. To activate smoothing at Ref/Eval window boundaries, enter a nonzero decay length in the **Decay Length** field. In this case, the doping profiles are smoothed using an error function with an inflection length given by the specified decay length.

In this case, the doping value at Ref/Eval window boundaries is half of the specified concentration.

For details, see *Sentaurus™ Mesh User Guide*, Placing Constant Profiles, and *Sentaurus™ Mesh User Guide*, Lateral Error Function.

11. Select the replacement option:

- Select **Replace** to replace all other previously defined profiles with this doping profile.
- Select **LocalReplace** to replace only the doping species being defined.
- Select **NoReplace** to add the new profile to all previously defined profiles.

12. Select **Not Eval** to suppress the evaluation of the analytic profile at the baseline itself. Use this option, for example, to avoid double-counting at region interfaces, where two different analytic profiles of the same species meet.

13. When you have completed the specification, click **Add Placement**.

After the specification is added, the **Add Placement** button changes to **Change Placement**, and the **Delete Placement** button becomes available.

The Scheme extensions for creating analytic doping profiles are as follows.

Gaussian profile definition:

```
(sdedr:define-gaussian-profile definition-name species
  "PeakPos" peak-position
  {"PeakVal" peak-concentration | "Dose" dose}
  {"ValueAtDepth" concentration-at-depth "Depth" depth |
   "Length" diffusion-length | "StdDev" standard-deviation}
  [{"Gauss" | "Erf" "Factor" factor} | [{"Eval" "init-string"
   "function-string"}])
```

Error function profile definition:

```
(sdedr:define-erf-profile definition-name species
  "SymPos" inflection-point
  {"MaxVal" max-concentration | "Dose" dose})
```

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

```
{"ValueAtDepth" concentration-at-depth "Depth" depth |
"Length" diffusion-length | "StdDev" standard-deviation}
["Gauss"|"Erf" "Factor" factor] | ["Eval" "init-string"
"function-string"])
```

External 1D profile definition:

```
(sdedr:define-1d-external-profile definition-name plx-file-name
"Scale" coordinate-scaling-factor-to-um
"DataScale" data-file-scaling-factor "Range" range-start range-end
["Gauss"|"Erf" "Factor" factor] | ["Eval" "init-string"
"function-string"])
```

Analytic formula definition:

```
(sdedr:define-analytical-profile definition-name species init formula
default-value ["general"])
```

Placement of the analytic profile:

```
(sdedr:define-analytical-profile-placement placement-name
definition-name RefEval-name {"Both" | "Positive" | "Negative"}
{"Replace" | "NoReplace" | "LocalReplace"}
{"Eval" | "NoEval"}
[RefEval-name decay-length "evalwin" |
region-name decay-length "region" |
material-name decay-length "material"])
```

For example, to place a Gaussian profile:

```
; - Halo implantation
; -- baseline definitions
(sdedr:define-refeval-window "BaseLine.Halo" "Line"
(position 0.02 0.0 0) (position 0.40 0.0 0))
; -- implant definition
(sdedr:define-gaussian-profile "Impl.Haloprof"
"BoronActiveConcentration"
"PeakPos" 0 "PeakVal" 1.5e18
"ValueAtDepth" 1.5e17 "Depth" 0.07
"Gauss" "Factor" 1.0)
; -- implant placement
(sdedr:define-analytical-profile-placement "Impl.Halo" "Impl.Haloprof"
"BaseLine.Halo" "Positive" "NoReplace" "Eval")
```

For example, to place a profile given by an analytic formula:

```
; Linear mole fraction profile 0.16 at base-col to 0 at base-emitter
(sdedr:define-refeval-window "BaseLine.Mole" "Line"
(position -0.45 0.5 0.0) (position 1.25 0.5 0.0))
(sdedr:define-analytical-profile "Prof.Mole" "xMoleFraction"
"a=0.16/0.075" "0.1*a*x" 0.1)
(sdedr:define-analytical-profile-placement "Place.Mole" "Prof.Mole"
"BaseLine.Mole" "Positive" "NoReplace" "Eval" "RWindow.BC" 0
"evalwin")
```

Including External 2D and 3D Doping Profiles

You can include external 2D or 3D doping profiles in a device structure. Such profiles are generated typically by process simulators, such as Sentaurus Process. Often the extent of the external doping profile matches the device geometry, which is the case when Sentaurus Structure Editor is used to simplify and mesh a structure generated by Sentaurus Process before it is passed to Sentaurus Device.

However, external doping profiles do not have to match the device geometry. Sentaurus Mesh extends automatically the doping profile to fill the evaluation window or cuts out the part that lies outside of the evaluation window. The alignment of the external doping profile with the evaluation window can also be specified. Furthermore, profiles can be reflected or rotated.

In addition, if a 2D profile is included in a 3D device structure, then the profile is extended automatically. If the global coordinates of the 2D profile itself lie already in the correct cross section of the 3D device structure, then no user intervention is needed for the profile extension. Otherwise, an appropriate transformation must be applied.

To include an external 2D or 3D doping profile in a device structure:

1. Choose **Device > External Profile Placement**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).

The External Profile Placement dialog box opens.

2. Edit the **Placement Name** field as needed (`ExternalProfilePlacement_<index>` is the default name), or select the name of an existing external profile specification for editing.
3. Under External Profile Definition, edit the **Name** field as needed (the default name is `ExternalProfileDefinition_<index>`).

You can link an existing external profile definition to the current external profile placement by selecting it from the corresponding list.

4. In the **Geometry File** field, enter the name of the TDR file that contains the required external profile. Click **Browser** to open a file browser.
5. Under Data Files, select the data files to be defined, and select a mode for the files.

When you are finished defining a file, click **Add**.

6. Under Species, select the doping you want to add. Then, enter the **Species Factor**, and click **Add**.

Continue to add species and species factors as required.

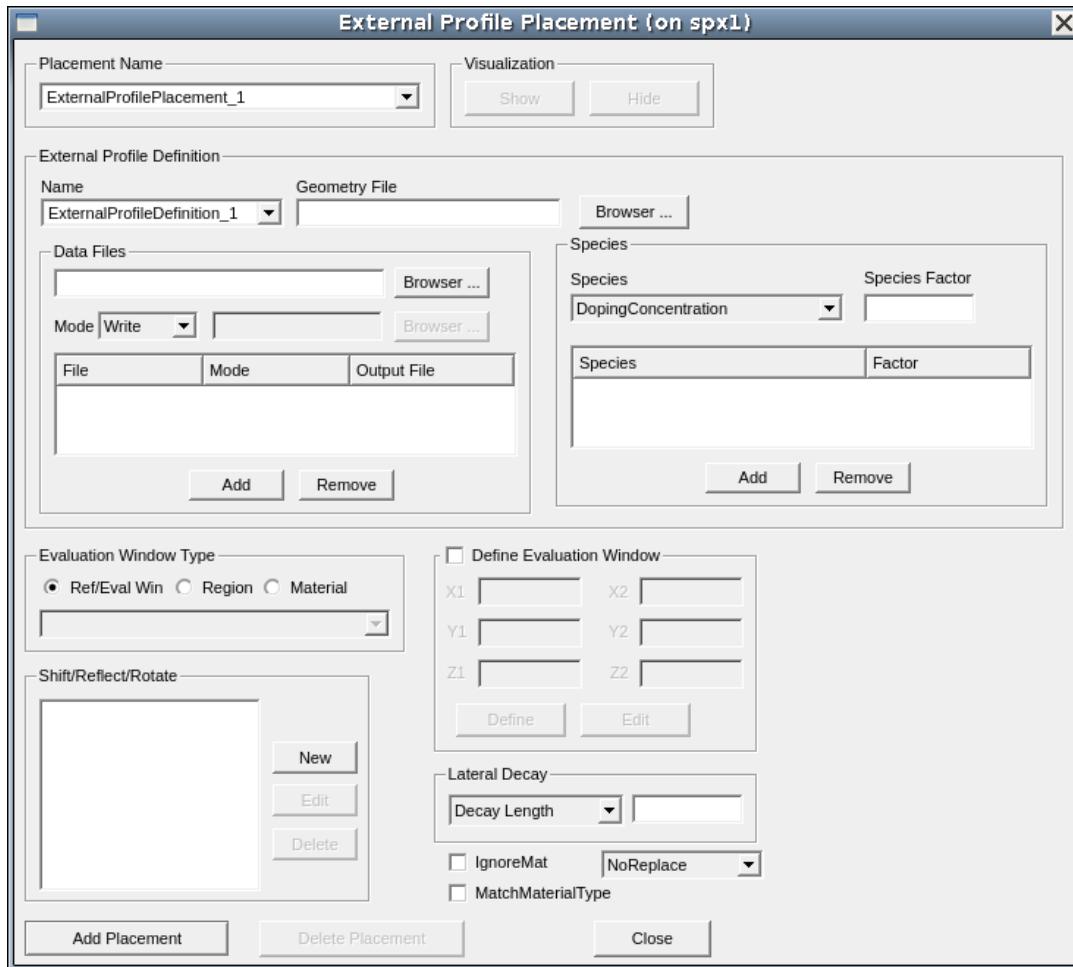
7. Under Evaluation Window Type, select **Ref/Eval Win**, **Region**, or **Material** as discussed in Step 3 of [Regular Mesh Refinement Boxes on page 245](#).

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

To place the external profile in the entire device, select **Ref/Eval Win** and leave the field empty.

8. To define an evaluation window, select **Define Evaluation Window**. Enter the coordinates for the opposite vertices of the evaluation window, and click **Define**.
9. Under Lateral Decay, to activate smoothing at Ref/Eval window boundaries, define a nonzero decay length in the field provided. Select **Decay Length** to use an error function or **Gauss Decay Length** to use a Gaussian function.



10. Select the replacement option:

- Select **Replace** to replace all other previously defined profiles with this doping profile.
- Select **LocalReplace** to replace only the doping species being defined.
- Select **NoReplace** to add the new profile to all previously defined profiles.

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

11. Under Shift/Reflect/Rotate, define the geometric transformations of translation, reflection, and rotation:
 - a. Click **New** to create a transformation operation.
The dialog box opens.
 - b. Select **Shift**, **Reflect**, or **Rotation** and enter the required values to characterize the operation, and then click **Create**.
The new operation appears in the list.
 - c. To add more operations after, at least, one has been defined, click **Prepend** or **Append**. To place the new operation within the list instead of at the beginning or end of the list, select one of the operations in the list, and click **Prepend** or **Append**.
 - d. To modify an operation, select an operation in the list and click **Edit**.
 - e. To delete an operation, select an operation in the list and click **Delete**.
 - f. To reorder the operations in the list, select one operation and drag it to a new location in the list.

Note:

Sentaurus Structure Editor respects the order of transformation operations. Operations are passed to Sentaurus Mesh in the order they appear in the list.

12. When you are finished, click **Add Placement**.

After the external profile is added, the **Add Placement** button changes to **Change Placement**, and the **Delete Placement** button becomes available.

The Scheme extensions for defining and placing external profiles are:

```
(sdedr:define-submesh definition-name TDR-file-name)

(sdedr:define-submesh-placement placement-name definition-name
  rfwin-name [ "PlacementType" { "region" | "material" | "evalwin" } ]
  { [ "DecayLength" decay-length ] |
    [ "GaussDecayLength" gauss-decay-length ] }
  [ "Replace" | "NoReplace" | "LocalReplace" ] [ "IgnoreMat" ]
  [ "MatchMaterialType" ])
```

For details, see *Sentaurus™ Mesh User Guide*, Defining Submeshes.

In addition, for transformation operations, the following Scheme extension appends transformations to the named submesh placement. Any number of transformations can be appended in one call of this Scheme extension, which can be called as many times as necessary. Transformations are appended and applied in the specified order:

```
(sdedr:transform-submesh-placement placement-name
  [ <transform1> <transform2> ... <transformN> ])
```

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

Here, <transformX> is any one of these transformations:

- "Reflect" axis
- "Rotation" angle axis
- "ShiftVector" (gvector <coord> <coord> <coord>)

where:

- axis can be "X", "Y", or "Z".
- angle, coord are floating-point numbers for the angle and for the x-, y-, and z-coordinates.

Examples

Clear all transformations defined for the named submesh placement:

```
(sdedr:clear-submesh-placement-transform "SubmeshPlacementName")
```

The following 2D examples use a common set of Scheme extensions to create a simplified device structure and to define a meshing strategy:

```
(sde:clear)
(sdgeo:create-rectangle (position -0.445 0.0 0) (position 0.0 0.5 0)
    "Silicon" "region_L")
(sdgeo:create-rectangle (position 0.0 0.0 0) (position 0.445 0.5 0)
    "Silicon" "region_R")
(sdedr:define-refinement-size "RefinementDefinition_1" 0.25 0.5 0.001
    0.001)
(sdedr:define-refinement-material "RefinementPlacement_1"
    "RefinementDefinition_1" "Silicon")
(sdedr:define-refinement-function "RefinementDefinition_1"
    "DopingConcentration" "MaxTransDiff" 1)
```

Starting with the previous Scheme script, the following examples present different variations of including an external doping profile.

Include an external doping profile called n34_fps.tdr in the entire device structure:

```
(sdedr:define-submesh "ExternalProfileDefinition_1" "n34_fps.tdr")
(sdedr:define-submesh-placement "ExternalProfilePlacement_1"
    "ExternalProfileDefinition_1" "" "NoReplace")
```

Note:

The empty string before "NoReplace" indicates that the profile will be included in the entire device structure. The profile is extended to regions of the device structure that are not covered by the external profile.

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

Restrict the inclusion of the external doping profile to a Ref/Eval window and reflect the external profile using a mirror plane that goes through the bounding box center of the external profile and is orthogonal to the x-axis:

```
(sdedr:define-submesh "ExternalProfileDefinition_1" "n34_fps.tdr")
(sdedr:define-refeval-window "RefEvalWin_1" "Rectangle"
    (position 0.0 0.0 0) (position 0.445 0.4 0))
(sdedr:define-submesh-placement "ExternalProfilePlacement_1"
    "ExternalProfileDefinition_1" "RefEvalWin_1" "NoReplace")
(sdedr:transform-submesh-placement "ExternalProfilePlacement_1"
    "Reflect" "X")
```

Shift the external profile by 0.445 μm to the left, and include it in all silicon regions:

```
(sdedr:define-submesh "ExternalProfileDefinition_1" "n34_fps.tdr")
(sdedr:define-submesh-placement "ExternalProfilePlacement_1"
    "ExternalProfileDefinition_1" "Silicon" "PlacementType" "Material"
    "NoReplace")
(sdedr:transform-submesh-placement "ExternalProfileDefinition_1"
    "ShiftVector" (gvector -0.445 0.0 0.0))
```

Reflect and shift the external profile by 0.445 μm to the left, and include it only in a given region:

```
(sdedr:define-submesh "ExternalProfileDefinition_1" "n34_fps.tdr")
(sdedr:define-submesh-placement "ExternalProfilePlacement_1"
    "ExternalProfileDefinition_1" "region_L" "PlacementType" "Region"
    "NoReplace")
(sdedr:transform-submesh-placement "ExternalProfileDefinition_1"
    "ShiftVector" (gvector 0.445 0 0) "Reflect" "X")
```

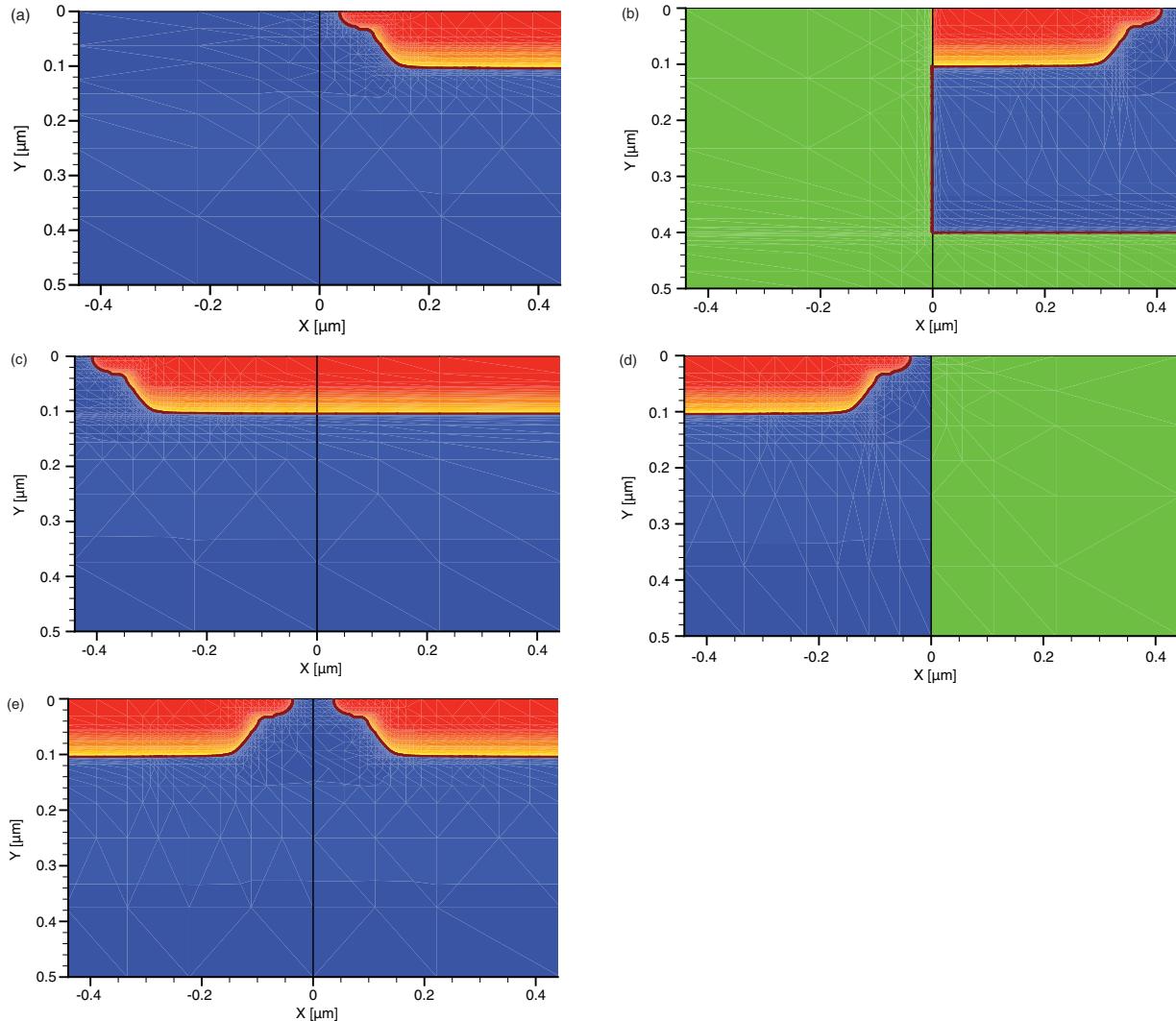
Include the same profile twice, once at its original location, and once reflected and shifted:

```
(sdedr:define-submesh "ExternalProfileDefinition_1" "n34_fps.tdr")
(sdedr:define-submesh-placement "ExternalProfilePlacement_1"
    "ExternalProfileDefinition_1" "region_R" "PlacementType" "Region")
(sdedr:define-submesh-placement "ExternalProfilePlacement_2"
    "ExternalProfileDefinition_1" "region_L" "PlacementType" "Region"
    "NoReplace")
(sdedr:transform-submesh-placement "ExternalProfilePlacement_2"
    "ShiftVector" (gvector 0.445 0 0) "Reflect" "X")
```

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

Figure 122 Including external doping profiles using different options: (a) simple inclusion; (b) reflections and Ref/Eval window; (c) shift; (d) reflect, shift, and region restriction; and (e) using the same profile twice



The following example illustrates how to include a 2D external doping profile in a simple 3D structure. Both examples use a common set of Scheme extensions to create a simplified device structure and to define a meshing strategy:

```
(sde:clear)
(sdegeo:create-cuboid (position 0.0 0.0 0.0)
    (position 0.445 0.445 0.445) "Silicon" "region_p1")
(sdedr:define-refinement-size "RefinementDefinition_1"
    0.2    0.2    0.2
    0.004  0.004  0.004)
(sdedr:define-refinement-material "RefinementPlacement_1"
```

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

```
"RefinementDefinition_1" "Silicon")
(sdedr:define-refinement-function "RefinementDefinition_1"
    "DopingConcentration" "MaxTransDiff" 1)
```

Include the profile without any alignment:

```
(sdedr:define-submesh "ExternalProfileDefinition_1" "n34_fps.tdr")
(sdedr:define-submesh-placement "ExternalProfilePlacement_1"
    "ExternalProfileDefinition_1" "Silicon" "PlacementType" "Material"
    "NoReplace")
```

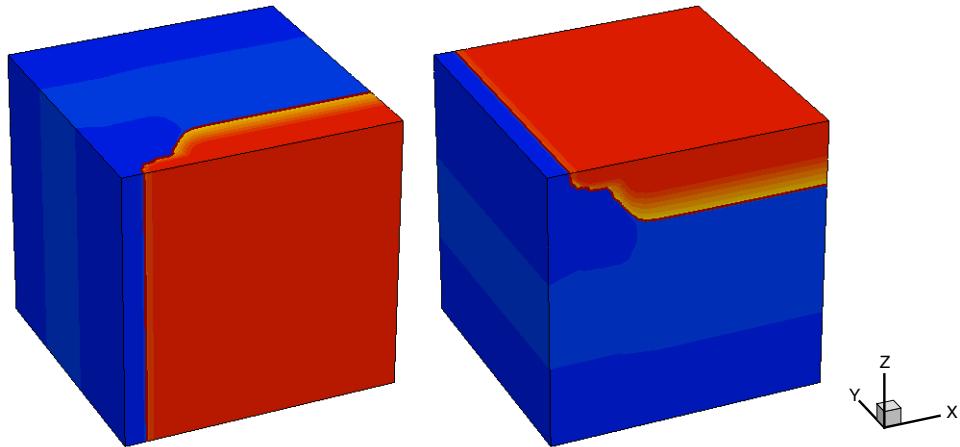
Note:

The 2D profile lies in the xy plane. The simple inclusion results in an extrusion of the profile along the z-axis.

Rotate the profile into the xz plane before inclusion:

```
(sdedr:define-submesh "ExternalProfileDefinition_1" "n34_fps.tdr")
(sdedr:define-submesh-placement "ExternalProfilePlacement_1"
    "ExternalProfileDefinition_1" "Silicon" "PlacementType" "Material"
    "NoReplace")
(sdedr:transform-submesh-placement "ExternalProfilePlacement_1"
    "Rotation" -90 "X" "ShiftVector" (gvector 0 0 0.445))
```

Figure 123 Including a 2D external doping profile in a 3D structure: (left) by default the profile lies in the xy plane and (right) using shifts and rotations to place the profile in the xz plane

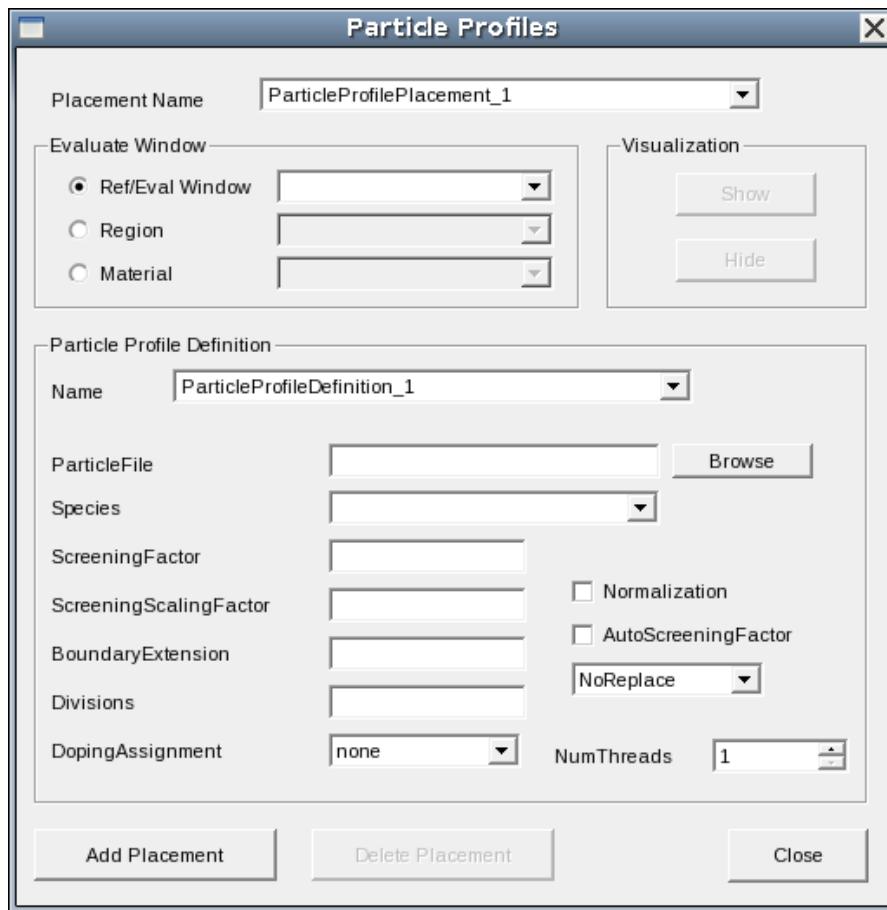


Defining Particle Doping Profiles

To define a particle doping profile:

1. Choose **Device > Particle Profile Placement**, or click the corresponding toolbar button (see [Table 14 on page 49](#)).

The Particle Profiles dialog box opens.



2. Edit the **Placement Name** field as needed (`ParticleProfilePlacement_<index>` is the default name), or select the name of an existing particle profile specification for editing.
3. Under Evaluate Window, select **Ref/ Win** and a Ref/Eval window as discussed in Step 3 of [Regular Mesh Refinement Boxes on page 245](#).
4. Under Particle Profile Definition, edit the **Name** field as needed (the default name is `ParticleProfileDefinition_<index>`).

Chapter 7: Generating Mesh and Doping Profiles

Defining Doping Profiles

You can link an existing particle profile definition to the current particle profile placement by selecting it from the corresponding list.

5. Edit the **ParticleFile** field, or click **Browse** to select the file.
 6. Optionally, from the **Species** list, select a species that contains the most common dopant species:
 - Select **Other** to access the complete list of DATEX variables.
 - Select **blank** at the end of the list so as not to use **Species**. The **blank** species is the default.
 7. Optionally, enter values for **ScreeningFactor**, **ScreeningScalingFactor**, **BoundaryExtension**, and **Divisions**. If required, select **Normalization** and **AutoScreeningFactor**.
- For details, see *Sentaurus™ Mesh User Guide*, Placing Constant Profiles, and *Sentaurus™ Mesh User Guide*, Defining Particle Profiles.
8. Select the replacement option:
 - Select **Replace** to replace all other previously defined profiles with this doping profile.
 - Select **LocalReplace** to replace only the doping species being defined.
 - Select **NoReplace** to add the new profile to all previously defined profiles.Omit this step when reusing an existing refinement definition.
 9. When you have finished, click **Add Placement**.

After the specification is added, the **Add Placement** button changes to **Change Placement**, and the **Delete Placement** button becomes available.

The Scheme extensions for creating particle doping profiles are [sdedr:define-particle-profile on page 552](#), [sdedr:define-particle-profile-placement on page 553](#), and [sdedr:define-refeval-window on page 554](#).

For example:

```
(sdedr:define-refeval-window "RefEvalWin_1" "Rectangle"
  (position -1 -0.5 0) (position -0.5 0 0))
(sdedr:define-particle-profile "ParticleProfileDefinition_1"
  "particlefile"
  "Species" "BoronActiveConcentration" "ScreeningFactor" 1
  "BoundaryExtension" 5 "Divisions" 10 "AutoScreeningFactor" #t
  "Normalization" #t "NumberOfThreads" 4 "DopingAssignment" "CIC")
(sdedr:define-particle-profile-placement "ParticleProfilePlacement_1"
  "ParticleProfileDefinition_1" "EvalWindow" "RefEvalWin_1" "evalwin"
  "Replace" #t)
```

Controlling the Boundary Tessellation

This section discusses boundary tessellation.

Global Tessellation Settings

Internally, Sentaurus Structure Editor uses a representation of geometric bodies that can include curved boundaries, for example, regions with circular or spline boundary edges in two dimensions, and regions with cylindrical or spherical boundary faces. In addition, certain sweep and revolve operations lead to curved boundaries.

Other TCAD Sentaurus tools, however, work exclusively with polyhedral boundaries. When exporting a device boundary, Sentaurus Structure Editor approximates all boundaries by a sequence of straight line segments (in two dimensions) or by patches of flat triangles (in three dimensions). This process is called *tessellation*. To save the tessellated boundary, use the `sdeio:save-tdr-bnd` Scheme extension (see [sdeio:save-tdr-bnd on page 745](#)) or choose **File > Save Boundary As** (or **Save Boundary**).

Converting the curved boundary to a polyhedral patch is not unique, and you can control the tessellation process.

Available Faceters

Two faceters are available to tessellate the internal (2D or 3D, possibly curved) model geometry. If a TDR boundary is saved using the `sdeio:save-tdr-bnd` Scheme extension, then you can use the "faceter" argument to select the faceter from either "v1" (default) or "v2".

In some cases, the "v2" faceter provides better quality tessellation, especially in three dimensions, when curved spline faces are present in the native model geometry.

The control parameters of *both faceters* can be set by using the `sdeio:save-tdr-bnd` Scheme extension. The faceters are controlled by the following:

- **Surface tolerance** refers to the allowable maximum deviation (distance, in global coordinates) between the internal curved representation and the tessellated output.
- **Normal tolerance** is the maximum difference (in degrees) that is allowed between the surface normals of the internal curved representation and the face normals of the tessellated output.
- **Aspect ratio** is the preferred aspect ratio of the boundary tessellation triangles.
- **Maximum edge length** is the maximum-allowed edge length in the tessellated output.

Chapter 7: Generating Mesh and Doping Profiles

Controlling the Boundary Tessellation

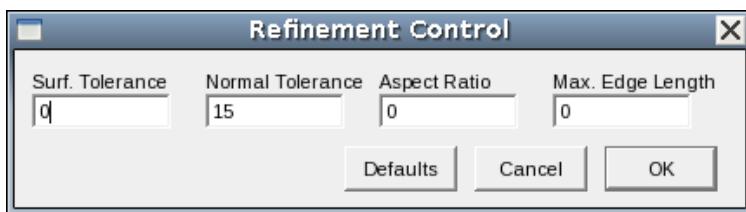
If the exported boundary contains an excessive number of elements and meshing takes a long time, then the values of the surface and normal tolerances might need to be relaxed. However, if important geometric features of the structure are not resolved in the exported boundary, then the surface tolerance and normal tolerance values might need to be decreased.

In addition to controlling the "v1" faceter by using `sdeio:save-tdr-bnd`, you can control the "v1" faceter by a global refinement setting.

To set the parameters for the "v1" faceter:

1. Choose **Mesh > Global Refinement**.

The Refinement Control dialog box opens.



2. Enter the values for the surface tolerance, the normal tolerance, the aspect ratio, and the maximum edge length.
3. Click **OK**.

The corresponding Scheme extension is:

```
(sde:setrefprops surface-tolerance normal-tolerance  
[aspect-ratio max-edge-length])
```

Examples

To follow the examples here, activate the facets rendering mode by choosing **View > Render > Facets** or clicking the corresponding toolbar button (see [Table 5 on page 46](#)).

Effect of Surface Tolerance

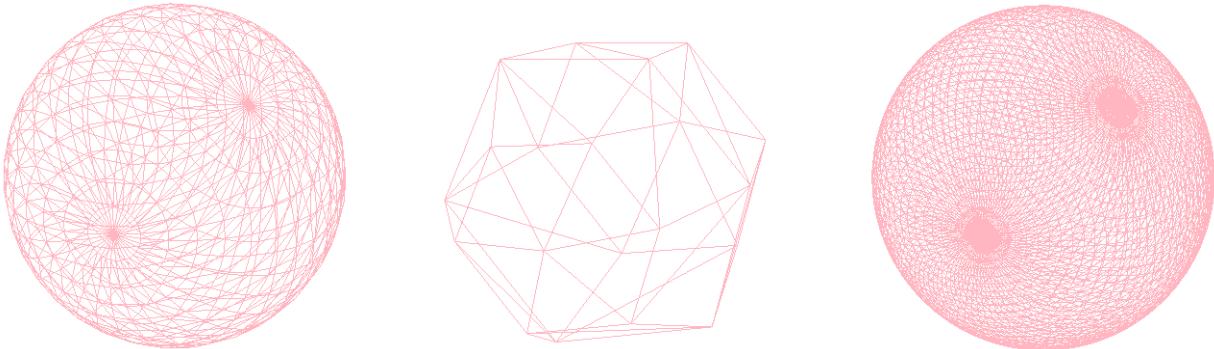
The following example illustrates the effect of surface tolerance:

```
(sde:clear)  
(sdegeo:create-sphere (position 0.0 0.0 0.0) 1.0 "Silicon" "region_1")  
; (a) Default settings:  
(sde:setrefprops -1 15 0 0)  
; (b) Relax all settings:  
(sde:setrefprops 0 90 0 0)  
; (c) Tight surface tolerance  
(sde:setrefprops 0.001 90 0 0)
```

Chapter 7: Generating Mesh and Doping Profiles

Controlling the Boundary Tessellation

Figure 124 Controlling surface tessellation: (left) default settings, (middle) relaxed settings, and (right) tight surface tolerance only

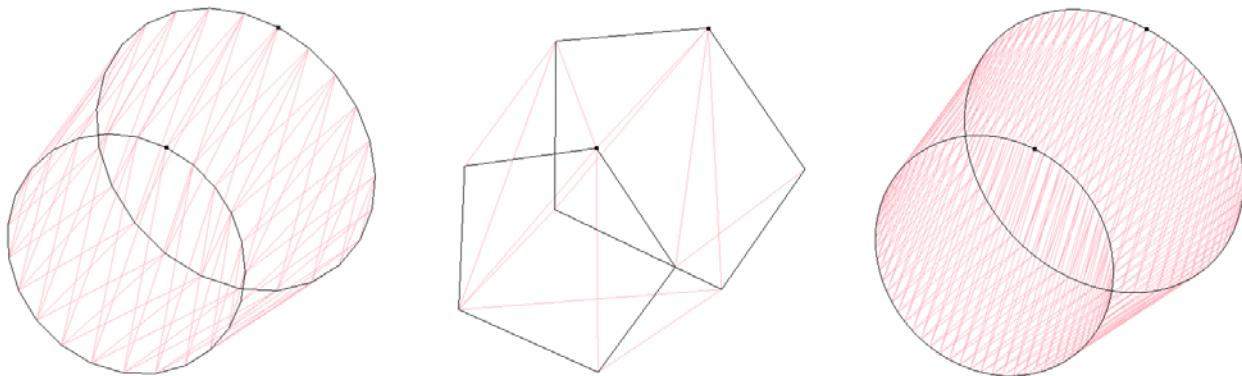


Effect of Normal Tolerance

This example illustrates the effect of normal tolerance:

```
(sde:clear)
(sdegeo:create-cylinder (position 0.0 0.0 -1.0) (position 0.0 0.0 1.0)
    1.0 "Silicon" "region_2")
; (a) Default settings:
(sde:setrefprops -1 15 0 0)
; (b) Relax all settings:
(sde:setrefprops 0 90 0 0)
; (c) Tight normal tolerance
(sde:setrefprops 0 5 0 0)
```

Figure 125 Controlling surface tessellation: (left) default settings, (middle) relaxed settings, and (right) tight normal tolerance only



Tighter surface and normal tolerances create more facets. Both parameters work effectively in a similar way, but use a slightly different criterion.

Chapter 7: Generating Mesh and Doping Profiles

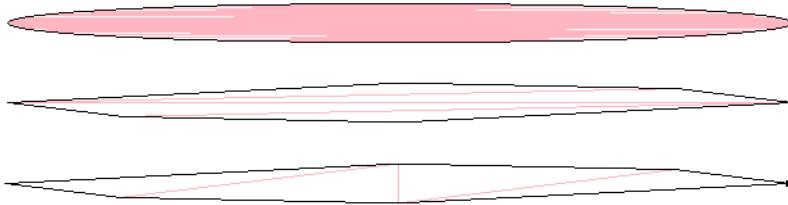
Controlling the Boundary Tessellation

Effect of the Aspect Ratio

The following example illustrates the effect of the aspect ratio:

```
(sde:clear)
(sdegeo:create-ellipse (position 0.0 0.0 0.0) (position 1.0 0.0 0.0)
  0.05 "Silicon" "region_1")
; (a) Default settings:
(sde:setrefprops -1 15 0 0)
; (b) Relax all settings:
(sde:setrefprops 0 90 0 0)
; (c) Aspect ratio
(sde:setrefprops 0 90 1.0 0)
```

Figure 126 Controlling surface tessellation: (top) default settings, (middle) relaxed settings, and (bottom) tight aspect ratio only



The aspect ratio setting does not create new nodes (2D) or faces (3D). Instead, an aspect ratio setting of 1 favors a tessellation for which the surface triangles are closer to equilateral by flipping internal edges.

Effect of Maximum Edge Length

This example illustrates the effect of the maximum edge length:

```
(sde:clear)
(sdegeo:create-cuboid (position -1.0 -1.0 -1.0) (position 1.0 1.0 1.0)
  "Silicon" "region_3")
; (a) Default settings:
(sde:setrefprops -1 15 0 0)
; (b) Tight normal tolerance
(sde:setrefprops 0 90 0 0.5)
```

For the specification of the surface tolerance, certain values have a predefined meaning. For example, a value of zero deactivates this refinement criterion. While positive values such as 0.01 refer to an absolute distance in the current units (for example, 0.01 µm), certain negative values define the tolerance distance as a fraction of the bounding box. A negative surface tolerance means that the tolerance depends on the size of the diagonal of the bounding box; the absolute value of the negative number is multiplied by a fraction (1/500) of the diagonal of the bounding box.

Chapter 7: Generating Mesh and Doping Profiles

Controlling the Boundary Tessellation

Figure 127 Controlling surface tessellation: (left) default settings and (right) maximum edge length only

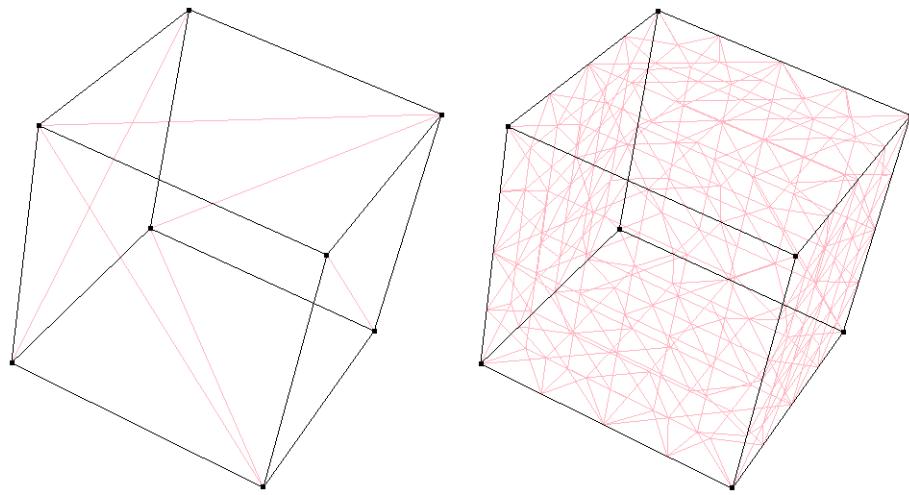


Table 31 Predefined values to specify the required surface tolerance

Surface tolerance value	Definition
0	Switches off or ignores the surface tolerance refinement criteria
-0.1	1/5000th of the box
-1	1/500th
-10	1/50th
...	...

Tessellating Spherical Faces

The default "v1" faceter triangulates spherical faces in such a way that a pole is not visible in the tessellated surface. This minimizes the vertex connectivity at each vertex.

By default, in the `sdeio:save-tdr-bnd` Scheme extension, the argument "grid mode" is set to "AF_GRID_ONE_DIR" (see [sdeio:read-tdr-bnd on page 742](#)).

For example:

```
(sde:clear)
(sdegeo:create-cuboid (position -1.0 -1.0 -1.0) (position 1.0 1.0 1.0)
  "Silicon" "region_1")
(sdegeo:fillet (list
  (car (find-edge-id (position 0 1 1))))
```

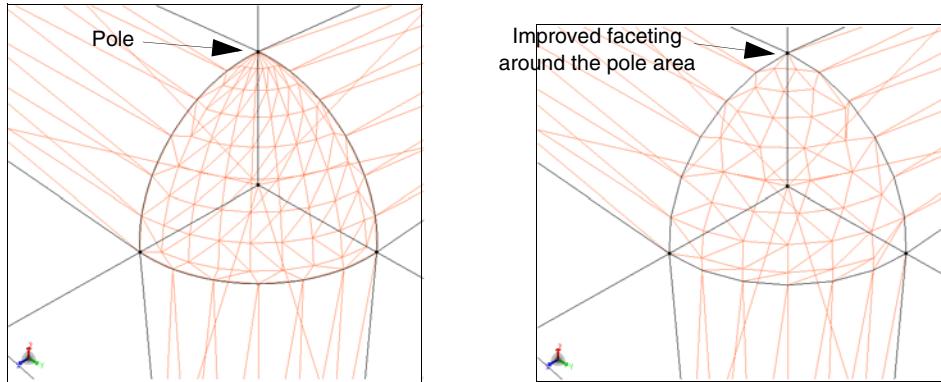
Chapter 7: Generating Mesh and Doping Profiles

Controlling the Boundary Tessellation

```
(car (find-edge-id (position 1 1 0)))
  (car (find-edge-id (position 1 0 1))) 0.4)
(sdeio:save-tdr-bnd (get-body-list) "spherical-face-tessellation.tdr")
```

Figure 128 illustrates the effect of the tessellation algorithm.

Figure 128 Tessellating spherical faces: visible pole with (left) "grid mode" "AF_GRID_TO_EDGES" and (right) "grid mode" "AF_GRID_ONE_DIR"



Boundary Tessellation in Three Dimensions

Sentaurus Structure Editor creates the TDR boundary output by using the *faceter component*. During model generation, Sentaurus Structure Editor creates a native model with solid bodies that can have nonplanar faces (for example, procedural faces such as spherical, cylindrical, toroidal, or spline faces). The default faceter component can tessellate these faces (generating a polyhedral approximation of the native model) that is written to the TDR boundary output.

The quality of the tessellation is very important, since the meshing engines use the tessellated boundary to create a volume mesh. If the tessellation quality is not good (sliver faces for example), then the meshing engines might be unable to create a volume mesh.

The tessellated TDR boundary can be saved using the `sdeio:save-tdr-bnd` Scheme extension.

Note:

If the `(option:set "hoops_direct_render" #t)` command is used, then the GUI shows the faceted model that will be written to the TDR boundary output. You can use the command `(render:rebuild)`, or right-click and choose **Show All**, to refresh the screen and to show the current facetting.

Defining Command File Sections Specific to Sentaurus Mesh

This section discusses command file sections specific to Sentaurus Mesh.

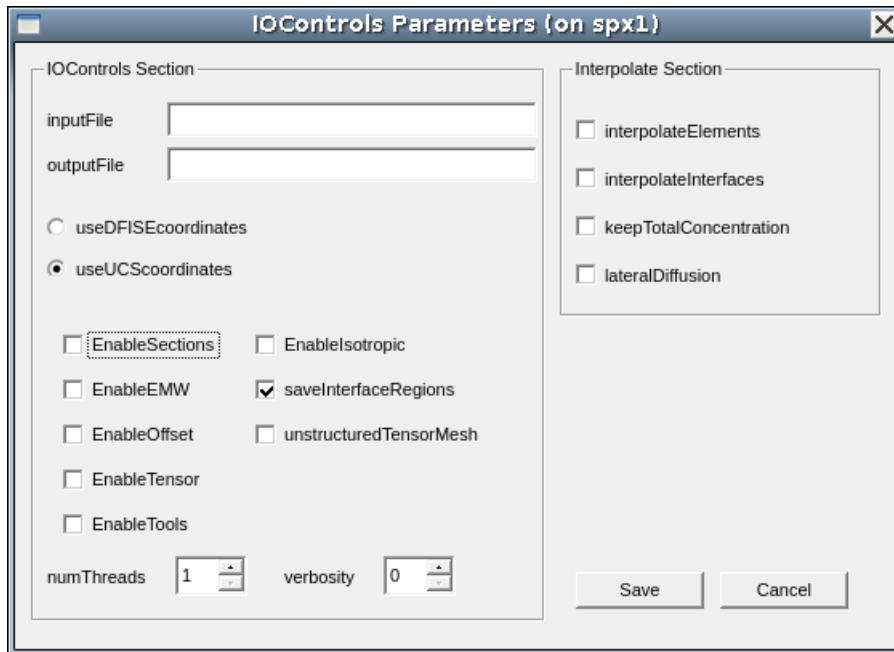
IOControls Section and Interpolate Section

You can define the parameters of the `IOControls` section and the `Interpolate` section by using the GUI or Scheme extensions (see [sdesnmesh:iocontrols on page 780](#) and [sdesnmesh:interpolate on page 779](#)).

To define the parameters of the `IOControls` section:

1. Choose **Mesh > IOControls and Interpolate Parameters**.

The `IOControls` Parameters dialog box opens.



2. Under `IOControls` Section, enter values for `inputFile`, or `outputFile`, or both in the corresponding boxes.

Note:

If you specify `inputFile`, then it overrides the base name of the saved boundary file when you create the mesh by either choosing **Mesh > Build Mesh** (see [Building the Mesh on page 286](#)) or using the `sde:build-mesh` Scheme extension (see [sde:build-mesh on page 438](#)).

See *Sentaurus™ Mesh User Guide*, IOControls Section, for descriptions of parameters.

3. Select other options as required.

Note:

By default, **EnableSections** is selected, which means the option `EnableSections` is added to the `IOControls` section of the mesh command file by default.

4. Under Interpolate Section, select options as required.

See *Sentaurus™ Mesh User Guide*, Interpolate Section, for descriptions of parameters.

5. Click **Save**.

AxisAligned Section

By default, the bisectional refinement algorithm of Sentaurus Mesh starts from the bounding box of the entire device structure. After all volume refinement criteria have been satisfied, interfaces are introduced. This can result in mesh spacing near the interface that is less than the requested mesh spacing.

The axis-aligned algorithm used in Sentaurus Mesh can partition the global bounding box into an array of subboxes. The bisectional refinement algorithm then operates independently in each partitioned boxes.

This feature can reduce mesh-induced numeric noise in parametric structures. For example, for the simulation of the threshold voltage roll-off characteristics of a given technology, you need to simulate the electrical characteristics of a set of transistors with different gate lengths, but identical source and drain areas.

Changing the gate length changes the mesh in the channel area. However, by placing partitioning lines between the gate and the source as well as drain areas, you ensure the mesh in the source and drain areas remains the same for all devices.

These partitioning lines (2D) or planes (3D) are defined using the `xCuts`, `yCuts`, and `zCuts` parameters of the `AxisAligned` section.

To define the parameters of the `AxisAligned` section:

1. Choose **Mesh > Axis-Aligned Parameters**.

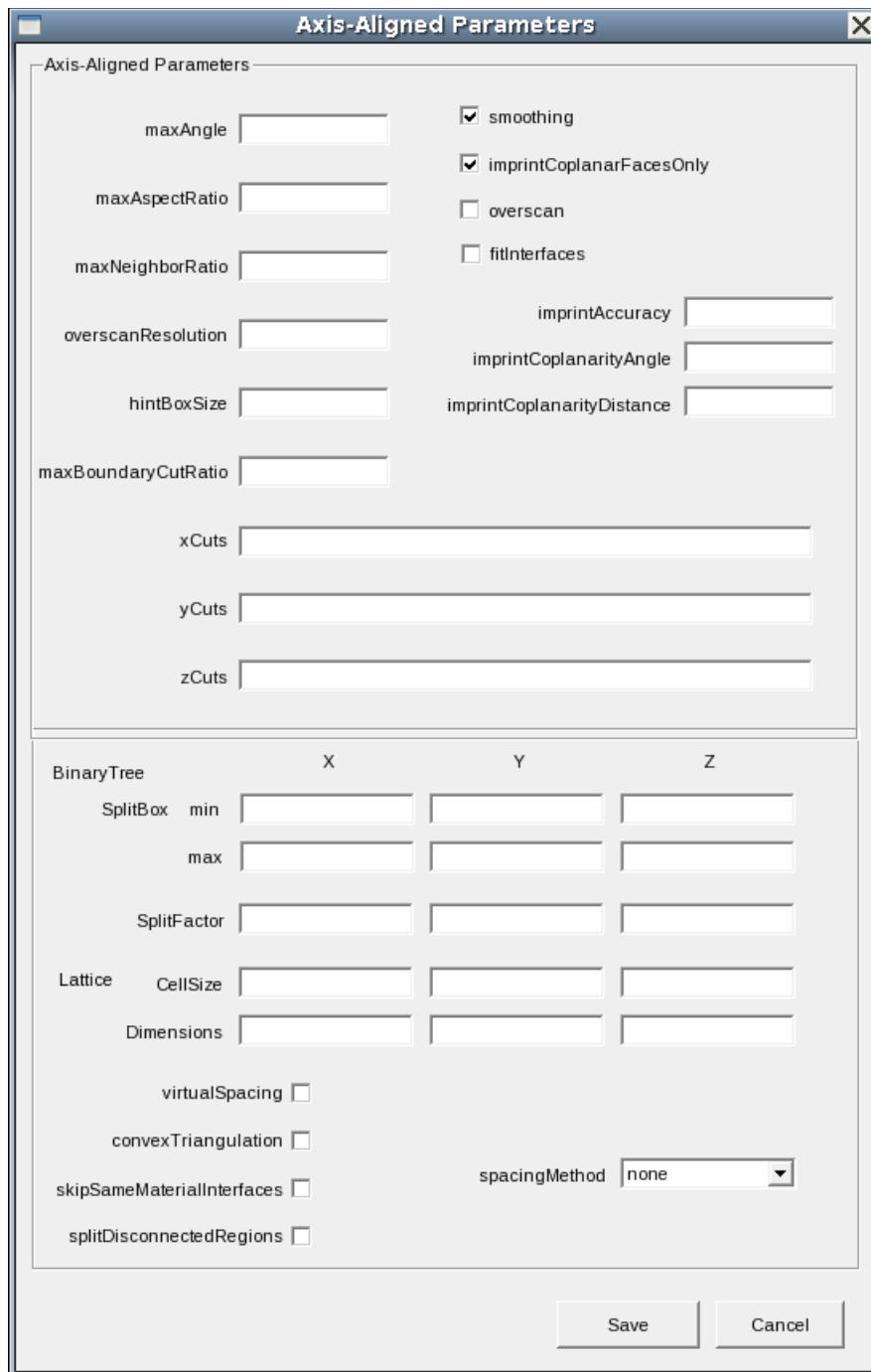
The Axis-Aligned Parameters dialog box opens.

2. Enter values for parameters, and select options as required.

Chapter 7: Generating Mesh and Doping Profiles

Defining Command File Sections Specific to Sentaurus Mesh

See *Sentaurus™ Mesh User Guide*, AxisAligned Section, for descriptions of parameters.



3. Click **Save**.

The corresponding Scheme extension is [sdesnmesh:axisaligned](#) on page 771. For example:

```
(sdesnmesh:axisaligned "yCuts" (list 0.1 0.5 2.0))
```

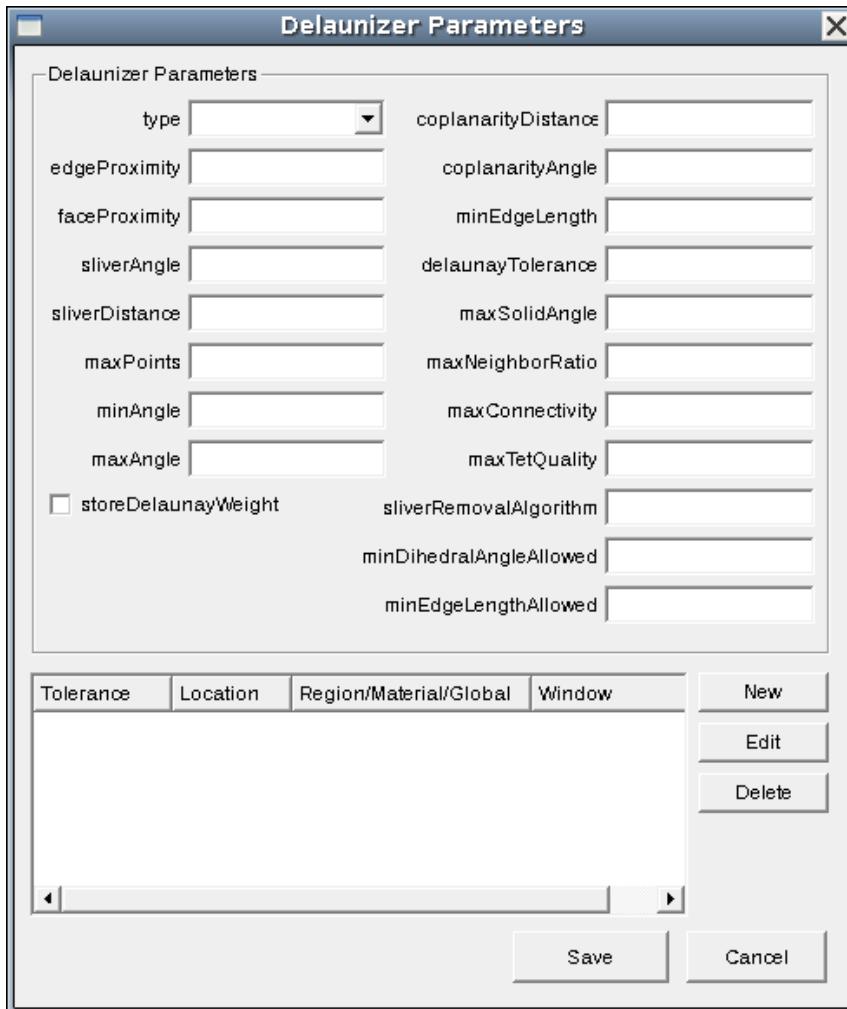
Delaunizer Section

The **Delaunizer** section controls the behavior of the delaunization algorithms used in Sentaurus Mesh.

To define the parameters of the **Delaunizer** section:

1. Choose **Mesh > Delaunizer Parameters**.

The **Delaunizer Parameters** dialog box opens.



Chapter 7: Generating Mesh and Doping Profiles

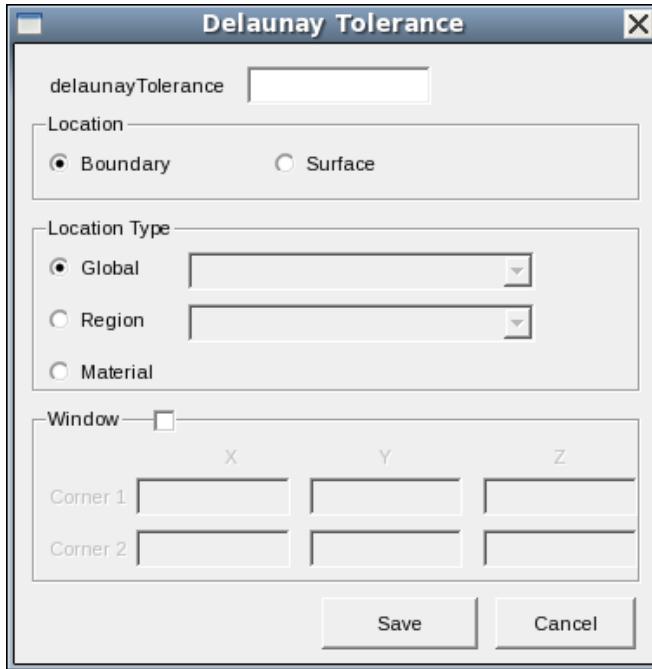
Defining Command File Sections Specific to Sentaurus Mesh

2. Enter values for parameters as required.

See *Sentaurus™ Mesh User Guide*, Delaunizer Section, for descriptions of parameters.

3. Click **New**.

The Delaunay Tolerance dialog box opens.



4. Enter values for parameters, and select options as required.
5. Click **Save** to close the Delaunay Tolerance dialog box.
6. Click **Save**.

The corresponding Scheme extensions are [sdesnmesh:delaunizer](#) on page 776 and [sdesnmesh:delaunizer-tolerance](#) on page 778. For example:

```
(sdesnmesh:delaunizer-tolerance 1 "boundary" "global")
(sdesnmesh:delaunizer-tolerance 0.0001 "boundary" "material" "Oxide"
    "Silicon")
(sdesnmesh:delaunizer "type" "boxmethod" "maxPoints" 100000
    "coplanarityDistance" 1e-05 "edgeProximity" 0.05
    "faceProximity" 0.05 "sliverAngle" 175 "sliverDistance" 0.01
    "maxSolidAngle" 360 "maxNeighborRatio" 1e+30
    "coplanarityAngle" 175 "minEdgeLength" 1e-09
    "delaunayTolerance" 0.0001 "storeDelaunayWeight" #t)
```

Tensor Section

To tensor mesh refinements:

1. Choose **Mesh > Tensor Parameters**.

The Tensor Block Editor opens.

2. Edit the `Tensor` section of the mesh command file as required.

For details, see *Sentaurus™ Mesh User Guide*, Tensor Section.

3. Click **Save**.

The corresponding Scheme extension is:

```
(sdesnmesh:tensor "tensor-block-contents")
```

where `"tensor-block-contents"` is a string surrounded by double quotation marks that represents the required contents of the `Tensor` section of the mesh command file.

You must escape double quotation marks with a backslash within `"tensor-block-contents"`. For example:

```
(sdesnmesh:tensor "Box {  
    name = \"TensorBox1\"  
    material = \"Silicon\"  
    startPoint = (0 0 0)  
    endPoint = (1.0 1.5 1.5)  
    tolerance = 0.25  
})
```

Note:

The previous `Tensor` section Scheme extensions are deprecated and cannot be used with `sdesnmesh:tensor` at the same time.

Boundary Section

The `Boundary` section allows you to control the boundary-processing algorithms available in Sentaurus Mesh.

To define the `Boundary` section of the mesh command file:

1. Choose **Mesh > Boundary Operators**.

The Mesh Boundary Operations opens.

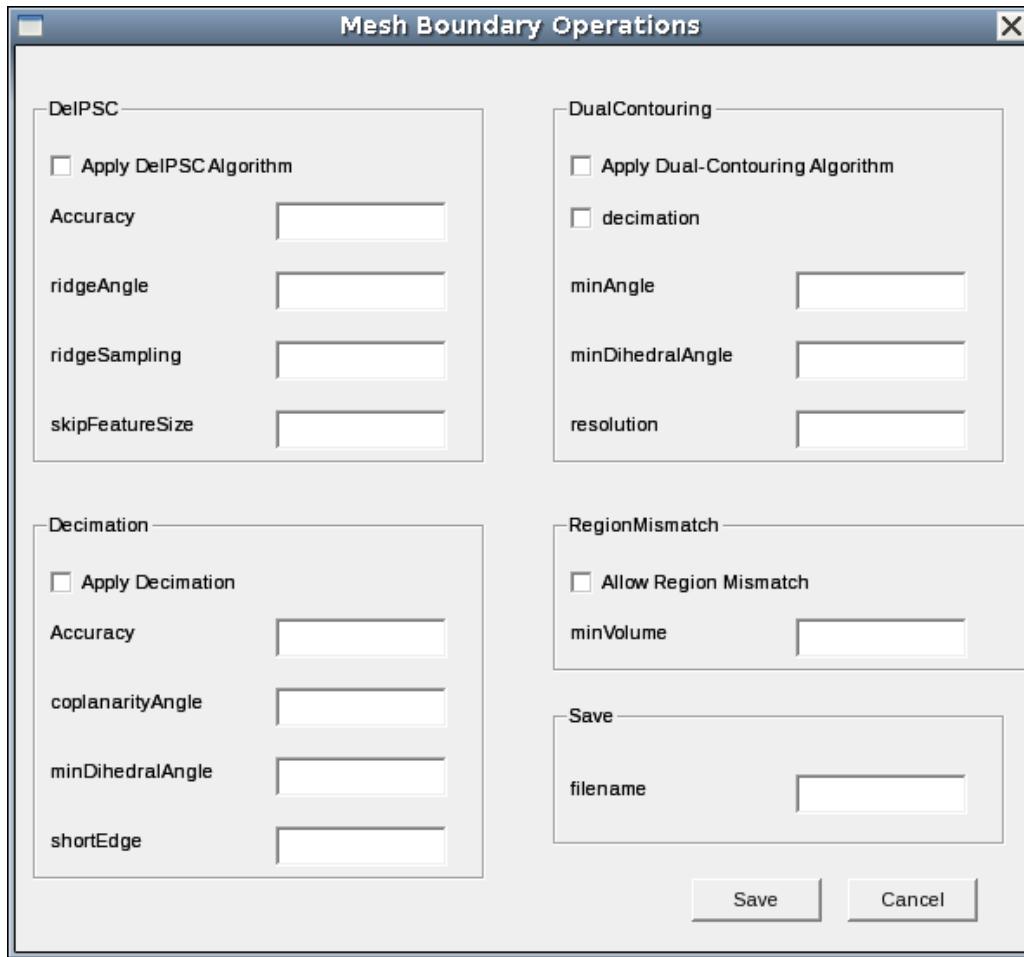
2. Select the required operations, and specify the parameters for those operations.

See *Sentaurus™ Mesh User Guide*, Boundary Section, for descriptions of parameters.

Chapter 7: Generating Mesh and Doping Profiles

Defining Command File Sections Specific to Sentaurus Mesh

3. (Optional) Under RegionMismatch, select **Allow Region Mismatch** and enter the value for **minVolume** for the mismatch if you want Sentaurus Mesh to continue when a small region is lost during the meshing process.



4. (Optional) Under Save, enter a file name if you want to save the modified boundary after the operations are performed.
5. Click **Save**.

The corresponding Scheme extension is [sdesnmesh:boundary](#) on page 774. For example:

```
(sdesnmesh:boundary "DelPSC" "accuracy" 0.001)
```

Boundary Section

The Boundary section allows you to control the boundary-processing algorithms available in Sentaurus Mesh.

Chapter 7: Generating Mesh and Doping Profiles

Defining Command File Sections Specific to Sentaurus Mesh

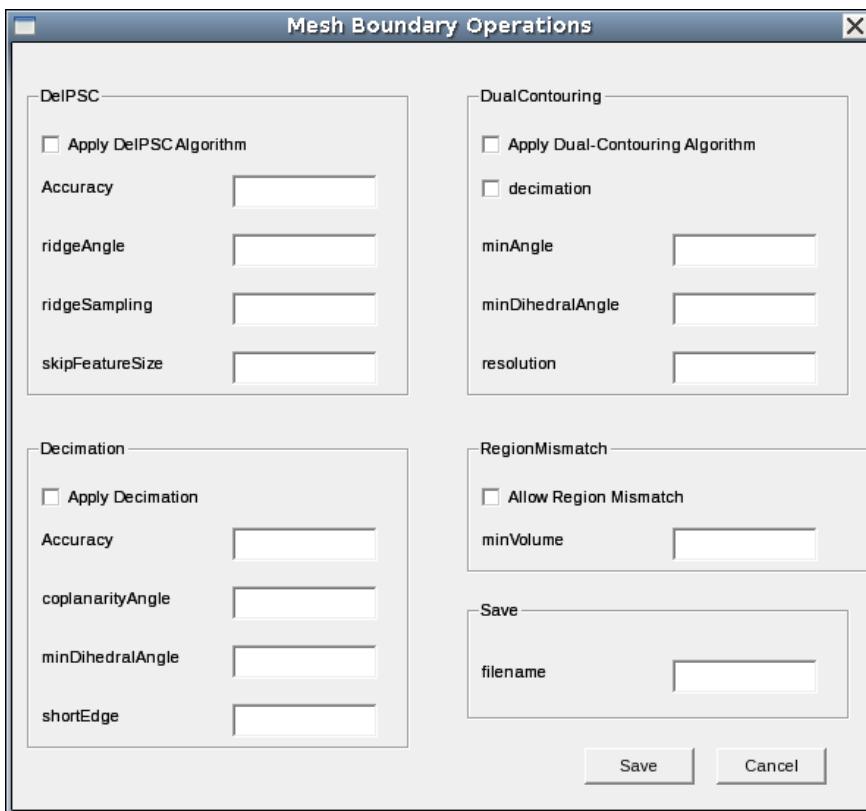
To define the Boundary section of the mesh command file:

1. Choose **Mesh > Boundary Operators**.

The Mesh Boundary Operations dialog box opens.

2. Select the required operations, and specify the parameters for those operations.

See *Sentaurus™ Mesh User Guide*, Boundary Section, for descriptions of parameters.



3. (Optional) Under RegionMismatch, select **Allow Region Mismatch** and enter the value for **minVolume** for the mismatch if you want Sentaurus Mesh to continue when a small region is lost during the meshing process.
4. (Optional) Under Save, enter a file name if you want to save the modified boundary after the operations are performed.
5. Click **Save**.

The corresponding Scheme extension is [sdesnmesh:boundary on page 774](#). For example:

```
(sdesnmesh:boundary "DelPSC" "accuracy" 0.001)
```

Tools Section

To define the `Tools` section of the mesh command file:

1. Choose **Mesh > Tools Parameters**.

The Tools Block Editor opens.

2. Edit the contents of the `Tools` section of the mesh command file as required.

For details, see *Sentaurus™ Mesh User Guide*, Tools Section.

3. Click **Save**.

The corresponding Scheme extension is:

```
(sdesnmesh:tools "tools-block-contents")
```

where `"tools-block-contents"` is a string surrounded by double quotation marks that represents the required contents of the `Tools` section of the mesh command file.

You must escape double quotation marks with a backslash within `"tools-block-contents"`. For example:

```
(sdesnmesh:tools "RandomizeDoping { \
    DopingAssignment = \"Sano\" \
    NumberOfRandomizedProfiles = 1 \
    FileIndex = 1 \
    Material \"Silicon\" { \
        Species \"BoronActiveConcentration\" { \
            ScreeningFactor = 2.5e6 \
            AutoScreeningFactor \
        } \
        Species \"ArsenicActiveConcentration\" { \
            ScreeningFactor = 1.3e7 \
            AutoScreeningFactor \
        } \
    } \
} ")
```

QualityReport Section

You can define the `QualityReport` section of the mesh command file. This section can have one global block and multiple material and region blocks.

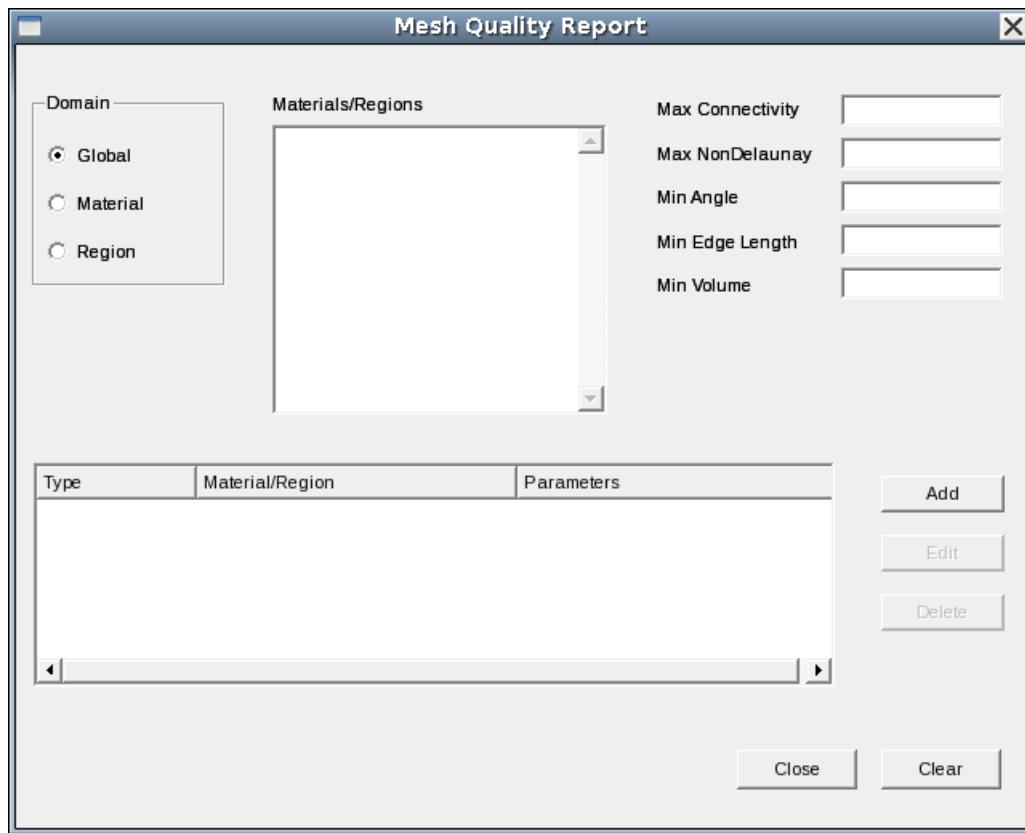
Chapter 7: Generating Mesh and Doping Profiles

Defining Command File Sections Specific to Sentaurus Mesh

To define the `QualityReport` section:

1. Choose **Mesh > Quality Report**.

The Mesh Quality Report opens.



2. Under Domain, select **Global**, **Material**, or **Region**.
3. Select the required items from the **Materials/Regions** list:
 - For a **Material** domain, this list contains the DATEX materials of the model.
 - For a **Region** domain, the list contains all region names of the model.
4. When the parameters are specified, click **Add**.
The defined block is listed in the lower pane.
5. (Optional) Select a list item and click **Edit** to edit the parameters and redefine the block.
6. (Optional) Select a list item and click **Delete** to remove it.
7. (Optional) Click **Clear** to delete all defined blocks of the `QualityReport` section.
8. Click **Close** to exit the dialog box.

Chapter 7: Generating Mesh and Doping Profiles

Building the Mesh

The corresponding Scheme extension is [sdesnmesh:qualityreport on page 782](#).

Building the Mesh

After all refinement and doping specifications have been created, you can mesh the structure by using Sentaurus Mesh.

To build the mesh:

1. Choose **Mesh > Build Mesh**.

The Build Mesh dialog box opens.



2. In the **Save Grid to File** field, enter the name of the file where the mesh will be saved, or click **Browser** to navigate to the required file.

Extensions such as `.cmd`, `.z`, and `.gz` are removed from the file name to form the base name of the mesh command file.

Note:

If you specified `inputFile` in the `IOControls` section, then it overrides the base name of the saved boundary file.

Under Mesh Viewer, **SVisual** is selected by default to use Sentaurus Visual to view the created mesh.

3. In the **Other options** field, enter command-line options as required.

See *Sentaurus™ Mesh User Guide*, Command-Line Options.

4. Click **Save Values** to save the global parameters without executing Sentaurus Mesh.
5. Click **Build Mesh** to create the mesh.

8

Creating Planar Layer Stacks

This chapter discusses how to create planar layer stacks efficiently including doping profiles, mole fraction profiles, and refinements.

Using the sdeepi:create-layerstack Scheme Extension

The `sdeepi:create-layerstack` Scheme extension is useful for simulating compound semiconductor devices based on epitaxial layer stacks. Such structures are typically grown using epitaxial growth techniques such as metal-organic chemical vapor deposition or molecular beam epitaxy. This Scheme extension facilitates structure generation for epitaxial layer stacks.

A typical epitaxial structure consists of a stack of planar layers. The straightforward approach to generate such a structure in Sentaurus Structure Editor requires approximately 10 to 15 commands to set up the geometry, doping, mole-fraction profiles, and refinements for each layer. This makes structure generation unnecessarily complex. Using `sdeepi:create-layerstack`, the layer stack description can be greatly simplified by using a single line to define all layer properties for each layer.

To facilitate working with and organizing material parameters, the MatPar utility is available (for details, see [1]).

The `sdeepi:create-layerstack` Scheme extension has the following features:

- All settings and layer properties of the layer stack are specified in a comma-separated value (CSV) file that can be edited easily in a text editor or a spreadsheet application.
- All layer properties are defined efficiently with a single line in the CSV file.
- Based on the CSV file, `sdeepi:create-layerstack` generates the geometric representation of the layer stack itself as well as doping, mole profile definitions, and refinements.
- Automatic creation of top and bottom contacts.
- Both two-dimensional and three-dimensional layer stacks can be created.

Chapter 8: Creating Planar Layer Stacks

Command File

- Both the unified coordinate system (UCS) and the DF–ISE coordinate system are supported.
- The epitaxial layers can have constant or graded doping and mole-fraction profiles.
- The layerwise mesh refinement strategy can be specified in the CSV file.

The `sdeepi:create-layerstack` Scheme extension creates a planar layer stack from the user-defined description specified in the command file `sde_epi.csv`, which consists of the following major sections:

- The *global section* where any global settings such as the lateral stack dimensions, global refinement, and the top and bottom contact names can be defined.
- The *layers section* that defines for each layer its properties such as region name, material, thickness, mole fraction, and refinement in a single row.
- An additional *parameter file section* where the generation of parameter files performed with MatPar can be controlled in more detail [1].

The command file `sde_epi.csv` can be edited easily by right-clicking the Sentaurus Structure Editor icon in the tool row and choosing **Edit Input > CSV Data File**.

Command File

The command file `sde_epi.csv` contains all the information needed to create the layer stack. The file is preprocessed before execution as usual; therefore, it can include any common preprocessing and Tcl preprocessing statements.

The entire file is structured as a comma-separated value (CSV) file. Therefore, it consists of fields (columns) separated by commas. Each column is handled as a separate entity, for example, the name of the layer or the thickness of the layer. Any trailing commas are ignored.

Any line starting with a hash (#) as the first nonspace character is interpreted as a comment and is ignored.

The following example is a CSV file generating a simple 2D p-n GaAs diode:

```
$global coordinateSystem=UCS
$global Ymin=0.0, Ymax=2.0
$global dXmax=0.2
$global topContact=anode, bottomContact=cathode
#Region, Material, ParFile, Thickness, Doping, MoleFrac, Refinement
cap,GaAs,,0.2,-1E+019,,(xref 0.02)
fsf,AlGaAs,AlGaAs.par,0.04,-2E+018,0.8,(xref 0.01)
emitter,GaAs,,0.8,-9E+017,,(mbox 0.05 1.1 both)
base,GaAs,,3.2,1E+017,,(mbox 0.01 1.1 both)
```

Chapter 8: Creating Planar Layer Stacks

Command File

```
bsf,AlGaAs,AlGaAs.par,0.2,5E+018,0.2,(mbox 0.01 1.2 both)
buffer,GaAs,,0.35,2E+018,,(mbox 0.01 1.2 up)
```

Global Section

The global section of the command file defines variables called *global variables*, which define the properties of the complete layer stack such as stack width, global refinement, and top and bottom contact names. The value of a particular global variable is the same for all layers in the stack.

A line starting with \$global is used to modify a global variable in the following way:

```
$global variable1=<value>, variable2=<value>, ...
```

Note:

The variable names are case sensitive.

An example of a typical global section is:

```
# Simple solar cell structure
$global coordinateSystem=DFISE
$global Xmax=@wtot@
$global dXmin=5, dXmax=5, dYmin=0.5, dYmax=0.5
$global topContact=anode, bottomContact=cathode
```

where:

- coordinateSystem sets the coordinate system used when saving the TDR file. More precisely, it defines the up direction of the structure:

Coordinate system	Dimension	Up direction
UCS	2D	-x
UCS	3D	-x
DF-ISE	2D	-y
DF-ISE	3D	+z

- Xmax sets the width of the layer stack to the Sentaurus Workbench parameter @wtot@.
- dXmin, dXmax, dYmin, and dYmax define some global refinement.
- topContact and bottomContact define the electrode names at the top and bottom of the structure, respectively.

Chapter 8: Creating Planar Layer Stacks

Command File

For a complete list of global variables, see [Table 32](#). Details about the variables are described in the following sections.

All global variables can be written to the layer information file `nX_epi.tcl` using the command:

```
(sdeepi:tcl "nX_epi.csv" "nX_epi.tcl")
```

Therefore, all global variables are available as Tcl variables in Tcl preprocessing blocks of any subsequent tools by sourcing this file. For example, to print the minimum y-coordinate of the stack inside a subsequent command file, insert the following:

```
! (
    source "nX_epi.tcl"
    puts min
) !
```

Note:

In Sentaurus Workbench, the preprocessed CSV file `nX_epi.csv` can be accessed by `@epicsv@`. Similarly, the layer information file `nX_epi.tcl` can be accessed through `@epitcl@`.

Table 32 Structure information in global section of CSV file, listing predefined global variables

Predefined global variables	Description	Settable?	Default	Unit
Geometric properties of complete layer stack in all three directions				
Xmin	Minimum extent of the layer stack in the x-direction.	Yes	0	µm
Xmax	Maximum extent of the layer stack in the x-direction. Ignored if set for the UCS.	(Yes)	1	µm
Ymin	Minimum extent of the layer stack in the y-direction.	Yes	0	µm
Ymax	Maximum extent of the layer stack in the y-direction. Ignored if set for the DF–ISE coordinate system (two dimensions).	(Yes)	1	µm
Zmin	Minimum extent of the layer stack in the z-direction. It can be set only for UCS (three dimensions); otherwise, it is ignored.	(Yes)	0	µm

Chapter 8: Creating Planar Layer Stacks

Command File

Table 32 Structure information in global section of CSV file, listing predefined global variables (Continued)

Predefined global variables	Description	Settable?	Default	Unit
Zmax	Maximum extent of the layer stack in the z-direction. Ignored if set in two dimensions.	(Yes)	1	µm
coordinateSystem	Either UCS (unified coordinate system) or DFISE (DF-ISE coordinate system).	Yes	UCS	—
dimension	For 2D structures, dimension=2. For 3D structures, dimension=3.	Yes	2	—
Global mesh refinement strategy for entire structure				
dXmin, dYmin, dZmin	Minimum element size in the xyz directions.	Yes	9999.0	µm
dXmax, dYmax, dZmax	Maximum element size in the xyz directions.	Yes	9999.0	µm
Electrode information				
topContact	If defined, contains the name of the contact to be created on the top of the layer stack.	Yes	—	—
bottomContact	If defined, contains the name of the contact to be created at the bottom of the layer stack.	Yes	—	—
Doping information				
NDopant	Doping species used for n-doping. For a more generic species, you can use NDopantActiveConcentration.	Yes	ArsenicActive Concentration	—
PDopant	Doping species used for p-doping. For a more generic species, you can use PDopantActiveConcentration.	Yes	BoronActive Concentration	—

Chapter 8: Creating Planar Layer Stacks

Command File

Table 32 Structure information in global section of CSV file, listing predefined global variables (Continued)

Predefined global variables	Description	Settable?	Default	Unit
Miscellaneous information				
columnNames	Contains a list of column names. To add user-defined columns, use the following syntax: \$global append columnNames <column1> <column2> ...	Yes	—	—
bottomRegionName	Region name of the bottommost layer.	No	—	—
topRegionName	Region name of the topmost layer.	No	—	—
nDopant	Dataset name used for n-type doping.	Yes	ArsenicActive Concentration	—
pDopant	Dataset name used for p-type doping.	Yes	BoronActive Concentration	—

Size and Position of Structure

The variables `Xmin`, `Xmax`, `Ymin`, `Ymax`, `Zmin`, and `Zmax` describe the extent of the layer stack in all three directions.

Figure 129 shows a 2D layer stack created by `sdeepi:create-layerstack` and illustrates both the UCS and the DF–ISE coordinate system. The coordinate system is chosen with the global variable `coordinateSystem`, for example, the default is `coordinateSystem=UCS`. In this case, the top of the layer stack is in the `x`-direction. The topmost layer in the command file will be created with its top edge at `Xmin`.

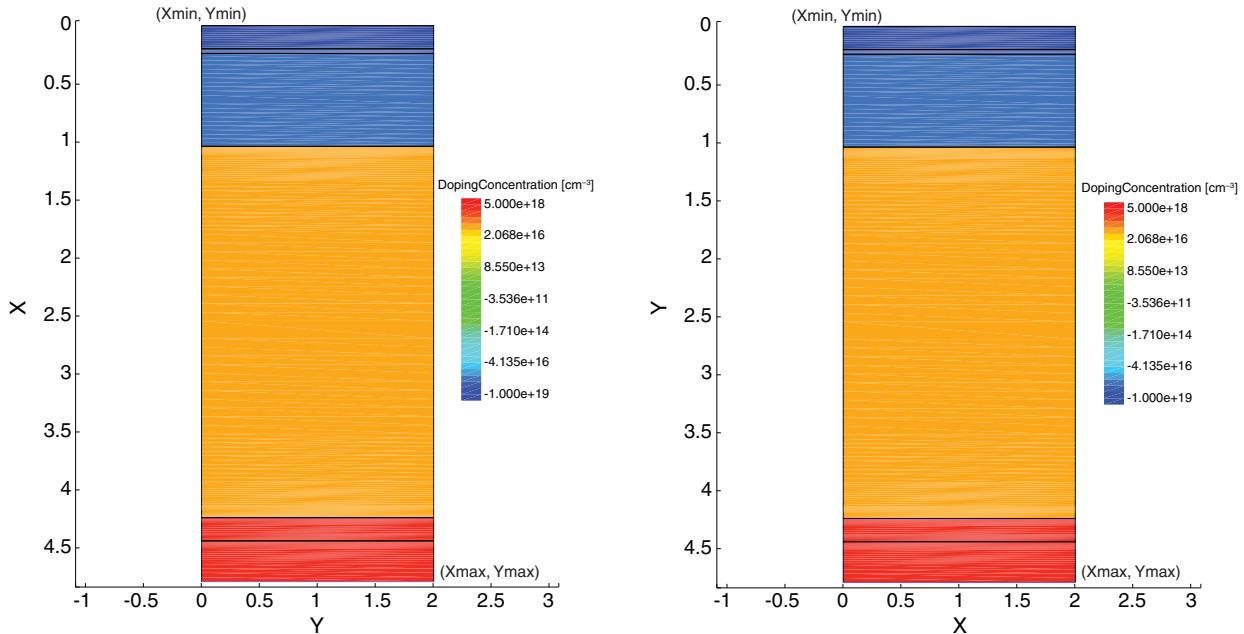
Set `coordinateSystem=DFISE` to activate the DF–ISE coordinate system. In this case, for a 2D structure, the top of the layer stack is in the `-y`-direction and the topmost layer is at `Ymin`.

Each subsequent layer line will be created in the order it appears in the command file, until the last layer is created. The bottom edge of the last layer defines `Xmax` for the UCS (or `Ymax` for the DF–ISE coordinate system for a 2D structure).

Chapter 8: Creating Planar Layer Stacks

Command File

Figure 129 Generated layer stack with: (left) UCS and (right) DF–ISE coordinate system



By default, $X_{min} = Y_{min} = 0$. In addition, as shown in Figure 129, it is required that $X_{min} < X_{max}$. All layers have the same width, which can be defined using Y_{min} and Y_{max} for the UCS, or X_{min} and X_{max} for the DF–ISE coordinate system (see Table 32 on page 290).

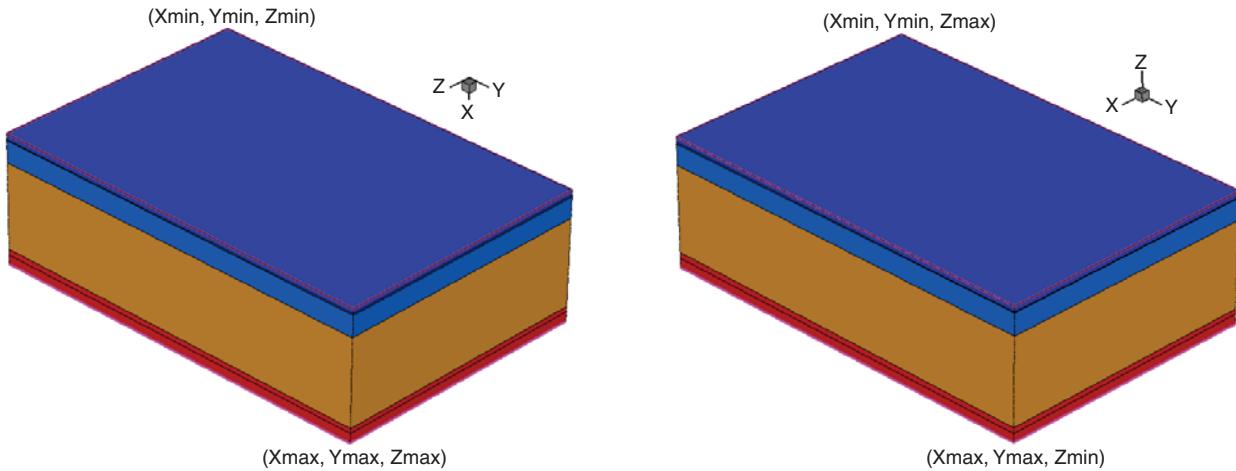
If the global variable `dimension=3` is specified, a corresponding 3D structure is created as illustrated in Figure 130. The top of the layer stack is in the $-x$ -direction for the UCS and in the $+z$ -direction for the DF–ISE coordinate system. The extent of the layer stack in the third dimension is controlled with the `Zmin` and `Zmax` variables for the UCS, and the `Ymin` and `Ymax` variables for the DF–ISE coordinate system.

The next sections describe the usage of `sdeepi` in the UCS.

Chapter 8: Creating Planar Layer Stacks

Command File

Figure 130 Three-dimensional layer stack with (left) UCS and updirection=-x, Ymax=15, and Zmax=10, and (right) DF-ISE coordinate system and updirection=+z, Xmax=10, and Ymax=15



Global Refinement Strategy

In addition to layerwise refinements (see [Refinement Column on page 304](#)), `sdeepi:create-layerstack` allows you to define a global refinement that is defined as a refinement window covering the entire layer stack.

`dXmin` and `dXmax` are minimum and maximum element sizes, respectively, for global mesh refinement in the x-direction for the entire structure. `dYmin`, `dYmax`, `dZmin`, and `dZmax` have similar meaning.

If you define `dXmin=0`, or `dYmin=0`, or `dZmin=0`, `sdeepi:create-layerstack` resets this minimum element size to be equal to the corresponding maximum element size. The global refinement window is positioned to cover the entire layer stack.

Note:

The `sdeepi:create-layerstack` Scheme extension generates global refinement commands only if you modify the value of at least one of these variables.

User-Defined Extension Columns

In addition to the predefined layer properties, `sdeepi:create-layerstack` allows you to add user-defined layer properties. This can be useful in cases where some tools need additional layer-specific data. For example, to switch on radiative recombination in Sentaurus Device for particular layers, an additional layer column `Rrad` could be defined with:

```
$global append columnNames Rrad
```

Chapter 8: Creating Planar Layer Stacks

Command File

This user-defined column can be filled with values as described in [Extension Columns on page 306](#). The column values are saved with all other properties in the Tcl file `nX_epi.tcl` and can be incorporated in any subsequent Tcl preprocessing block.

Note:

To create multiple extension columns, use the following syntax:

```
$global append columnNames <extensionColumnName1>  
    <extensionColumnName2> ...
```

Layers Section

All layers of the layer stack are defined in the layers section, which consists of one line for each layer to be created. Basically, any line not starting with a hash (#) or a dollar sign (\$) is a layer line. Stack generation starts with the first layer line as the topmost layer of the stack and continues from top to bottom until the last layer line is reached.

Each layer line consists of several comma-separated columns, which `sdeepi:create-layerstack` uses to create the layer stack:

```
<Region>, <Material>, <SourceParFile>, <Thickness>, <Doping>,  
<MoleFraction>, <Refinement>, [<Extension1>, <Extension2>, ...]
```

Lines starting with an empty region name are used to create materialwise parameter file definitions for MatPar [\[1\]](#).

[Table 33](#) summarizes the columns used to describe each layer.

Table 33 Columns in layers section of CSV file

Column	Description
Region	Region name.
Material	Material name.
SourceParFile	Name of the source parameter file. Only used if MatPar is used to generate parameter files.
Thickness	Layer thickness. Default unit: μm
Doping	Doping of each layer. A negative sign indicates p-type doping; a positive sign indicates n-type doping. Default unit: cm^{-3}
MoleFraction	Mole fraction for ternary or quaternary materials. Default unit: [1]

Chapter 8: Creating Planar Layer Stacks

Command File

Table 33 Columns in layers section of CSV file (Continued)

Column	Description
Refinement	Regionwise mesh refinement strategy for each layer. Default unit: μm
	Additional user-defined extension columns can be added for various purposes.

The following is an example of the layers section of the command file of a GaAs single-junction solar cell:

```
# Layers section
# Region, Material, SourceParFile, Thickness, Doping, MoleFraction,
# Refinement
cap,GaAs,,0.5,-5e19,,(xref 0.1)
fsf,AlGaAs,AlGaAs.par,0.03,-4e18,0.8,(xref 0.01)
emitter,GaAs,,0.5,-5e17,,(mbox 0.05 1.1 both)
base,GaAs,,2.5,1e17,,(mbox 0.05 1.1 both)
substrate,Germanium,Ge.par,5.0,1e18,,(mbox 0.05 1.1 both)
```

[Table 34](#) shows how the same listing in a spreadsheet format, underlining the well-arranged representation, compares to the source file listing.

Table 34 Example of layers section and material and interface section of epi_epi.csv

Line #	Region	Material	SourceParFile	Thickness	Doping	Mole Fraction	Refinement
# Layers section:							
1	cap	GaAs		0.5	-5.00e19		(xref 0.1)
2	fsf	AlGaAs	AlGaAs.par	0.03	-4.00e18	0.8	(xref 0.01)
3	emitter	GaAs		0.5	-5.00e17		(mbox 0.05 1.1 both)
4	base	GaAs		2.5	1.00e17		(mbox 0.05 1.1 both)
5	substrate	Germanium	Ge.par	5.0	1.00e18		(mbox 0.05 1.1 both)

Note:

If all columns following a particular column are empty in a layer definition, any trailing commas are ignored. Therefore, the following two layer definitions are identical:

```
cap,GaAs,GaAs.par,0.5,-5e19,,,  
cap,GaAs,GaAs.par,0.5,-5e19
```

Leading and intermediate commas are important.

Region Column

The `sdeepi:create-layerstack` Scheme extension creates a region for each layer. The name of the region is specified in the Region column.

Material Column

For each layer, the names of materials that Sentaurus Device recognizes are specified in the Material column. For example, for a germanium layer, `Germanium` is specified in the Material column.

To generate a list of all materials that Sentaurus Device recognizes, use:

```
sdevice -L:materials
```

If a particular material is not in the list, it can be made available to all TCAD Sentaurus tools by creating a local `datexcodes.txt` file. For details on how to create new materials, see the *Sentaurus™ Device User Guide*.

SourceParFile Column

If MatPar is not used for parameter file generation, the SourceParFile column is ignored and can remain empty.

Otherwise, this column contains the source parameter file to be used for this layer [\[1\]](#).

Thickness Column

The thickness of each layer in micrometers is specified in the Thickness column. If the Thickness column is empty, or is less than or equal to zero, `sdeepi:create-layerstack` does not create the layer.

Doping Column

The doping information for each layer is specified in the Doping column. Doping values greater than zero create n-type doping using the dataset `ArsenicActiveConcentration`.

Chapter 8: Creating Planar Layer Stacks

Command File

Doping values less than zero create p-type doping using the dataset BoronActiveConcentration.

You can overwrite the default doping species by using the global variables NDopant and PDopant (see [Table 32 on page 290](#)).

One or several constant or analytic doping profiles can be specified using the commands in [Table 35](#). These commands create region-based doping profiles for the layer in the y-direction. The following types of doping profile can be defined in the y-direction:

- Constant
- Linear
- Error function
- Gaussian function
- General analytic function $f(x, y, z)$

If multiple doping profiles must be created, the bracket form must be used for profile definition. For example, to specify a constant doping of `n=1e15` and `p=2e15` in the same layer, use `(1e15)(-2e15)`.

Table 35 Commands for Doping column

Command	Description
Constant doping profile	
<code><doping></code> or: <code>(<doping> [<options:rnp>])</code>	Constant doping concentration. Default unit: cm ⁻³
Linear doping profile	
<code>(lin <Ntop> <Nbot> [<options:rsnp>])</code>	Linear doping profile using analytic profile.
<code><Ntop></code>	Value of doping concentration at the top edge of the layer. Default unit: cm ⁻³
<code><Nbot></code>	Value of doping concentration at the bottom edge of the layer. Default unit: cm ⁻³
Error function or Gaussian function doping profile	
<code>(erf gauss <Nmax> <ymax> <length> [<options:udrsnp>])</code>	Error function or Gaussian function doping profile.

Chapter 8: Creating Planar Layer Stacks

Command File

Table 35 Commands for Doping column (Continued)

Command	Description
<Nmax>	Maximum concentration/peak concentration. Default unit: cm ⁻³
<ymax>	For erf, this is the symmetry position. For gauss, this is the peak position. Default unit: μm
<length>	For gauss, <length> represents the distance between the peak position and a place where the profile value decays by a factor of exp(-1). For erf, <length> represents the distance between the symmetry position and a place where the profile value decays by a factor of 1/2*(1+erf(1)). Default unit: μm
General analytic function doping profile	
(func <func> [<options:rsgnp>])	Contains the general function expression, where the coordinates x,y,z are local to the layer assuming (xmin,ymin,zmin) of the layer as origin. The layer coordinates xmin, ymin, zmin, xmax, ymax, zmax and the layer extents dx, dy, dz also can be used in <func>.
<func>	Contains the general function expression, where the xyz coordinates are local to the layer assuming (xmin,ymin,zmin) of the layer as origin. Default unit: cm ⁻³
Options	
u d	Controls the edge or face of the layer at which the baseline of the profile is placed: <ul style="list-style-type: none">• u = Baseline is placed at the top edge or face (default).• d = Baseline is placed at the bottom edge or face.
r	If specified, the option Replace is used in the AnalyticalProfile section of the Placement section of the mesh command file. Otherwise, NoReplace is used.
s	If specified, EvaluateWindow is not defined in the AnalyticalProfile section of the Placement section of the mesh command file.
g	Uses the global instead of the local layer-specific coordinate system for x,y,z of the general function.

Chapter 8: Creating Planar Layer Stacks

Command File

Table 35 Commands for Doping column (Continued)

Command	Description
n p	Ignores the sign of the doping concentration and uses the specified doping type.
Default constant value for profiles	
(default <doping>)	Only used if MatPar is used for parameter file generation. Used in cases where material parameter calculation requires a constant doping value. <doping> defines the constant doping value to be used if required in the material parameter calculation. Default unit: cm ⁻³

For each layer, the doping profiles are created with the following parameters:

- The `AnalyticalProfile` section of the `Definition` section:
 - `Species=ArsenicActiveConcentration` | `BoronActiveConcentration` (for n-type or p-type doping, respectively)
You can overwrite the default doping species by using the global variables `NDopant` and `PDopant` (see [Table 32 on page 290](#)).
 - `Function=General` | `Error` | `Gauss` (for linear or general functions, error, or Gaussian function doping profile, respectively)
 - `LateralFunction=Erf(Factor=0)` (for error and Gaussian profiles)
- The `AnalyticalProfile` section of the `Placements` section:
 - The baseline for each layer is the bottom or top edge (2D) or face (3D), depending on the specification of the option `u|d` in the Doping column (see [Table 35](#)).
 - By default, the option `NoReplace` is used. However, if the option `r` is specified in the Doping column (see [Table 35](#)), `Replace` is used instead.
 - The variable `Direction` is not specified in the `ReferenceElement` section since the function values are computed on both sides of the baseline.
 - By default, all analytic doping profiles are placed using the `EvaluateWindow` section. For each layer, the domain for the evaluation of the profile is the layer region and `DecayLength=0`. The `EvaluateWindow` is not defined if the option `s` is specified in the Doping column (see [Table 35](#)).

For more details, write out the mesh command file with `sdedr:write-cmd-file`, and see [Sentaurus™ Mesh User Guide](#).

Chapter 8: Creating Planar Layer Stacks

Command File

Examples of specifying constant doping profiles are given in [Table 34 on page 296](#). Lines 1–3 are defined as p-type doped layers, and lines 4 and 5 are defined as n-type doped layers.

An example of an n-type doped layer with a Gaussian doping profile, with a peak concentration of $1\text{e}15 \text{ cm}^{-3}$, a relative peak position from the top of the layer of $0.2 \mu\text{m}$, and a decay length of $0.3 \mu\text{m}$ is defined with the following line:

```
4dopgauss,GaAs,,1,(gauss 1e15 0.2 0.3)
```

A constant p-type background doping of $2\text{e}16 \text{ cm}^{-3}$ plus an n-type sine-modulated doping using a default value of $1\text{e}16 \text{ cm}^{-3}$ for the material parameter calculation can be defined as follows:

```
6dopfunc,GaAs,,1,(2e16 p)(func 1e15*sin(2*3.1415*x/dx) n)(default -1e16)
```

If MatPar is used, then a default value for the reference doping concentration can be defined for layers with doping profiles with the command `default`. A constant doping value could be necessary in some cases, if during material parameter calculation, the specified model cannot handle the doping profile and requires a constant value instead.

MoleFraction Column

The mole-fraction information of layers consisting of ternary or quaternary compound semiconductor materials is described in the MoleFraction column. The column is empty for elemental or binary semiconductors. A constant mole fraction or a mole-fraction profile very similar to doping profiles can be specified using the commands in [Table 36](#). These commands create region-based mole-fraction profiles for the layer in the y-direction.

A default reference mole-fraction value can be defined for layers with mole-fraction profiles with the command `default`.

A constant mole-fraction value might be necessary in some cases, if during material parameter calculation, the specified model cannot handle the mole-fraction profile and requires a constant mole-fraction value instead.

Table 36 Commands for MoleFraction column

Command	Description
Constant mole fraction	
<xMole> or: (<xMole> [<options:rxy>])	Defines a constant x-mole fraction for ternary compound semiconductors.

Chapter 8: Creating Planar Layer Stacks

Command File

Table 36 Commands for MoleFraction column (Continued)

Command	Description
Linear mole-fraction profile	
(lin <xMoleTop> <xMoleBot> [<options:rsxy>])	Linear x–mole fraction profile using analytic profile.
<xMoleTop>	Mole-fraction value at the top edge of the layer.
<xMoleBot>	Mole-fraction value at the bottom edge of the layer.
Error function or Gaussian function mole-fraction profile	
(erf gauss <xMoleMax> <ymax> <length> [<options:udrsxy>])	Error function or Gaussian function mole-fraction profile.
<xMoleMax>	Maximum value/peak value.
<ymax>	Symmetry position/peak position. Default unit: μm
<length>	For gauss, <length> represents the distance between the peak position and a place where the profile value decays by a factor of $\exp(-1)$. For erf, <length> represents the distance between the symmetry position and a place where the profile value decays by a factor of $1/2*(1+\text{erf}(1))$. Default unit: μm
General analytic function doping profile	
(func <func> [<options:rsgxy>])	Generates an arbitrary analytic mole-fraction profile using the general function evaluator of Sentaurus Mesh.
<func>	Contains the general function expression, where the coordinates x,y,z are local to the layer assuming (xmin,ymin,zmin) of the layer as origin. The layer coordinates xmin, ymin, zmin, xmax, ymax, zmax and the layer extents dx, dy, dz also can be used in <func>.
Options	
u d	Controls the edge or face of the layer at which the baseline of the profile is placed: <ul style="list-style-type: none"> • u = Baseline is placed at the top edge or face (default). • d = Baseline is placed at the bottom edge or face.

Chapter 8: Creating Planar Layer Stacks

Command File

Table 36 Commands for MoleFraction column (Continued)

Command	Description
r	If specified, the option Replace is used in the AnalyticalProfile section of the Placement section of the mesh command file. Otherwise, NoReplace is used.
s	If specified, EvaluateWindow is not defined in the AnalyticalProfile section of the Placement section of the mesh command file.
g	Uses the global instead of the local layer-specific coordinate system for x,y,z of the general function.
x y	Sets the mole-fraction type explicitly. Default is x.
Default constant value for profiles	
(default <xMole> [<yMole>])	Only used if MatPar is used for parameter file generation. Used in cases where material parameter calculation requires a constant x- or y-mole fraction value.
<xMole>, <yMole>	Defines the constant x- and y-mole fraction value to be used if required in the material parameter calculation.
Mole-fraction profiles for quaternary materials	
(y <yMole> lin erf func gauss ...)	Creates a y-mole fraction profile very similar to the above definitions for x-mole fraction profiles.
(<xMole> <yMole>)	Defines a constant x- and y-mole fraction for quaternary compound semiconductors.
(... x)(... y)	In general, x- and y-mole fraction profiles can be defined separately using the x and y options. For example: (lin 0.1 0.2 x)(lin 0.9 0.8 y)

For example, to create a layer with a constant x-mole fraction profile with a value of 0.1, the following lines are identical:

```
1molconst,AlGaAs,,1,1e15,0.1
2molconst,AlGaAs,,1,1e15,(0.1)
3molconst,AlGaAs,,1,1e15,(0.1 x)
```

Chapter 8: Creating Planar Layer Stacks

Command File

The following line creates a linear x–mole fraction profile with a value of 0.3 at the top and 0.4 at the bottom:

```
3mollin,AlGaAs,,1,1e15,(lin 0.3 0.4)
```

To create a constant mole-fraction profile of $x=0.2$ $y=0.3$, all of the following expressions are identical:

```
(0.2 0.3), (0.2)(y 0.3), (0.2)(0.3 y), (0.3 y)(0.2 x)
```

Refinement Column

The Refinement column in the command file offers two different meshing strategies:

- Constant refinement in the xyz directions with the commands `xref`, `yref`, and `zref`.
- Graded refinement, dense at an interface, relaxing towards the bulk, using the command `mbox` (which stands for multibox).

Mesh refinement for a particular layer is performed using the commands `xref` and `mbox` (see [Table 37](#)). The terminology related to mesh refinement is discussed in the *Sentaurus™ Mesh User Guide*.

Table 37 Commands for Refinement column

Command	Description
Uniform regionwise refinement	
<code>(xref yref zref <minelsize> [<maxelsize>])</code>	Uniform regionwise refinement in the xyz directions.
<code><minelsize></code>	Specifies the minimum element size (mesh spacing) in the corresponding direction. Default is the layer extent in the corresponding direction.
<code><maxelsize></code>	Specifies the maximum element size (mesh spacing) in the corresponding direction. By default, it is set equal to the minimum element size.
Multibox regionwise refinement	
<code>(mbox <minelsize> <ratio> <side:udlrfb>)</code>	Defines a multibox refinement for a layer for the specified sides.
<code><minelsize></code>	Specifies the minimum element size at the interface.

Chapter 8: Creating Planar Layer Stacks

Command File

Table 37 Commands for Refinement column (Continued)

Command	Description
<ratio>	Controls how fast the mesh spacing increases from the minimum value to the maximum value, which is the layer thickness. <code>ratio=1</code> is equal to a constant meshing with <minelsize>. With <code>ratio=2</code> , the element size is doubled at each step.
<side:udlrfb>	Controls the edge (2D) or the face (3D) of the multibox from which the graded mesh will start. Default is <code>ud</code> . Options (described for the UCS case; see Figure 130 on page 294) are: <ul style="list-style-type: none"> • <code>u</code> (up): Grading starts from the top (<code>xmin</code>) of the layer. • <code>d</code> (down): Grading starts from the bottom (<code>xmax</code>) of the layer. • <code>l</code> (left): Grading starts from the left (<code>ymin</code>) of the layer. • <code>r</code> (right): Grading starts from the right (<code>ymax</code>) of the layer. • <code>f</code> (front): For 3D structures, grading starts from the front (<code>zmax</code>) of the layer. • <code>b</code> (back): For 3D structures, grading starts from the back (<code>zmin</code>) of the layer. The keywords written in parentheses after each option can be used as synonyms. In addition, the keyword <code>both</code> can be used as a synonym for <code>ud</code> .
MaxLenInt interface refinement	
(int <Value> <Factor> <side:udlrfb>)	Specifies a <code>MaxLenInt</code> -type interface refinement.
<Value>	Specifies the initial thickness.
<Factor>	Specifies the factor by which the initial thickness should grow into the material.
<side:udlrfb>	Controls the edge (2D) or the face (3D) of the multibox from which the graded mesh will start. Default is <code>ud</code> . Options (described for the UCS case; see Figure 130 on page 294) are: <ul style="list-style-type: none"> • <code>u</code> (up): Grading starts from the top (<code>xmin</code>) of the layer. • <code>d</code> (down): Grading starts from the bottom (<code>xmax</code>) of the layer. • <code>l</code> (left): Grading starts from the left (<code>ymin</code>) of the layer. • <code>r</code> (right): Grading starts from the right (<code>ymax</code>) of the layer. • <code>f</code> (front): For 3D structures, grading starts from the front (<code>zmax</code>) of the layer. • <code>b</code> (back): For 3D structures, grading starts from the back (<code>zmin</code>) of the layer. The keywords written in parentheses after each option can be used as synonyms. In addition, the keyword <code>both</code> can be used as a synonym for <code>ud</code> .

Chapter 8: Creating Planar Layer Stacks

Command File

In addition to the layerwise refinements described here, refinements can be specified either using the global variables `dxmin`, `dxmax`, and so on, or using the standard Sentaurus Structure Editor refinement commands.

Looking at [Table 34 on page 296](#), the refinement statement for the region `cap` in line 1 (`xref 0.1`) generates an equidistant refinement in the x-direction with a spacing of 0.1 μm. Very similar refinement is generated with the expression (`xref 0.01`) for the region `fsf` in line 2 but using a spacing of 0.01 μm. For lines 3–5, each layer is refined in the x-direction with a multibox refinement statement (`mbox 0.05 1.1 both`), starting with a spacing of 0.05 at both interfaces and gradually increasing the element size by a factor of 1.1 towards the bulk.

Extension Columns

In addition to the predefined layer properties, `sdeepi:create-layerstack` allows you to add multiple user-defined layer properties that are appended to the layer line as additional columns.

The Extension column can have arbitrary content and can be used by any subsequent Tcl preprocessing blocks. For example, you could introduce an additional layer property or column `Rrad` to switch on the radiative recombination model in the Sentaurus Device command file for certain layers.

The following lines in the CSV file would define such an additional column, with the value `no` for the `fsf` region, and the value `yes` for the emitter region:

```
$global append columnNames Rrad
fsf,AlGaAs,AlGaAs.par,0.03,-4.00e18,0.8,(xref 0.01),no
emitter,GaAs,,0.5,-5.00e17,,(mbox 0.05 1.1 both),yes
```

With the Scheme command `sdeepi:tcl`, all the layer information including the Extension columns can be written to a Tcl file @epitcl@ (for example, `n1_epi.tcl`):

```
(sdeepi:tcl "@epicsv@" "@epitcl@")
```

The following is an excerpt from the Tcl file:

```
set epi(region,fsf,region) "fsf"
set epi(region,fsf,Rrad) "no"
set epi(region,emitter,region) "emitter"
set epi(region,emitter,Rrad) "yes"
```

The values of the `Rrad` column then can be accessed and used in the Sentaurus Device command file to decide whether radiative recombination should be switched on or off for a certain layer:

```
! (
    source @epitcl@
    foreach {key region} [array get epi "region,* ,region"] {
        if {"$epi(region,$region,Rrad)" == "yes"} {
            puts "Physics (region=\"$region\")"
```

Chapter 8: Creating Planar Layer Stacks

Processing the Layer Stack

```
        {Recombination(Radiative)}"
    }
)
!
```

The first line sources all Tcl variables of the layer information file into the Tcl interpreter of the preprocessor. The `foreach` statement loops through all Tcl variables that have the form `epi(region,*,region)`, that is, all layers of the layer stack. The variable `$region` contains the region name of the current layer. Any other column value can be accessed by `$epi(region,$region,<column>)` where `<column>` is replaced by the column name of interest. In this case, you want to access the Extension column `Rrad`, therefore, you use `$epi(region,$region,Rrad)` and, if its content is "yes", a regionwise `Physics` entry is printed to the command file.

The result of the preprocessed block for this example is:

```
Physics (region="emitter") {Recombination(Radiative)}
```

Note:

By default, there is no Extension column.

Processing the Layer Stack

The `sdeepi:create-layerstack` Scheme extension creates a layer stack as it is produced during any nonstructuring deposition process, for example, molecular beam epitaxy (MBE) or metal organic vapor phase epitaxy (MOVPE). Subsequent commands can be used to reflect any further processing steps of the wafer such as lateral patterning steps.

[Figure 131](#) illustrates this by a layered GaAs solar cell structure. On the left, the structure created using `sdeepi:create-layerstack` is shown; on the right, the processed solar cell structure is illustrated with a partially removed cap layer on the top, a partially deposited ARC, and contacts.

For convenience, all global variables as well as geometric variables can be made available to Sentaurus Structure Editor as Scheme variables using:

```
(sdeepi:publish-global-vars #t)
```

By default, to avoid overwriting existing Scheme variables, none of the layer stack variables is retained after the stack creation.

Various device-processing steps such as etching and additional layer deposition require the top and bottom coordinates of a layer. These are available as the Scheme variables `x0_<region>` and `x1_<region>`:

- `x0_<region>` is the x-coordinate of the top edge of the region.
- `x1_<region>` is the x-coordinate of the bottom edge of the region.

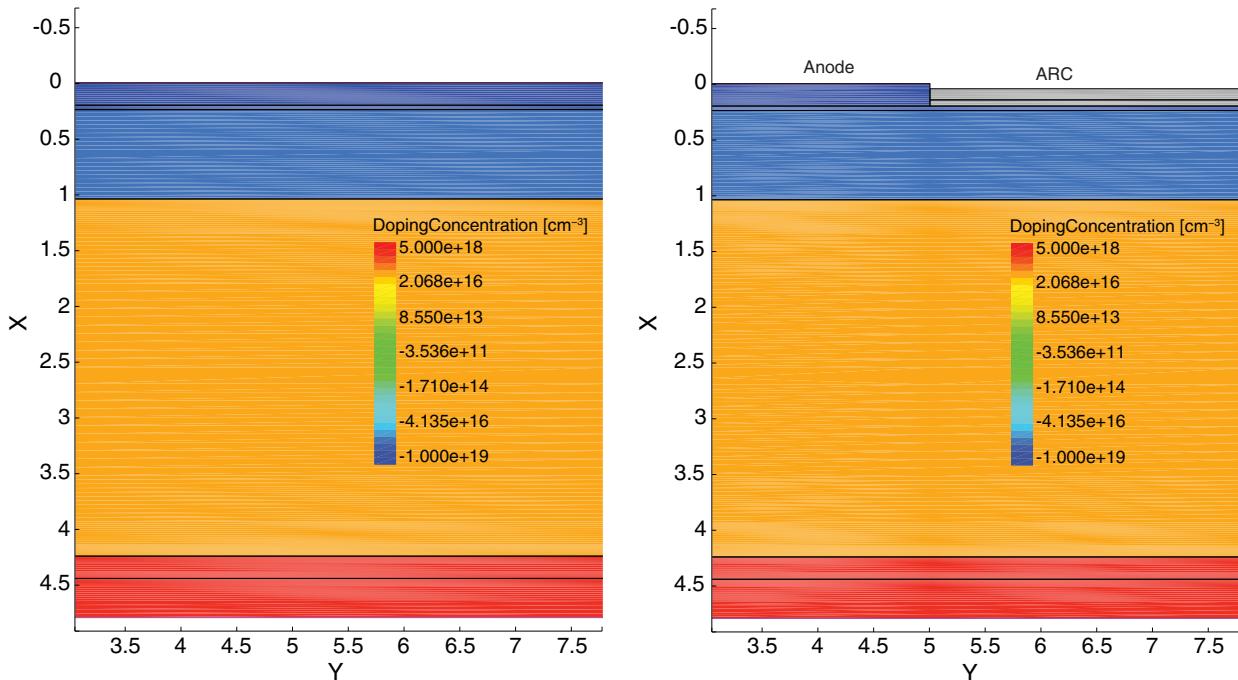
Chapter 8: Creating Planar Layer Stacks

References

To investigate the layer stack creation in more detail and to evaluate the Scheme file step by step in Sentaurus Structure Editor, you can save the used Scheme commands to a file. For example:

```
(sdeepi:scm "@epicsv@" "@episcm@")
```

Figure 131 (Left) Layer stack as created by sdeepi:create-layerstack and (right) device structure with additional processing steps



References

- [1] *Simulation of 2D Single-Junction GaAs Solar Cell*, available from TCAD Sentaurus Version T-2022.03 installation, go to [Applications_Library/Solar/SolarCell_SingleJunct_GaAs](#).

9

Working With Scheme and Scheme Extensions

This chapter discusses working with Scheme and Scheme extensions in Sentaurus Structure Editor.

Scheme Data Types

The Scheme scripting language is used in Sentaurus Structure Editor and the process emulation module of Sentaurus Structure Editor (Procem).

Table 38 lists the data types that can be used as arguments when calling the Scheme functions in Sentaurus Structure Editor.

Table 38 Data types for Scheme functions related to Sentaurus Structure Editor

Data type	Description
BOOLEAN	A native Scheme data type having either the value #t (true) or #f (false). It represents a logical or Boolean value.
DATEXMAT	A STRING type variable, containing a valid material name, as defined in the datexcodes.txt file. "Silicon" and "Oxide" are valid DATEXMAT variables.
ELIST	An ACIS entity list.
ENTITY	An ACIS entity.
GVECTOR	A GVECTOR entity is an ACIS entity. It contains the keyword gvector and it is a composition of three real numbers representing the x-, y-, and z-components of a 3D vector. For example, (define g1 (gvector 1 0 0)) defines a GVECTOR called g1. gvector is an ACIS-defined Scheme data type, used to represent a vector with magnitude and direction. It is called gvector only to differentiate it from the inherent Scheme data type vector, which represents an array.
INTEGER	A Scheme data type containing a single integer value.
PLIST	A polygon list: (list POLYGON POLYGON ... POLYGON).

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

Table 38 Data types for Scheme functions related to Sentaurus Structure Editor (Continued)

Data type	Description
POLARITY	Either "light" or "dark".
POLYGON	A position list: (list POSITION POSITION ... POSITION).
POSITION	A POSITION entity is an ACIS entity. It contains the keyword position and three real numbers. For example, (position 0 0 0) or (position -3 -3.1 -3.14) are valid POSITION entities. (define p1 (position a b c)) defines a Scheme object p1 that holds a POSITION entity. The x-coordinate of p1 is equal to a, the y-coordinate is equal to b, and the z-coordinate is equal to c.
REAL	A Scheme data type containing a single real or double value.
STRING	A Scheme object containing a text string. A string is enclosed in braces or starts with the character '. Two examples of a string are "this_is_a_valid_string" and 'this_is_also_fine'.
VARIABLE	A Scheme variable.

Basic Scheme Programming

Sentaurus Structure Editor uses Scheme, or more precisely, the Elk extension of Scheme. Scheme is a LISP-like programming language that differs significantly from most of the commonly used programming languages. The effective use of Sentaurus Structure Editor (except the GUI layer) requires a working knowledge of Scheme. In addition to the information given in this manual, there are many resources for learning Scheme, including:

- *R5RS, The Revised Report on the Algorithmic Language Scheme*
The report is available from <https://schemers.org/Documents/Standards/R5RS/HTML/>.
- R. K. Dybvig, *The Scheme Programming Language*, Cambridge, Massachusetts: MIT Press, 3rd ed., 2003.
- D. P. Friedman and M. Felleisen, *The Little Schemer*, Cambridge, Massachusetts: MIT Press, 4th ed., 1996.
- D. P. Friedman and M. Felleisen, *The Seasoned Schemer*, Cambridge, Massachusetts: MIT Press, 1996.
- <https://www.scheme.com> (website of a Scheme dialect, Chez Scheme)
- <https://schemers.org/> (website of Scheme-related topics)

In the following sections, the Scheme commands that are most commonly used in a Sentaurus Structure Editor script are introduced.

Basic Scheme Syntax

A Scheme command is enclosed in parentheses (white spaces are allowed):

```
(Scheme command)
```

All text in a line after a semicolon is treated as a comment:

```
; This is a comment  
(Scheme command) ; This is also a comment
```

A Scheme command can extend over several lines, or several Scheme commands can be placed on a single line. A given Scheme command is identified by the outermost matching parentheses pair:

```
(beginning of Scheme command  
continuation of the same Scheme command  
end of the Scheme command)  
  
(first Scheme command) (second Scheme command) (...)
```

Defining Simple Variables and Data Types

Integers and floating-point numbers are treated as numbers. To declare and define a numeric variable:

```
(define i 3)  
(define pi 3.141593)
```

Strings are enclosed in double quotation marks. To declare and define a string:

```
(define W "Hello World")
```

Characters are preceded by #\. To declare and define a character:

```
(define CHAR #\a)
```

Use the variable name to reference a variable:

```
i      ;-> 3  
pi    ;-> 3.141593  
W      ;-> "Hello World"  
CHAR   ;-> #\a
```

Update an existing variable with set!:

```
(define j 1)  
(set! j (+ j 1))  
j ;-> 2
```

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

Note:

Use `define` when a variable is introduced for the first time. Use `set!` to assign a new value to an existing variable.

The `define` command creates a local variable (lexical scoping). For example, a variable defined in a specific function is not visible in another function or in the main program. Different procedures can have their own private variables defined using the same variable name. The `set!` command does not alter the scope of already defined variables, while redefining a preexisting variable might alter its scope. Scheme variables need not be deleted explicitly. An automatic garbage collector will remove all Scheme variables when they go out of scope. Memory allocation or deallocation is performed automatically by Scheme.

Do not use the Scheme keyword `length` as a user-defined variable.

Scheme is case sensitive.

Printing Text to Standard Output

Use the `display` command to write to standard output. Use `newline` to create a line break:

```
(display "The value of i is ") (display i) (newline)
(display "The value of pi is ") (display pi) (newline)
(display "The string W contains >") (display W) (display "<") (newline)

;-> The value of i is 3
;-> The value of pi is 3.141593
;-> The string W contains >Hello World<
```

String Operations

To automatically generate identifiers, for example, region names, certain string operations are useful.

To define a string:

```
(define TEXTSTRING "This is a string")
```

To determine the length of a string:

```
(define STRINGLENGTH (string-length TEXTSTRING))
STRINGLENGTH ;-> 16
```

To retrieve the k-th character of a string:

```
(define k 5)
(define kthCHARACTER (string-ref TEXTSTRING k))
kthCHARACTER ;-> #\i
```

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

Note:

The index counter starts from 0.

To retrieve substrings:

```
(define iStart 10)
(define iEnd    16)
(define SUBSTRING (substring TEXTSTRING iStart iEnd))
SUBSTRING ;-> "string"
```

To concatenate two strings:

```
(define RNAME "Region.")
(define QUALIFIER "Substrate")
(define REGIONNAME (string-append RNAME QUALIFIER))
REGIONNAME ;-> "Region.Substrate"
```

To convert a number (index) to a string:

```
(define RNAME "Region.")
(define INDEX 3)
(define REGIONNAME (string-append RNAME (number->string INDEX)))
REGIONNAME ;-> "Region.3"
```

Lists

Some Sentaurus Structure Editor commands accept lists as arguments. Lists can also be useful in device parameterization. To define a list:

```
(define ABCList  (list #\a #\b #\c #\d #\e))      ; List of characters
(define NUMList  (list 1 2 3 4 5 6))                ; List of numbers
(define MIXList  (list #\a 2 3.1415 "TCL"))        ; Mixed list
(define EMPTYList (list))                            ; Empty list
```

Note:

Declare an empty list first if you want to append to it, for example, in a `do` loop.

To reference a list as a whole, use the list name:

```
ABCList ;-> (#\a #\b #\c #\d #\e)
NUMList ;-> (1 2 3 4 5)
MIXList ;-> (#\a 2 3.1415 "TCL")
EMPTYList ;-> ()
```

To determine the number of elements in a list:

```
(define NumberOfElements (length ABCList))
NumberOfElements ;-> 5
```

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

To append an element to a list:

```
(define NewElement #\f)
(define ABCPlusList (append ABCList (list NewElement)))
ABCList ;-> (#\a #\b #\c #\d #\e #\f)
```

To concatenate two lists:

```
(define CombinedList (append ABCList NUMList))
CombinedList ;-> (#\a #\b #\c #\d #\e 1 2 3 4 5 6)
```

To reverse a list:

```
(define ReverseList (reverse NUMList))
ReverseList ;-> (6 5 4 3 2 1)
```

To reference the first element of a list:

```
(define FirstElement (car ABCList))
FirstElement ;-> #\a
```

or:

```
(define FirstElement (list-ref ABCList 0))
```

To reference all but the first element of a list:

```
(define RestOfList (cdr ABCList))
RestOfList ;-> (#\b #\c #\d #\e)
```

To reference the k -th element in a list:

```
(define k 3)
(define kthElement (list-ref ABCList k))
kthElement ;-> #\d
```

Note:

The numbering of list elements starts at 0.

To reference a sublist containing all but the first k elements, use:

```
(define k 3)
(define kRestOfList (list-tail ABCList k))
kRestOfList ;-> (#\d #\e)
```

Arithmetic Expressions

Some of the most useful arithmetic operators are:

- Addition, subtraction, multiplication, and division: +, -, *, /
- Trigonometric and related functions: sin, cos, tan, asin, acos, atan

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

- Exponential and related functions: `exp`, `log`, `sqrt`
- Raising `a` to the power of `b`: `expt a b`
- Rounding functions: `floor`, `ceiling`, `truncate`, `round`
- Maximum and minimum: `max`, `min`: `(max x1 x2 ...)`, `(min x1 x2 ...)`
- Other math functions defined in Sentaurus Structure Editor: `erf`, `erfc`
(These functions are not part of the standard Scheme language; they are defined locally in Sentaurus Structure Editor.)

Scheme uses the so-called polish-reverse notation for arithmetic expressions:

To define the variable `j` and assign it to `i+5`:

```
(define j (+ i 5))
j ;-> 8
```

To set the variable `myangle` to `sin(PI/2)`:

```
(define myangle (sin (/ PI 2)))
myangle ;-> 1.0
```

To set the variable `x` to `xo + R*cos(2*PI*fi/360)`:

```
(define xo 1)
(define R 0.2)
(define fi 30)
(define x (+ xo (* R (cos (/ (* 2 PI fi) 360)))))
x ;-> 1.17320508075689
```

Boolean Operations

True and false constants are denoted by `#t` and `#f`, respectively. Use `not` to invert a Boolean:

```
(not #t) ;-> #f
```

Numeric comparators are `=`, `>`, `>=`, `<`, `<=`, for example:

```
(= 1 2) ;-> #f
(> 1 2) ;-> #f
(>= 1 2) ;-> #f
(< 1 2) ;-> #t
(<= 1 2) ;-> #t
```

Use the `equal?` operator to test if two variables are equal:

```
(define a 1)
(define b "hello")
(equal? a b) ;-> #f
```

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

```
(set! b 1)
(equal? a b) ;->#t
```

To test two strings, you can also use `string=?:`

```
(define a "HELLO")
(define b "hello")
(string=? a b) ;-> #f
```

The `equal?` operator can also be used to test ACIS entities:

```
(define a (position 0 0 0))
(define b (position 0 0 0))
(equal? a b) ;-> #t
(set! b (position 1 0 0))
(equal? a b) ;-> #f
```

If Blocks

To create a simple If block:

```
(define val 0)
(if (= val 0)
    (begin
        (display "val is zero")(newline)
    )
) ;-> "val is zero"
```

To create an If-Else block:

```
(define val -1)
(if (< val 0) ; Test
    (begin ; Execute if condition is true
        (display "val is negative") (newline)
    )
    (begin ; Execute if condition is false
        (display "val is positive") (newline)
    )
) ;-> "val is negative"
```

To create an If-Elseif block:

```
(define val -1)
(cond
    ((= val 0) ; First test
        (begin ; Execute if first condition is true
            (display "val is zero") (newline)
        )
    )
    ((> val 0) ; Elseif test
        (begin ; Execute second condition is true
            (display "val is positive") (newline)
        )
    )
)
```

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

```
)  
)(else  
    (begin ; Execute if none of the conditions are true  
        (display "val is negative") (newline)  
    )  
) ;-> "val is negative"
```

Note:

To *comment out* a large block of comments in a script, enclose the block in a trivial If block:

```
(if #f (begin  
    <commands to be "commented out">  
)
```

Simple Do Loops

In this example, a simple Do loop is used to create a stack of five silicon squares:

```
(sde:clear)  
(define L 1) ; Side length of first square  
(define Y 0) ; Bottom y-coordinate of current square  
(do ( (i 0 (+ i 1)) ) ; i: Counter name; 0: initial value;  
     ; (+ i 1): incrementer  
     ( (= i 5) ) ; End Tester  
     (begin ; Body of loop  
         (define REGION (string-append "region." (number->string i)))  
         (sdegeo:create-rectangle  
             (position (* -0.5 L) Y 0.0)  
             (position (* 0.5 L) (+ Y L) 0.0) "Silicon" REGION)  
         (set! Y (+ Y L))  
         (set! L (* 0.75 L))  
     )  
)
```

Note:

Scheme Do loops are more flexible than Do loops in most other languages. This example illustrates a basic setup.

For Each Loops

In a For Each loop, an action is taken for each element in a list. In the following example, the For Each loop steps through the lists MATERIALS, WIDTHS, and HEIGHTS, and creates a rectangle for each list element:

```
(sde:clear)  
(define MATERIALS (list "GaAs" "AlGaAs" "AlAs" "InAlAs" "InAs"  
    "InAsP" "InP"))
```

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

```
(define WIDTHS      (list 2.0  1.7  1.5  1.3  1.1  1.0   0.5))
(define HEIGHTS     (list 1.0  0.25 0.1  0.25 0.3  0.45  1.0))
(define Y 0)

(for-each
  (lambda (MATERIAL WIDTH HEIGHT)           ; Names of local variables
    (begin                                     ; Body of the loop
      (define REGION (string-append "region." MATERIAL))
      (sdegeo:create-rectangle
        (position 0          Y          0.0)
        (position WIDTH (+ Y HEIGHT) 0.0) MATERIAL REGION)
      (set! Y (+ Y HEIGHT)))
    )
  ) MATERIALS WIDTHS HEIGHTS               ; Lists
)
```

Procedures

Procedures can be implemented to reuse common code sections or to structure your scripts. The following example creates a ‘Unit Cell’ structure consisting of a slab of silicon with a shallow trench isolation (STI)– like oxide trench at the side. The procedure takes the coordinate of the lower-left corner as an argument:

```
(sde:clear)
(define CreateUnitCell                      ; Name of procedure
  (lambda (Xo Yo)                         ; Argument list
    (begin                                     ; Body of procedure
      (sdegeo:set-default-boolean "ABA")
      (define SiREGION (string-append "R.Silicon."
        (number->string Xo) "." (number->string Yo)))
      (define OxREGION (string-append "R.Oxide."
        (number->string Xo) "." (number->string Yo)))
      (sdegeo:create-rectangle
        (position Xo Yo 0.0)
        (position (+ Xo 1.0) (+ Yo 1.0) 0.0) "Silicon" SiREGION)
      (sdegeo:create-rectangle
        (position Xo Yo 0.0)
        (position (+ Xo 0.25) (+ Yo 0.75) 0.0) "Oxide" OxREGION)
    )
  )
)
```

Now, a row of these unit cells can be created by calling the procedure for different arguments:

```
(CreateUnitCell 1 1) ; Procedure calls
(CreateUnitCell 2 1)
(CreateUnitCell 3 1)
```

Chapter 9: Working With Scheme and Scheme Extensions

Basic Scheme Programming

The next example shows a procedure, which returns a value. The variable containing the value of interest is simply referenced in the last statement of the procedure:

```
(define convert-degree-to-radians
  (lambda (DEG)
    (define pi (acos -1.0))
    (define RAD (* DEG (/ PI 180.0)))
    RAD
  )
)

(define DEG 90)
(define RAD (convert-degree-to-radians 90))
RAD ;-> 1.5707963267949
```

Scheme procedures, or any other code segments, can be saved in a library file, for example, `util.scm` and loaded into a Scheme script with:

```
(load "util.scm")
```

System Calls

To call an external UNIX or TCAD Sentaurus utility:

```
(system:command "<Unix command string>" )
```

For example, to remove the temporary file `test_bnd.tdr`, use:

```
(system:command "rm -f test_bnd.tdr" )
```

Error Signaling to Sentaurus Workbench

If the command `system:command` is used in a Sentaurus Structure Editor script and you want to signal the status of the executed command back to Sentaurus Workbench, use the following solution:

```
(define res (system:command "Mycommand . . ."))
(if (not (equal? res 0))
  (begin
    (sde:error "Mycommand failed . . .")
  )
)
```

If `system:command` returns an error (anything else than 0), Sentaurus Workbench will pick up the error signal, and the executed node will be shown as red.

10

Geometric Query Functions

This chapter describes the boundary representation method on which Sentaurus Structure Editor is based and introduces several support functions to query and debug the model.

Entity IDs and Attributes

Geometry-related operations in Sentaurus Structure Editor are based on ACIS, which is a boundary representation geometry kernel. Boundary representation uses a complex data structure describing the geometry. The data structure of a geometric model can be accessed through a topology graph, which contains the building entities (such as edges, faces, and shells) of the device, and shows the relationships and connections between these entities. A typical 3D body is described with respect to a lump list (disjoint parts of the body). Each lump contains a shell list and each shell is described with respect to a face list. Each 3D body is composed of faces. A shell is a connected face list. For example, a cube is defined with respect to six planar faces, and the six faces define one shell. Similarly, a cube with an internal void is composed of two shells and each shell is defined by a face list.

Faces are described with respect to a loop and an edge list. A face is described with respect to its boundary edges. For example, a rectangle is defined by four linear edges. A circular sheet body is defined by a circular edge. A connected edge list is called a *loop*. Simply connected faces have one loop. Faces with internal holes have multiple loops. For example, a circular sheet with an internal rectangular hole is composed of two loops. The outer loop contains one circular edge, while the internal loop contains four linear edges. A linear edge is defined by two vertices.

It is usually not necessary to consider the topological representation of defined bodies when using typical GUI operations or the Scheme interface. For certain geometric operations, however, for example, rounding a set of edges of a 3D body, it is necessary to refer to entities from the underlying topology representation. These edges must be identified by using either the GUI or Scheme interface.

Each generated device is a collection of several different bodies. In two dimensions, these bodies are solid bodies with zero volume (*non-manifold solid bodies*). In three dimensions, these bodies have a finite, positive volume (*manifold solid bodies*). When the geometric model is generated using either the GUI actions or `sdegeo` Scheme functions, Sentaurus

Chapter 10: Geometric Query Functions

Topological Entity Types

Structure Editor assigns several different attributes to these bodies. Two important attributes are the ‘material’ attribute and the ‘region’ attribute.

An attribute is a general-purpose data entity that attaches to other entities to record user-defined or some other (internal) information. Attribute objects are saved and restored as part of the model when the model is saved in the native ACIS .sat or .sab format. If the geometric model is exported to the TDR boundary format, only the TCAD-specific attributes (contacts, and material and region names) are saved to the boundary file.

The material attribute is a DATEX material (one of the materials defined in the `datexcodes.txt` file). The region attribute is a string that identifies a region to which a particular body belongs. Each body within a region must have the same ‘material’ attribute.

The boundary file description contains some other restrictions to the use of the region attribute (that is, disjoint bodies cannot have the same region name). Since Sentaurus Structure Editor can generate bodies with multiple lumps, these lumps are separated automatically when exporting the structure to the boundary file in TDR format. In addition, **Edit > Separate Lumps** or the Scheme extension `sde:separate-lumps` can be used to separate lumps explicitly.

Under typical circumstances, when the automatic region-naming feature is switched on, the geometry generation and editing actions accessible from the GUI ensure that region-naming is consistent with the requirements of the boundary format. When using direct Scheme extensions to generate the geometry, you can generate structures in which the region and material properties of bodies are inconsistent with these additional restrictions imposed by the TDR boundary format. In this case, you must set region names and material properties explicitly to resolve any inconsistencies.

During model generation, the geometry engine assigns a unique entity number to each generated body. When a Scheme function needs an entity ID or a list of entity IDs, refer to these automatically assigned entity IDs.

Topological Entity Types

The valid entity types are:

body	A body is a topological entity of the highest level. It can be a wire body, solid body, or mixed body. Wire bodies contain wires, coedges, edges, and vertices. Solid bodies contain lumps, shells, subshells, faces, loops, coedges, edges, and vertices. Mixed (solid and wire) bodies contain lumps, shells, subshells, faces, loops, coedges, edges, vertices, and wires. All body objects are saved and restored as a part of the model.
------	---

Chapter 10: Geometric Query Functions

Topological Entity Types

edge	An edge is a topological entity associated with a curve. It is bound by one or more vertices, and has one vertex at each end. If the reference at one or both ends is <code>NULL</code> , the edge is unbound in that direction. Each edge contains a record of its sense (<code>FORWARD</code> or <code>REVERSED</code>) relative to its underlying curve. All edge objects are saved and restored as part of the model.
face	A face is a topological entity that represents a portion of a geometric surface. One or more loops of edges bound a face, which can be open or closed. A face with no loops occupies the entire surface, finite or infinite, on which the face lies. Therefore, a face can represent an infinite plane or a complete sphere. Each face records its sense relative to its underlying surface (same sense or opposite sense). All face objects are saved and restored as part of the model.
loop	A loop is a topological entity that represents a connected portion of the boundary of a face. Loops can be open or closed. A loop can comprise a group of coedges connected in a branched arrangement or in a simple, open chain. A loop can be a coedge shrunk to a single vertex. All loop objects are saved and restored as a part of the model.
lump	A lump is a topological entity that represents a connected 3D (solid) or 2D (sheet) region. A body can contain zero or more lumps. Each lump represents a disjoint set of points. One lump is completely enclosed inside the void of another solid lump. Each lump must have at least one shell. All lump objects are saved and restored as a part of the model.
shell	A shell is a topological entity consisting of a set of connected faces. All faces are connected along edges or vertices. A shell represents a sheet region or bounds a solid region or both. A shell that bounds a solid region is entirely peripheral or void or neither. All shell objects are saved and restored as a part of the model.
vertex	A vertex is a topological entity representing the end of one or more edges. <i>Vertex</i> refers to a point object in space and to the edges that it bounds. Other edges are found by following pointers through coedges. All vertex objects are saved and restored as a part of the model.
wire	A wire is a topological entity that is a collection of edges and vertices. Wires typically represent profiles, construction lines, and center lines of swept shapes. Wires can also represent wireframes that form shells when surfaced. All wire objects are saved and restored as a part of the model.
wire body	A wire body is a topological entity that is a body consisting of wires (as opposed to lumps). Wire bodies contain wires, loops, coedges, edges, and vertices. All wire-body objects are saved and restored as a part of the model.

Selecting Geometric Objects

This section describes how to select geometric objects.

Graphic-Supported Object Selection

Several different operations require the identification of geometric objects (entities). Either the GUI or direct Scheme functions can be used to identify entities. The GUI toolbar buttons (the Select button) can be used for interactive entity selection (see [Selecting Entities on page 60](#)). The default entity that can be selected is **body**. The selection criteria can be set to a different entity type by right-clicking in the view window and setting the selection to the required type, or by selecting the corresponding level in the Selection Level list.

The corresponding Scheme command is

```
(sde:set-selection-level "body" | "face" | "edge" | "vertex" | "other")
```

The function `(sde:selected-entities)` returns a list of the selected (highlighted) entity IDs. Note that this Scheme function always returns a list, even if only a single entity is selected. For example, this Scheme function can return:

```
(sde:selected-entities) ; -> (#[vertex 8 1])
```

Note:

This list contains only a single entry.

Use `(car (sde:selected-entities))` to return only the first (and, here, the only) item on this list:

```
(car (sde:selected-entities)) ; -> #[vertex 8 1]
```

You also can use the Topology Browser to find entity IDs (see [Visualizing the Internal Entity Representation on page 74](#)).

Script-Based Object Selection

To select all geometric bodies in the structures, use the Scheme command:

```
(part:entities (filter:type "solid?"))
```

This function returns refinement/evaluation (Ref/Eval) windows as well. To distinguish between the true geometric bodies and the Ref/Eval windows, use the following Scheme functions.

Chapter 10: Geometric Query Functions

Selecting Geometric Objects

For all true geometric bodies, use:

```
(get-body-list)
```

For all Ref/Eval windows, use:

```
(get-drs-list)
```

For example:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 1.0 1)
    "Silicon" "region_1")
; -> #[body 5 1]
(sdegeo:create-cuboid (position 0.3 -0.2 0.0) (position 0.7 0.0 1)
    "SiO2" "region_2")
; -> #[body 6 1]
(sdedr:define-refeval-window "RefEvalWin_1" "Cuboid"
    (position 0.3 0.0 0) (position 0.7 0.2 1))
; -> #[body 7 1]
(sdedr:define-refeval-window "RefEvalWin_2" "Rectangle"
    (position -0.2 0.0 0) (position 0.3 0.0 1))
; -> #[body 8 1]

(part:entities (filter:type "solid?"))
; -> (#[body 5 1] #[body 6 1] #[body 7 1] #[body 8 1])
(get-body-list)
; -> (#[body 5 1] #[body 6 1])
(get-drs-list)
; -> (#[body 7 1] #[body 8 1])
```

Note:

To export the boundary in TDR format use the Scheme command:

```
(sdeio:save-tdr-bnd (get-body-list) "filename.tdr")
```

To find the ID of an entity of a given type at a given location, use the Scheme commands:

```
(find-body-id position)
(find-face-id position)
(find-edge-id position)
(find-vertex-id position)
(find-material-id material)
(find-region-id region-name)
```

Each of these commands returns an entity list containing the entity IDs of all entities that satisfy the search criteria. If no entity is found, an empty list is returned. The `find` functions require a specification of the exact position for the required entity. For example:

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1)
    "Silicon" "region_1")
(find-body-id (position 0.5 0.5 0.5)) ; -> (#[body 5 1])
(find-face-id (position 0 0 0.5))) ; -> (#[face 29 1] #[face 28 1])
(find-edge-id (position 0 0 0)))
; -> (#[edge 14 1] #[edge 12 1] #[edge 11 1])
```

Chapter 10: Geometric Query Functions

Finding Region Names and Material Properties

```
(find-vertex-id (position 1 1 1)) ; -> (#[vertex 19 1])
(find-material-id "Silicon") ; -> (#[body 5 1])
(find-region-id "region_1") ; -> (#[body 5 1])
```

These Scheme commands always return a list, even if the list contains only a single entry. Some Scheme commands, however, expect a single entity as an argument, not a list. Use the Scheme function `car` to return only the first element of the list. For example:

```
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Silicon" "region_1")
(sdegeo:move-vertex (car (find-vertex-id (position 0 0 0)))
    (position -1 -1 0))
```

The function `find-material-id` can be used to find the entity numbers of all bodies having the specified DATEX material as the material attribute.

The function `find-region-id` can be used to find the entity numbers of all bodies having the specified region as the region attribute.

If the exact location is not known use the `sdegeo:find-closest` Scheme commands:

```
(sdegeo:find-closest-face position)
(sdegeo:find-closest-edge position)
(sdegeo:find-closest-vertex position)
```

For example:

```
(sdegeo:create-cuboid (position 0.1 -0.1 0.1) (position 1.1 0.9 0.9)
    "Silicon" "region_1")
(sdegeo:find-closest-face (position 0 0 0)) ; -> (#[face 6 1] . 0.1414)
(sdegeo:find-closest-edge (position 0 0 0.5)) ; -> (#[edge 7 1] . 0.1414)
(sdegeo:find-closest-vertex (position 1 1 1)) ; -> (#[vertex 8 1] . 0.1732)
```

Finding Region Names and Material Properties

When the ID of a certain topological entity is identified, the entity can be further investigated. As previously mentioned, each body has several different attributes attached. The function `generic:get` can be used to query the attributes. For example, the function `(generic:get entity_id "material")` returns the DATEX material of the selected body. The function `(generic:get entity_id "region")` returns the attached region name:

```
(define BODY (sdegeo:create-cuboid (position 0 0 0) (position 1 1 1)
    "Silicon" "region_1"))
(generic:get BODY "material") ; -> "Silicon"
(generic:get BODY "region") ; -> "region_1"
```

Several functions of Sentaurus Structure Editor support entity debugging. The function `sde:info` lists the entity IDs, and material and region names. The argument list of `sde:info` is an entity, a list of entities, or 'all.' For example:

```
(sde:info "all")  
  
Entity Id;   Material name;   Region name;  
#[body 5 1]  Silicon          region_1
```

Automatic Region-Naming

Each body that is to be written to a TDR boundary file must have two attached attributes: the material attribute and region attribute. If the automatic region-naming option (choose **Draw > Auto Region Naming**) is switched on, each newly created region is automatically assigned a unique region name. If the `sdegeo` Scheme extensions are used, the region names must be specified in the argument list.

The automatic region-naming option can also be set from Scheme. The function `(sdegeo:get-auto-region-naming)` can be used to obtain the status of the option. This function returns either `#t` or `#f`.

The function `(sdegeo:set-auto-region-naming #t | #f)` can be used to switch on or off the feature. When an explicit `sdegeo` call is made to generate a new body, the region counter must be set automatically to the next available region number for a further GUI body generation.

The automatic region-naming option assigns the following region name to created bodies:

```
"region_" + the region counter
```

The region counter starts from 1 and increases automatically. The `sde:set-region-counter` Scheme extension can be used to set the region counter explicitly. For example, to set the counter to 12:

```
(sdegeo:get-region-counter) ; -> 1  
(sdegeo:set-region-counter 12)
```

When a TDR boundary file is loaded and regions follow the abovementioned naming convention, the counter is set automatically to the next available integer.

Regions with multiple lumps are separated and renamed automatically by default before the TDR boundary output is created. For example, if the original body has a region attribute "region_1" and the region has three lumps, the original region will be separated into three bodies with the region names "region_1_lump_1", "region_1_lump_2", and "region_1_lump_3".

To switch off automatic region-renaming, set the `use-unique-region-names` global Scheme variable to `#f`.

Chapter 10: Geometric Query Functions

Finding Vertex Positions

Choose **Edit > Separate Lumps** to invoke the lump separation explicitly.

The corresponding Scheme command is:

```
(sde:separate-lumps)
```

List of Supported Materials

Sentaurus Structure Editor uses the default `datexcodes.txt` file to load a list containing the allowable materials. This DATEX material list shows all of the defined DATEX materials and always shows the active DATEX material. If a GUI action needs a DATEX material as input and the action does not contain a method to select it, it will always use the active DATEX material shown in the DATEX material list.

The Scheme extension `(sde:get-default-material)` returns the active DATEX material and the Scheme extension `sde:set-default-material` can be used to change the active DATEX material.

For example, to change the active material to `PolySilicon`:

```
(sde:set-default-material "PolySilicon")
```

Note:

To introduce custom materials to Sentaurus Structure Editor, create a modified copy of the `datexcodes.txt` file in the current working directory (see *Utilities User Guide*, Chapter 1).

Finding Vertex Positions

When the entity ID of a vertex (`vertex_id`) is found (`vertex:position vertex_id`), it can be used to return the position of a vertex.

For example:

```
(define BODY (sdegeo:create-rectangle (position 0.0 0.0 0)
                                         (position 1.0 1.0 0) "Silicon" "region_1"))
(define i 1)
(for-each
  (lambda (VERTEX)
    (begin
      (display " [") (display i) (display "]:: ")
      (display " X=") (display (position:x
                                (vertex:position VERTEX)))
      (display ",Y=") (display (position:y
                                (vertex:position VERTEX)))
      (set! i (+ i 1))
    )
  )
(entity:vertices BODY)
```

Chapter 10: Geometric Query Functions

Vertex–Vertex Distance

```
)  
(newline)  
; -> [1]: X=0,Y=0 [2]: X=1,Y=0 [3]: X=1,Y=1 [4]: X=0,Y=1
```

Vertex–Vertex Distance

To determine the distance between two vertices:

1. Choose **Edit > 2D Edit Tools > Vertex-to-Vertex Distance**, or click the corresponding toolbar button.
2. Select the first vertex and then the second vertex in the view window.

The corresponding distance is displayed in the command-line window.

The corresponding Scheme command is:

```
(sdegeo:distance (list vertex vertex))
```

For example:

```
(sdegeo:create-cuboid (position 0.0 0.0 0.0) (position 1.0 1.0 1.0)  
    "Silicon" "region_1"))  
(sdegeo:distance (list  
    (car (find-vertex-id (position 0.0 0.0 0.0)))  
    (car (find-vertex-id (position 1.0 1.0 1.0)))))  
; -> 1.73205
```

Debugging Topological Entities

When a geometric entity (for example, a body) is generated, a unique entity ID (an integer) is assigned to the body. This unique entity ID can be used to refer to that particular entity in further operations. The command `(part:entities)` lists all defined entities.

Several Scheme functions can be used to debug an entity. The Scheme function `(entity:debug entity [0|1|2|3|4])` can be used for basic debugging. The argument `entity` specifies the entity to be queried where `level` is an optional argument that controls the amount of the debugging information:

- level 0 Writes no output and only returns the entity type.
- level 1 Writes only data specific to the given entity.
- level 2 Writes additional information for some entity types. For example, if the entity is an EDGE, the coordinates of the endpoints and the curve data are also written. This extension writes only data specific to the given entity.

Chapter 10: Geometric Query Functions

Finding Edges, Faces, and Other Elements of a Body

level 3 Writes the size of the entity and all associated entities.

level 4 Writes full information for the entity and all associated entities.

For example:

```
(define BODY (sdegeo:create-rectangle
    (position 0.0 0.0 0.0) (position 1.0 1.0 0.0) "Silicon" "region_1"))
(entity:debug BODY 0)
; -> "solid body"
```

Finding Edges, Faces, and Other Elements of a Body

Several Scheme functions can be used to find edges, faces, and other elements of a body:

entity:edges	Returns the edge list of the specified entity or entities.
entity:faces	Returns the face list of the specified entity or entities.
entity:loops	Returns the loop list of the specified entity or entities.
entity:lumps	Returns the lump list of the specified entity or entities.
entity:shells	Returns the shell list of the specified entity or entities.
entity:vertices	Returns the vertex list of the specified entity or entities.

For example:

```
(define BODY (sdegeo:create-cuboid
    (position 0.0 0.0 0.0) (position 1.0 1.0 1.0)
    "Silicon" "region_1"))

; All faces of the body:
(define FACES (entity:faces BODY))
; FACES: (#[face 6 1] #[face 7 1] #[face 8 1] #[face 9 1] ...)

; All edges of the first face:
(define EDGES (entity:edges (car FACES)))
; EDGES: (#[edge 12 1] #[edge 13 1] #[edge 14 1] #[edge 15 1])

; All vertices of the first edge of the first face:
(define VERTICES (entity:vertices (car EDGES)))
; VERTICES: (#[vertex 16 1] #[vertex 17 1])

; All vertices of the body:
(define ALL_VERTICES (entity:vertices BODY))
```

Chapter 10: Geometric Query Functions

Bounding Box Query

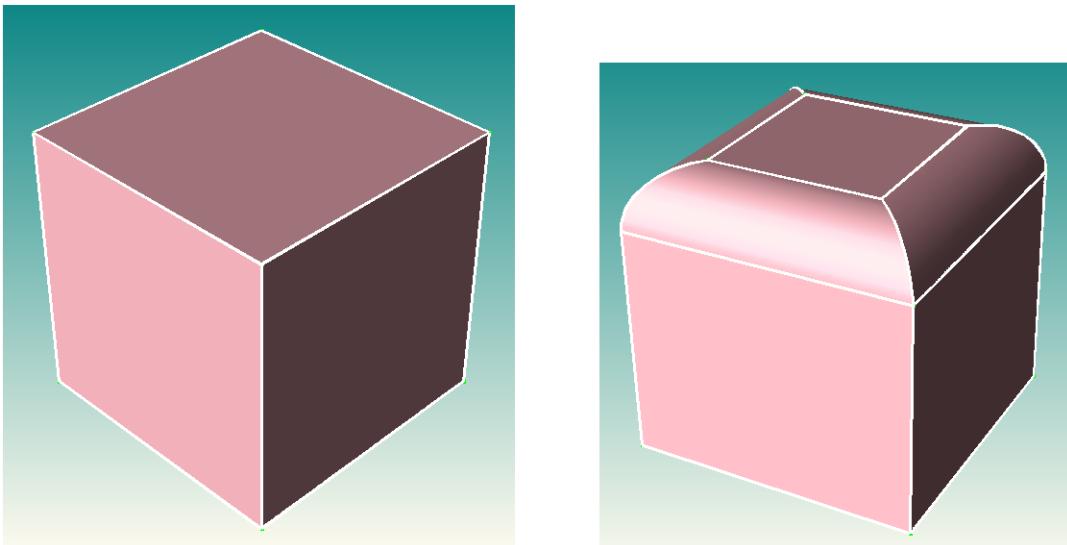
```
; ALL_VERTICES: (#[vertex 16 1] #[vertex 17 1] #[vertex 18 1] ...)
```

These functions can be used, for example, to round all edges associated with a certain face of an object. Here, a simple cube is created, then the top face is identified using the function `find-face-id` (see [Script-Based Object Selection on page 323](#)).

For this face, all edges are identified using the `entity:edges` function, and finally these edges are rounded using the `sdegeo:fillet` function:

```
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
    "Silicon" "region_1")
(define top_face (find-face-id (position 5 5 10)))
(define top_edges (entity:edges top_face))
(sdegeo:fillet top_edges 2)
```

Figure 132 Example of finding geometric entities



Bounding Box Query

To find the bounding box of an entity or a group of entities, use the Scheme command:

```
(entity:box entity|entity-list)
```

For example:

```
(define BODY1 (sdegeo:create-cuboid (position 0.0 0.0 0.0)
    (position 1.0 1.0 1) "Silicon" "region_1"))
(define BODY2 (sdegeo:create-cuboid
    (position 0.3 -0.2 0.0) (position 0.7 0.0 1) "SiO2" "region_2"))
(define BODY1BBOX (entity:box BODY1))
; BODY1BOX: (#[ position 0 0 0 ] . #[ position 1 1 1 ])
```

Chapter 10: Geometric Query Functions

Scheme Functions for Entity Queries

```
(define GlobalBBBOX (entity:box (get-body-list)))
; GlobalBBBOX: (#[ position 0 -0.2 0 ] . #[ position 1 1 1 ])
(display " Ymin=") (display (position:y (car GlobalBBBOX)))
(display " Ymax=") (display (position:y (cdr GlobalBBBOX)))
(newline)
; -> Ymin=-0.2 Ymax=1
```

Scheme Functions for Entity Queries

[Table 39](#) lists the Scheme functions that can be used to query entities and to find the type of the selected entities.

Table 39 Scheme functions for entity queries

Scheme function	Description
body?	Determines whether a Scheme object is a body.
curve:circular?	Determines whether a Scheme object is a curve–circular.
curve:elliptical?	Determines whether a Scheme object is a curve–elliptical.
curve:linear?	Determines whether a Scheme object is a linear curve.
curve?	Determines whether a Scheme object is a curve.
edge:circular?	Determines whether a Scheme object is a circular edge.
edge:curve?	Determines whether a Scheme object is a curve–edge.
edge:elliptical?	Determines whether a Scheme object is an elliptical edge.
edge:linear?	Determines whether a Scheme object is a linear edge.
edge:spline?	Determines whether a Scheme object is a spline–edge.
edge?	Determines whether a Scheme object is an edge.
entity?	Determines whether a Scheme object is an entity.
face:conical?	Determines whether a Scheme object is a conical face.
face:cylindrical?	Determines whether a Scheme object is a cylindrical face.
face:planar?	Determines whether a Scheme object is a planar face.
face:spherical?	Determines whether a Scheme object is a spherical face.

Chapter 10: Geometric Query Functions

Scheme Functions for Entity Queries

Table 39 Scheme functions for entity queries (Continued)

Scheme function	Description
face:spline?	Determines whether a Scheme object is a face-spline.
face:toroidal?	Determines whether a Scheme object is a toroidal face.
face?	Determines whether a Scheme object is a face.
loop:external?	Determines whether a loop is internal or external.
loop?	Determines whether a Scheme object is a loop.
lump?	Determines whether a Scheme object is a lump.
position?	Determines whether a Scheme object is a position.
shell?	Determines whether a Scheme object is a shell.
solid:closed?	Determines whether a Scheme object is a closed solid.
solid:manifold?	Determines whether a Scheme object has a positive volume.
solid?	Determines whether a Scheme object is a solid.
vertex?	Determines whether a Scheme object is a vertex.
wire:closed?	Determines whether a Scheme object is a closed wire.
wire:planar?	Determines whether a Scheme object is a planar wire.
wire?	Determines whether a Scheme object is a wire.
wire-body?	Determines whether a Scheme object is a wire body.

For example:

```
(define BODY (sdegeo:create-rectangle
  (position 0 0 0) (position 1 1 0) "Silicon" "region_1" ))
(body? BODY)
; -> #t
(face? BODY)
; -> #f
```

11

Miscellaneous Utilities

This chapter outlines additional features in Sentaurus Structure Editor.

Background Image Loader

Sentaurus Structure Editor supports the creation of device geometries by digitizing images, for example, transmission electron micrographs (TEMs). These images can be loaded as a background image into the view window. [Table 40](#) lists the relevant Scheme extensions.

Note:

This feature is accessible only using Scheme commands in the command-line window.

The image loader supports only images in the older GIF format, GIF87a. Most current graphic programs use the GIF format GIF89a. To check the format version of a GIF file, look at the first line of the file using, for example, the UNIX command:

```
> head --lines=1 filename.gif
```

The first six characters are either GIF87a or GIF89a. If the GIF file is in GIF89 format, use an older graphics utility program such as xv 3.10a (a shareware program written by John Bradley) to convert the GIF file to GIF87a.

Table 40 Scheme extensions for manipulating background image

Scheme extension	Definition	Arguments
(sde:create-bg-image filename hsize vsizE)	Loads an image file (in GIF format)	filename: STRING hsizE: horizontal size, REAL vsizE: vertical size, REAL
(sde:show-bg-image)	Shows the image	None
(sde:hide-bg-image)	Hides the image	None
(sde:delete-bg-image)	Removes the image from memory	None

Chapter 11: Miscellaneous Utilities

User-Defined Dialog Boxes

Table 40 Scheme extensions for manipulating background image (Continued)

Scheme extension	Definition	Arguments
(sde:bg-image-transparency value)	Sets the transparency of the image (0-1)	transparency value, REAL (0-1)

To convert an image from the GIF89a format to GIF87a format:

1. Open the GIF89a file in a graphics program.
2. Start xv, and grab the opened GIF89a image with xv.
3. Save the image in GIF format.

xv will save the captured image using the older GIF87a format.

To load an image into Sentaurus Structure Editor and scale it to 0.2 μm × 0.2 μm:

```
(sde:create-bg-image "TEM.gif" 0.2 0.2)  
(sde:show-bg-image)
```

To digitize a region of the device structure, follow the procedure for creating arbitrary polygons (see [Drawing Polygons on page 82](#)).

User-Defined Dialog Boxes

User-defined dialog boxes can be created to assist with the generation of parametric devices. This feature is accessible only from Scheme.

This feature can be used to create a convenient central template library for company-wide use. Users of this central template library would only need to enter the necessary parameters in the custom dialog box and the structure is created automatically.

To create a user-defined dialog box, first declare the dialog box, using `sde:create-dialog`. Then, input fields, which will serve as the argument list of the associated Scheme function, must be defined using `sde:dialog-add-input`. An optional bitmap icon can also be added to the dialog box, using `sde:dialog-add-pixmap`.

Note:

Only bitmaps in BMP format are supported.

The **OK** button of the dialog box can be attached to execute an already defined Scheme script using the `sde:dialog-ok-command` command. The dialog box can be displayed using `sde:dialog-show` and can be removed from memory using `sde:dialog-delete`.

Chapter 11: Miscellaneous Utilities

User-Defined Dialog Boxes

The following Scheme commands can be used to create user-defined dialog boxes:

```
(define dialog-id (sde:create-dialog dialog-label))
(sde:dialog-add-input dialog-id varname labeltext inputtype
    [defaultvalue])
(sde:dialog-add-pixmap dialog-id bmpfilename)
(sde:dialog-ok-command dialog-id schemefnname argument-list)
(sde:dialog-show dialog-id)
(sde:dialog-delete dialog-id)
```

Detailed syntax descriptions of these commands can be found in [Appendix A on page 339](#).

To create and launch a user-defined dialog box:

1. Define a Scheme function that the dialog box will execute when the **OK** button is clicked.
2. Define the dialog box, create a bitmap image that shows all the necessary parameters, and add the bitmap image and the necessary parameters to the dialog box.
3. Launch the dialog box.

Example: Defining a Dialog Box

The following example illustrates how to define and use a dialog box.

Step 1: Define a Scheme Function That the Dialog Box Executes

```
(define create-3d-mosfet
  (lambda (Lsub Wsub Hsti Wsti Toxi Lgat)
    (define Hsil 1.0) (define Hpol 0.3)
    (define Xgat1 (* 0.5 (- Lsub Lgat)))
    (define Xgat2 (* 0.5 (+ Lsub Lgat)))
    (sdegeo:create-cuboid (position 0.0 0.0 0.0)
      (position Lsub Wsub Hsil) "Silicon" "R.Substrate")
    (sdegeo:set-default-boolean "ABA")
    (sdegeo:create-cuboid (position 0.0 (- Wsub Wsti) (- Hsil Hsti))
      (position Lsub Wsub Hsil) "Oxide" "R.STI")
    (sdegeo:create-cuboid (position Xgat1 0.0 Hsil)
      (position Xgat2 Wsub (+ Hsil Toxi)) "Oxide" "R.Gox")
    (sdegeo:create-cuboid (position Xgat1 0.0 (+ Hsil Toxi))
      (position Xgat2 Wsub (+ Hsil Hpol)) "PolySilicon" "R.Poly")
  )
)
```

Step 2: Define and Configure the Dialog Box

```
(define mos-3d-dialog (sde:create-dialog "Simple 3D MOSFET"))
(sde:dialog-add-pixmap mos-3d-dialog "MOS3D.bmp")
(sde:dialog-add-input mos-3d-dialog "Lsub" "Substrate Length: Lsub="
  "real" 1.2)
```

Chapter 11: Miscellaneous Utilities

Starting Sentaurus Structure Editor With User-Defined Variables

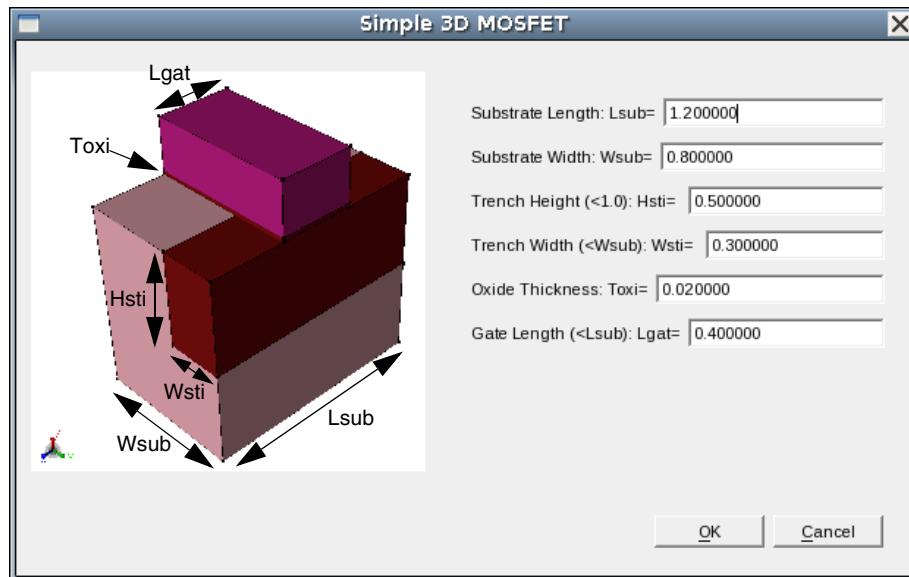
```
(sde:dialog-add-input mos-3d-dialog "Wsub" "Substrate Width: Wsub="  
    "real" 0.8)  
(sde:dialog-add-input mos-3d-dialog "Hsti"  
    "Trench Height (<1.0): Hsti= " "real" 0.5)  
(sde:dialog-add-input mos-3d-dialog "Wsti"  
    "Trench Width (<Wsub): Wsti= " "real" 0.3)  
(sde:dialog-add-input mos-3d-dialog "Toxi" "Oxide Thickness: Toxi="  
    "real" 0.02)  
(sde:dialog-add-input mos-3d-dialog "Lgat"  
    "Gate Length (<Lsub): Lgat=" "real" 0.4)  
(sde:dialog-ok-command mos-3d-dialog "create-3d-mosfet"  
    "Lsub Wsub Hsti Wsti Toxi Lgat")
```

Step 3: Launch the Dialog Box

```
(sde:dialog-show mos-3d-dialog)
```

Figure 133 shows the dialog box that was defined and created in Step 2, and launched in Step 3.

Figure 133 User-defined dialog box



Starting Sentaurus Structure Editor With User-Defined Variables

Scheme variables can be specified at the command line. When Sentaurus Structure Editor is launched, these variables are accessible from the Scheme command-line window (or in batch mode, the variables are defined for script execution).

Chapter 11: Miscellaneous Utilities

User-Defined GUI Interactions

In this way, a Scheme script can be started in such a way that the argument variables are passed to the script from the command line.

The syntax is:

```
-var varName=varValue
```

Neither `varName` nor `varValue` can contain any space characters (that is, the `varName=varValue` construct should be seen as one argument). `varValue` can be either a string or number. For example:

```
> sde -l cmdinputdemo.scm -e -var mw=10 -var mh=5
```

where `cmdinputdemo.scm` is defined as:

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position mw mh 0)
  "Silicon" "region_1")
(sdeio:save-tdr-bnd (get-body-list) "testarginput.tdr")
```

User-Defined GUI Interactions

This section discusses user-defined GUI interactions.

Dialog Boxes for Obtaining Values

The following functions can be used to interact with the GUI and to receive user input:

```
(sde:gui-get-integer entry-field-label default-integer title)
(sde:gui-get-real entry-field-label default-real title)
(sde:gui-get-string entry-field-label default-string title)
```

For example, to display a custom dialog box with the title **CMOS** and a field **Total Width**, which expects a real number that defaults to 10:

```
(define Wtot (sde:gui-get-real "Total Width" 10 "CMOS"))
```

The entered value is stored here in the Scheme variable `Wtot`.

GUI Actions for Obtaining Positions

Simple support functions are also provided to perform interactive model manipulation.

To select a position from the view window:

```
(define MyPosition (sde:pick-point-on-wp))
```

Chapter 11: Miscellaneous Utilities

User-Defined GUI Interactions

After this Scheme command is given, move the mouse in the view window and click the position of interest. The resulting position is stored here in the Scheme variable `MyPosition`.

To select a pair of positions from the view window:

```
(define MyPositionPair (sde:pick-two-points-on-wp))
```

After this Scheme command is given, move the mouse in the view window to the first point of interest, drag to the second point of interest and release the left mouse button. The resulting list of two position is stored here in the Scheme variable `MyPositionPair`.

Message Boxes

To post a message box:

```
(sde:post-message message)
```

For example:

```
(sde:post-message "This is a message\n from SDE")
```

The dialog box has an **OK** button that must be clicked to continue. No new-line character is needed. Multiple lines must be separated by "`\n`".

A

Commands

This appendix describes the supported Scheme commands, functions, and extensions.

Presentation of Commands

Each Scheme command, function, or extension accepts zero or more arguments, and can return a value. The type of arguments and the return type are written in uppercase.

If a Scheme list of a certain data type is used, then it is indicated as <DATATYPE> LIST, for example, STRING LIST or ENTITY LIST.

See [Chapter 9 on page 309](#) for an explanation of the data types used in Sentaurus Structure Editor.

Renamed Commands and Obsolete Commands

The following Scheme commands have been renamed or replaced to follow a more consistent Scheme naming convention. The old command names still work but are replaced internally with the new command name. The argument lists are the same, and there are no compatibility issues.

Note:

Do not use old command names in command files.

Old Scheme command name	New Scheme command name
dnce	sdegeo:dnce
sde:selected-refinements	sde:selected-refeval-windows
sdeaxisaligned:set-parameters	sdesnmesh:axisaligned
sdedelaunizer:set-parameters	sdesnmesh:delaunizer

Appendix A: Commands

Presentation of Commands

Old Scheme command name	New Scheme command name
sdedelaunizer:set-tolerance	sdesnmesh:delaunizer-tolerance
sdedr:define-refinement-window	sdedr:define-refeval-window
sdedr:delete-refinement-window	sdedr:delete-refeval-window
sdedr:redefine-refinement-window	sdedr:redefine-refeval-window
sdefloops:begin	sdesp:begin
sdefloops:define-step	sdesp:define-step
sdefloops:finalize	sdesp:finalize
sdefloops:restore-state	sdesp:restore-state
sdefloops:tag-initial	sdesp:tag-initial
sdegeo:define-2d-contact	sdegeo:set-contact
sdegeo:define-3d-contact	sdegeo:set-contact
sdegeo:fillet-edges	sdegeo:fillet
sdegeo:split-solid	sde:split-solid
sdeinterpolate:set-parameters	sdesnmesh:interpolate
sdeio:read_dfise_mask	sdeio:read-dfise-mask
sdenoffset:create-noffset-block	sdedr:offset-block
sdenoffset:create-noffset-interface	sdedr:offset-interface
sdesnmesh:replace-tensor-block	sdesnmesh:tensor
sdesnmesh:replace-tools-block	sdesnmesh:tools
sdesnmesh:set-iocontrols	sdesnmesh:iocontrols

Appendix A: Commands

Presentation of Commands

The following Scheme commands are obsolete and no longer supported. There are no replacements for these commands. If you execute a script containing any of these commands, then the script execution terminates, and an error message is written to the .log and .log.err files.

Note:

Remove these obsolete commands from scripts.

Obsolete Scheme command
sdedr:offset-boundary
sdedr:offset-isoline
sdenoffset:create-boundary
sdenoffset:create-global
sdenoffset:create-isoline

Appendix A: Commands

afm-smooth-layers

afm-smooth-layers

This Scheme extension creates a layered lens structure with a texture map. The texture map is defined in a CSV file.

Syntax

```
(afm-smooth-layers fname z0 zplanar layerregion layermaterial  
layerthickness)
```

Returns

BOOLEAN

Arguments

Argument	Description
fname	The name of the input file. Argument type: STRING
z0	The bottom z-coordinate. Argument type: REAL
zplanar	The top planar z-coordinate. Argument type: REAL
layerregion	List of region names. Argument type: STRING LIST
layermaterial	List of DATEX materials. Argument type: DATEXMAT LIST
layerthickness	Specifies the thickness of each layer. Argument type: REAL LIST

Examples

```
(define n 10)  
(define a (* 2 PI))  
(define oport (open-output-file "xx.csv"))  
(display ", " oport)  
(do ((i 0 (+ i 1))) ((> i n))  
    (display (/ (* i a) n) oport)  
    (display ", " oport)  
)  
(display "\n" oport)  
(do ((i 0 (+ i 1))) ((> i n))  
    (display (/ (* i a) n) oport)  
    (display ", " oport)  
(do ((j 0 (+ j 1))) ((> j n))  
    (define yp (* (sin (* j (/ a n))) (cos (* i (/ a n))))))  
    (display yp oport)  
    (if (not (equal? j n))  
        (display ", " oport))
```

Appendix A: Commands

bbox

```
)  
  (display "\n" oport)  
)  
(close-output-port oport)  
  
(sde:clear)  
(define layermaterial (list "TCO" "Copper" "PolySi"))  
(define layerregion (list "tco" "copper" "polysi"))  
(define layerthickness (list 2 2 2))  
(sdegeo:create-cuboid (position 0 0 0) (position (* 2 PI) (* 2 PI) 4)  
  "Silicon" "xx")  
(afm-smooth-layers "xx.csv" 0 4 layerregion layermaterial  
  layerthickness)
```

bbox

This Scheme extension returns the minimum and maximum values corresponding to a diagonal across the bounding box of a body, relative to the active coordinate system.

This Scheme extension returns a pair comprising (`min-pt . max-pt`). The returned bounding box is tight (exact).

Syntax

```
(bbox entity-list)
```

Returns

POSITION LIST

Arguments

Argument	Description
entity-list	This argument comprises an entity or a list of entities. Each entity or list of entities can be a body, a wire body, a face, or an edge. Argument type: ENTITY ENTITY LIST

Appendix A: Commands

bbox-exact

bbox-exact

This Scheme extension returns the exact minimum and maximum values corresponding to a diagonal across the *tight bounding box* of a body, relative to the active coordinate system.

This Scheme extension returns a pair comprising (min-pt . max-pt). The returned bounding box is tight (exact).

Syntax

```
(bbox-exact entity-list)
```

Returns

POSITION LIST

Arguments

Argument	Description
entity-list	This argument comprises an entity or a list of entities. Each entity or list of entities can be a body, a wire, a face, or an edge. Argument type: ENTITY ENTITY LIST

Appendix A: Commands

body?

body?

This Scheme function determines whether a Scheme object is a body. It returns #t if the object is a body; otherwise, it returns #f.

Syntax

(body? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

Examples

```
(define myrect (sdegeo:create-rectangle (position 0 0 0)
                                         (position 1 1 0) "Silicon" "Region_1"))
(body? myrect)
;; #t
```

Appendix A: Commands

build-csv-lens

build-csv-lens

This Scheme extension creates a solid body from user-defined data.

Syntax

```
(build-csv-lens fname bottom-z material-name [region-name])
```

Returns

ENTITY ID (body)

Arguments

Argument	Description
fname	The name of the input data file in CSV format. Argument type: STRING
bottom-z	Argument type: REAL
material-name	Argument type: DATEXMAT
region-name	Optional. Argument type: STRING

Description

The input data file is a CSV file, which defines the user data over a tensor grid, in the following format ($n \times m$ grid):

```
,x1, x2, ..., xn  
y1, f11, f12, ..., f1n  
y2, f21, f22, ..., f2n  
...  
ym, fm1, fm2, ..., fmn
```

The x_1, x_2, \dots, x_n values define the x-coordinates of the tensor grid.

The y_1, y_2, \dots, y_m values define the y-coordinates of the tensor grid.

The f_{ij} values define the function values at the given (x_j, y_i) tensor-grid points. The base of the created solid body is placed in the xy plane at $z = \text{bottom-}z$, and the tensor-grid points define the top surface in the +z-direction.

Appendix A: Commands

color:rgb

color:rgb

This Scheme extension specifies the red, green, and blue (RGB) color components with normalized real numbers, ranging from 0 to 1.

Syntax

(color:rgb red green blue)

Returns

COLOR

Arguments

Argument	Description
red	Specifies the red color component. Argument type: REAL
green	Specifies the green color component. Argument type: REAL
blue	Specifies the blue color component. Argument type: REAL

Description

Some colors are predefined as follows:

```
(define rgb:red (color:rgb 1 0 0))
(define rgb:green (color:rgb 0 1 0))
(define rgb:blue (color:rgb 0 0 1))
(define rgb:yellow (color:rgb 1 1 0))
(define rgb:magenta (color:rgb 1 0 1))
(define rgb:cyan (color:rgb 0 1 1))
(define rgb:black (color:rgb 0 0 0))
(define rgb:white (color:rgb 1 1 1))
```

Therefore, `rgb:red`, `rgb:green`, and so on can also be used as `color:rgb` values.

Appendix A: Commands

complete-edge-list

complete-edge-list

This Scheme extension appends the specified edge list, with all neighboring linear edges.

If some 2D boundary simplification algorithms are used, which require an edge list as input, then extend the edge list with all neighboring linear edges. This ensures that the boundary simplification algorithm does not leave gaps or does not create overlaps between neighboring bodies.

Syntax

```
(complete-edge-list edge-list)
```

Returns

LIST

Arguments

Argument	Description
edge-list	Specifies an edge list. Argument type: EDGE LIST

Examples

```
(sde:clear)
(define mb1 (sdegeo:create-rectangle (position 0 0 0)
                                       (position 10 10 0) "Silicon" "x1"))
(define mb2 (sdegeo:create-rectangle (position 5 5 0)
                                       (position 15 15 0) "PolySilicon" "x2"))
(define elist (entity:edges mb1))
(length elist)
;; 6
(define elistc (complete-edge-list elist))
(length elistc)
;; 8
```

Appendix A: Commands

convert-to-degree

convert-to-degree

This Scheme extension converts an angle from radian to degree.

Syntax

(convert-to-degree angle)

Returns

REAL

Arguments

Argument	Description
angle	Argument type: REAL

convert-to-radian

This Scheme extension converts an angle from degree to radian.

Syntax

(convert-to-radian angle)

Returns

REAL

Arguments

Argument	Description
angle	Argument type: REAL

Appendix A: Commands

edge?

edge?

This Scheme function determines whether a Scheme object is an edge. It returns #t if a Scheme object is an edge.

Syntax

(edge? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

Appendix A: Commands

edge:circular

edge:circular

This Scheme function creates an arc with the specified center position and radius.

The start and end of the angle are measured counterclockwise from the x-axis of the active coordinate system. The start and end locations must be in the current xy construction plane.

Syntax

```
(edge:circular center-position radius [start-angle=0 [end-angle=360]])
```

Returns

ENTITY (EDGE)

Arguments

Argument	Description
center-position	Specifies the center position of the arc. Argument type: POSITION
radius	Specifies an implicit line between the center position and the edge of the arc. Argument type: REAL
start-angle	Optional. Specifies the starting point of the arc in degrees. Argument type: REAL
end-angle	Optional. Specifies the end angle of the arc in degrees. Argument type: REAL

Appendix A: Commands

edge:circular?

edge:circular?

This Scheme function determines whether a Scheme object is a circular edge.

It returns #t if a Scheme object is a circular edge.

Syntax

(edge:circular? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

edge:elliptical?

This Scheme function determines if a Scheme object is an elliptical edge.

It returns #t if a Scheme object is an elliptical edge.

Syntax

(edge:elliptical? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

Appendix A: Commands

edge:end

edge:end

This Scheme function returns the end position of an edge.

Syntax

(edge:end edge)

Returns

POSITION

Arguments

Argument	Description
edge	Specifies an input edge. Argument type: EDGE

edge:length

This Scheme function returns the length of an edge.

Syntax

(edge:length edge)

Returns

REAL

Arguments

Argument	Description
edge	Specifies an input edge. Argument type: EDGE

Appendix A: Commands

edge:linear

edge:linear

This Scheme function specifies locations as positions and creates a linear edge between two locations.

Syntax

```
(edge:linear position position)
```

Returns

```
ENTITY (EDGE)
```

Arguments

Argument	Description
position	The first position argument specifies the start location of the line. The second position argument specifies the end location of the line. Argument type: POSITION

Examples

```
; Scheme function
; edge:linear
; Create two linear edges given two positions.
(define edge1 (edge:linear (position 0 0 0) (position 30 30 0)))
```

Appendix A: Commands

edge:linear?

edge:linear?

This Scheme function determines whether a Scheme object is a linear edge.

It returns #t if a Scheme object is a linear edge.

Syntax

(edge:linear? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

Examples

```
(define myrect (sdegeo:create-rectangle (position 0 0 0)
                                         (position 1 1 0) "Silicon" "Region_1"))
(define myedges (entity:edges myrect))
(edge:linear? (list-ref myedges 0))
;; #t
```

Appendix A: Commands

edge:mid-point

edge:mid-point

This Scheme function returns the midpoint position of an edge.

Syntax

```
(edge:mid-point edge [approximation=#t])
```

Returns

POSITION

Arguments

Argument	Description
edge	Specifies an input entity. Argument type: EDGE
approximation	Optional. If this argument is set to true (#t), then the Scheme function returns the exact geometric midpoint of the edge. If it is set to false (#f), then the Scheme function returns the midpoint in parameter space. The default is #t. Argument type: BOOLEAN

edge:start

This Scheme function returns the starting position of an edge.

Syntax

```
(edge:start edge)
```

Returns

POSITION

Arguments

Argument	Description
edge	Specifies an input entity. Argument type: EDGE

Appendix A: Commands

edge:type

edge:type

This Scheme function determines the type of an edge.

Syntax

(edge:type edge)

Returns

STRING

Arguments

Argument	Description
edge	Specifies an input edge. Argument type: EDGE

entity:box

This Scheme function returns the extrema box (minimum and maximum values) corresponding to a diagonal across the bounding box of a body, relative to the active coordinate system.

It returns a pair comprising (min-pt . max-pt). It returns a solid box if create-box is specified.

Syntax

(entity:box entity-list [create-box])

Returns

POSITION LIST

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Each entity or list of entities can be a body, a wire, a face, or an edge. Argument type: ENTITY ENTITY LIST
create-box	Optional. If this argument is specified, then a solid box is returned. Argument type: BOOLEAN

Appendix A: Commands

entity:copy

entity:copy

This Scheme function copies any entity (such as a solid, a face, and an edge) and all entities attached to it.

Syntax

(entity:copy entity-list)

Returns

ENTITY | ENTITY LIST

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities to be copied. Argument type: ENTITY ENTITY LIST

Appendix A: Commands

entity:debug

entity:debug

This Scheme function prints information about the data structure of an entity.

Syntax

```
(entity:debug entity [level])
```

Returns

STRING

Arguments

Argument	Description
entity	Specifies an entity to be queried. Argument type: ENTITY
level	Optional. Controls the amount of debug information generated, where: <ul style="list-style-type: none">• 0 writes no output and only returns the entity type.• 1 writes only data specific to the given entity.• 2 writes additional information for some entity types. For example, if the entity is an edge, then the coordinates of the endpoints and the curve data also are written. This Scheme function writes only data specific to the given entity.• 3 writes the size of the entity and all associated entities.• 4 writes the complete information for the entity and all associated entities. Argument type: INTEGER

Appendix A: Commands

entity:deep-copy

entity:deep-copy

This Scheme function deep copies an entity (solid, face, edge, and so on) as well as all attached entities.

The difference between `entity:deep-copy` and `entity:copy` is that `entity:deep-copy` makes a copy of `splf_splf`; whereas, `entity:copy` creates a pointer to `splf_splf`.

The deep-copy functionality is used instead of the regular copy when it is required that no links exist with shared information between the original and deep copies.

Syntax

```
(entity:deep-copy entity-list)
```

Returns

ENTITY | ENTITY LIST

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities to be deep copied. Argument type: ENTITY ENTITY LIST

entity:delete

This Scheme function deletes an entity or a list of entities, and any attributes attached to those entities.

Syntax

```
(entity:delete entity-list)
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities to be deleted. Argument type: ENTITY ENTITY LIST

Appendix A: Commands

entity:dist

entity:dist

This Scheme function finds the minimum distance between two entities, or an entity and a position.

Using two input entities, this Scheme function finds a position on each entity such that the distance between the two is the minimum distance. Supported entities include BODY, EDGE, FACE, LOOP, LUMP, SHELL, VERTEX, and WIRE.

Syntax

```
(entity:dist part1 part2 [acis-opts])
```

Returns

```
(REAL POSITION (ENTITY | ENTITY LIST . STRING))
```

Arguments

Argument	Description
part1	Specifies either an entity or a position. Caution: If part1 is defined as a position, then part2 must be an entity. Argument type: ENTITY POSITION
part2	Specifies either an entity or a position. Argument type: ENTITY POSITION
acis-opts	Optional. Use this argument to switch on journaling and versioning options. Argument type: ACIS OPTIONS

Appendix A: Commands

entity:edges

entity:edges

This Scheme function returns a list of all edge entities of an entity or a list of entities.

It returns an empty list when no edges are found.

Syntax

```
(entity:edges entity-list)
```

Returns

EDGE | EDGE LIST

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Argument type: ENTITY ENTITY LIST

entity:erase

This Scheme function erases, but does not remove, the specified entity or list of entities.

The entities remain available for later redisplay. To redisplay an erased entity or a list of entities, use the Scheme function `render:rebuild`.

Syntax

```
(entity:erase entity-list)
```

Returns

Input entity-list

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities to be erased from the display. Argument type: ENTITY ENTITY LIST

Appendix A: Commands

entity:faces

entity:faces

This Scheme function returns a list of all face entities of an entity or a list of entities. It returns an empty list when no faces are found. The input argument is an `entity-list` from which a list of all faces is to be obtained.

Syntax

```
(entity:faces entity-list)
```

Returns

FACE | FACE LIST

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Argument type: ENTITY ENTITY LIST

Appendix A: Commands

entity:loops

entity:loops

This Scheme function returns a list of all the loops of an entity or a list of entities. If no loops are found, then it returns an empty list.

Syntax

```
(entity:loops entity-list [include-pat])
```

Returns

LOOP | LOOP LIST

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities to be queried. Argument type: ENTITY ENTITY LIST
include-pat	Optional. Determines how this Scheme function deals with patterned objects. By default, patterned objects are included in the list of entities. You can specify any one of the following: <ul style="list-style-type: none">• 0 – Patterned objects are created if they do not already exist and are included in the list.• 1 – Only those patterned objects that already exist are included in the list.• 2 – No patterned objects, except seed pattern objects, are included in the list. Argument type: INTEGER

Appendix A: Commands

entity:lumps

entity:lumps

This Scheme function returns a list of all the lumps of an entity or a list of entities. If no lumps are found, then it returns an empty list.

Syntax

```
(entity:lumps entity-list)
```

Returns

LUMP | LUMP LIST

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities to be queried. Argument type ENTITY ENTITY LIST

Appendix A: Commands

entity:set-color

entity:set-color

This Scheme function sets the display color for an entity or a list of entities.

Syntax

```
(entity:set-color entity-list color)
```

Returns

Previous color of entity

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities to be assigned a color. Argument type: ENTITY ENTITY LIST
color	This argument can accept an integer or a color:rgb value. It specifies a new color to be assigned to entities. The predefined color values include: <ul style="list-style-type: none">• BLACK = 0 = #[color 0 0 0]• RED = 1 = #[color 1 0 0]• GREEN = 2 = #[color 0 1 0]• BLUE = 3 = #[color 0 0 1]• CYAN = 4 = #[color 0 1 1]• YELLOW = 5 = #[color 1 1 0]• MAGENTA = 6 = #[color 1 0 1]• WHITE = 7 = #[color 1 1 1] These color variable names are already defined in the Sentaurus Structure Editor Scheme interpreter and can be used instead of the numeric values. Argument type: COLOR

Examples

Example 1

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silver"
    "region_1")
(sdegeo:create-sphere (position 1 1 1) 0.5 "SiliconCarbide"
    "region_2")
(entity:set-color (find-face-id (position 0.1 0.1 1)) RED)
```

Appendix A: Commands

entity:shells

Example 2

```
(sde:clear)
(sdgeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon" "xx")
(entity:set-color (find-face-id (position 0.5 0.5 1)) 3)
(entity:set-color (find-face-id (position 1 0.5 0.5)) rgb:yellow)
(entity:set-color (find-face-id (position 0.5 1 0.5)) CYAN)
```

entity:shells

This Scheme function returns a list of shell entities from a topological entity or a list of entities. If no shell entities are found, then this Scheme function returns an empty list.

Syntax

```
(entity:shells entity-list [include-pat])
```

Returns

SHELL | SHELL LIST

Arguments

Argument	Description
entity-list	Specifies a topological entity or an entity list. Argument type: ENTITY ENTITY LIST
include-pat	Optional. Determines how this Scheme function deals with patterned objects. By default, patterned objects are included in the list of entities. You can specify any one of the following: <ul style="list-style-type: none">• 0 – Patterned objects are created if they do not already exist and are included in the list.• 1 – Only those patterned objects that already exist are included in the list.• 2 – No patterned objects, except seed pattern objects, are included in the list. Argument type: INTEGER

Appendix A: Commands

entity:vertices

entity:vertices

This Scheme function returns a list of all the vertices in an entity or a list of entities. It returns an empty list if no vertices are found.

Syntax

```
(entity:vertices entity-list)
```

Returns

VERTEX | VERTEX LIST

Arguments

Argument	Description
entity-list	Specifies an entity or an entity list. Argument type: ENTITY ENTITY LIST

Appendix A: Commands

env:set-tolerance

env:set-tolerance

This Scheme function sets modeling tolerances. The system tolerances are set in the following order: resabs, resnor, resfit, and resmch. The ratio of resabs to resnor is the largest number that can be represented, that is, the modeling range must be within this ratio.

Note:

For each argument, REAL sets the value, BOOLEAN #f leaves the value unchanged, and BOOLEAN #t sets the value to the default.

Syntax

```
(env:set-tolerance resabs resnor resfit resmch)
```

Returns

The set tolerance values: (REAL REAL REAL REAL)

Arguments

Argument	Description
resabs	Determines whether two positions in space are equal. This tolerance defines the smallest distance between two distinct points. Argument type: REAL (default=1e-6)
resnor	Checks a number for equality with 0. This tolerance checks the components of numbers that are 0 to determine whether vectors are parallel or perpendicular, or to check for zero lengths. Argument type: REAL (default=1e-10)
resfit	Fits interpolation curves when intersecting surfaces. Argument type: REAL (default=1e-3)
resmch	Used by the spline functions for purposes similar to resnor. Argument type: REAL (default=1e-11)

Appendix A: Commands

env:tolerance

env:tolerance

This Scheme function returns the system tolerances as a list, in the following order: resabs, resnor, resfit, resmch. The Scheme function `env:set-tolerance` explains each tolerance value (see [env:set-tolerance on page 369](#)).

Syntax

`(env:tolerance)`

Returns

The set tolerance values: `(REAL REAL REAL REAL)`

erf

This Scheme function returns the error function (erf) value of the input.

Syntax

`(erf value)`

Returns

`REAL`

Arguments

Argument	Description
value	Argument type: <code>REAL</code>

Appendix A: Commands

erfc

erfc

This Scheme function returns the complementary error function (erfc) value of the input.

Syntax

(erfc value)

Returns

REAL

Arguments

Argument	Description
value	Argument type: REAL

exists-empty-mask-name

This Scheme function checks whether the specified empty mask exists.

Syntax

(exists-empty-mask-name maskname)

Returns

BOOLEAN

Arguments

Argument	Description
maskname	Specifies the name of an empty mask. Argument type: STRING

Appendix A: Commands

exists-mask-name

exists-mask-name

This Scheme function checks whether the specified mask exists.

Syntax

```
(exists-mask-name maskname)
```

Returns

BOOLEAN

Arguments

Argument	Description
maskname	Specifies the name of a mask. Argument type: STRING

Appendix A: Commands

extract-interface-normal-offset-refwindow

extract-interface-normal-offset-refwindow

This command generates the non-regularized intersection between two bodies and creates a 3D offset body from the non-regularized intersection, by offsetting the faces of the interface body in the normal direction, and assigns it as a doping/refinement/submesh (DRS) body.

Syntax

```
(extract-interface-normal-offset-refwindow body1 body2  
    offset-distance refwindowname)
```

Returns

ENTITY ID of the created Ref/Eval window

Arguments

Argument	Description
body1	Argument type: ENTITY
body2	Argument type: ENTITY
offset-distance	Argument type: REAL
refwindowname	Argument type: STRING

Examples

```
(sde:clear)  
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 4)  
    "Silicon" "x1"))  
(define mb2 (sdegeo:create-cylinder (position 5 5 4) (position 5 5 8)  
    2 "PolySilicon" "x2"))  
(extract-interface-normal-offset-refwindow mb1 mb2 1 "rw1")
```

Appendix A: Commands

extract-interface-offset-refwindow

extract-interface-offset-refwindow

This command generates the non-regularized intersection between two bodies and creates a 3D offset body from the non-regularized intersection and assigns it as a DRS body.

Syntax

```
(extract-interface-offset-refwindow body1 body2 offset-distance  
refwindowname)
```

Returns

ENTITY ID of the created Ref/Eval window

Arguments

Argument	Description
body1	Argument type: ENTITY
body2	Argument type: ENTITY
offset-distance	Argument type: REAL
refwindowname	Argument type: STRING

Examples

```
(sde:clear)  
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 4)  
"Silicon" "x1"))  
(define mb2 (sdegeo:create-cylinder (position 5 5 4) (position 5 5 8)  
2 "PolySilicon" "x2"))  
(extract-interface-offset-refwindow mb1 mb2 1 "rw1")
```

Appendix A: Commands

extract-refpolyhedron

extract-refpolyhedron

This Scheme extension converts a 3D geometric body to a polyhedral refinement window. The ACIS surface mesher of Sentaurus Structure Editor, which saves the tessellated polyhedral boundary for meshing, converts the 3D geometric body (possibly with nonplanar faces) to a polyhedral refinement window.

Syntax

```
(extract-refpolyhedron bodyid refwindowname)
```

Returns

ENTITY

Arguments

Argument	Description
bodyid	Argument type: ENTITY
refwindowname	Argument type: STRING

Examples

Example 1

```
(sde:clear)
(define mb (sdegeo:create-cuboid (position 0 0 0) (position 1 1 1)
    "Silicon" "x1"))
(extract-refpolyhedron mb "ref1")
```

Example 2

```
(sde:clear)
(define mb (sdegeo:create-cylinder (position 0 0 0) (position 0 0 1)
    0.1 "Silicon" "x1"))
(extract-refpolyhedron mb "ref1")
(sdedr:define-refinement-size "RD_1" .2 .2 .1 .1)
(sdedr:define-refinement-placement "RP_1" "RD_1" "ref1")
(sdedr:write-cmd-file "xx.cmd")
```

Appendix A: Commands

extract-refwindow

extract-refwindow

This Scheme extension defines refinement windows, matching the specified geometry face or faces.

Syntax

```
(extract-refwindow faceid refwindowname)
```

Returns

None

Arguments

Argument	Description
faceid	Argument type: FACE FACE LIST EDGE
refwindowname	Argument type: STRING

Description

This Scheme extension goes through the specified face list (or the specified single face list), extracts the vertex list for each face, and defines a separate polygonal refinement window for each face. Planar faces will have the corresponding refinement polygon defined as one single refinement polygon. However, for curved faces, the geometric face will be triangulated first and a separate refinement polygon (triangle) will be defined for each triangle. The coplanar triangular faces of the extracted refinement body are merged by default. Merging the coplanar triangular faces can be disabled by setting the `merge-extracted-drs-faces` global variable to `#f`. (It is set to `#t` by default.)

The `extract-refwindow` Scheme extension also supports converting linear geometric edges to Ref/Eval windows (line elements). The argument list is unchanged; however, in this case, the first argument must be a linear edge.

Examples

Example 1

```
(sde:clear)
(define mypyramid (sdegeo:create-pyramid
  (position 0 0 0) 20 40 40 6 12 "PolySilicon" "r1"))
(define myface (list-ref (entity:faces mypyramid) 2))
(extract-refwindow myface "refinement1")
```

Appendix A: Commands

face:area

Example 2

```
(sde:clear)
(define mycube (sdegeo:create-cuboid (position 0 0 0)
                                         (position 10 10 10) "Silicon" "r1"))
(define myedge (car (find-edge-id (position 10 5 10))))
(sdegeo:fillet myedge 4)
(define mflist (entity:faces mycube))
(extract-refwindow (list-ref mflist 2) "refwin1")
(extract-refwindow (list-ref mflist 5) "refwin2")
(extract-refwindow (list-ref mflist 0) "refwin3")
```

Example 3

```
(sde:clear)
(define mb (sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
                                         "Silicon" "xx"))
(extract-refwindow (car (entity:edges mb)) "rw1")
```

face:area

This Scheme function calculates the area of a specified face. The accuracy of the calculation is fixed at 0.001 for an area of geometry that cannot be determined analytically.

Syntax

```
(face:area face)
```

Returns

REAL

Arguments

Argument	Description
face	Specifies an input face. Argument type: FACE

Appendix A: Commands

face:conical?

face:conical?

This Scheme function determines whether a Scheme object is a conical face.

It returns #t if the object is a conical face; otherwise, it returns #f.

Syntax

(face:conical? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

face:cylindrical?

This Scheme function determines whether a Scheme object is a cylindrical face.

The returned Boolean specifies whether the supplied entity input is a cylindrical face.

Syntax

(face:cylindrical? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Note: This input argument is a cylindrical face and <i>not</i> a solid cylinder. Argument type: SCHEME OBJECT

Appendix A: Commands

face:planar?

face:planar?

This Scheme function determines whether a Scheme object is a planar face.

It returns #t if the specified object is a planar face.

Syntax

(face:planar? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

face:plane-normal

This Scheme function returns the face normal of the specified planar face.

Syntax

(face:plane-normal face)

Returns

GVECTOR

Arguments

Argument	Description
face	Specifies a planar face to be queried. Argument type: ENTITY FACE

Appendix A: Commands

face:spherical?

face:spherical?

This Scheme function determines whether a Scheme object is a spherical face.

It returns #t if the specified object is a spherical face.

Syntax

(face:spherical? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

face:spline?

This Scheme function determines whether a Scheme object is a face-spline.

Syntax

(face:spline? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

Appendix A: Commands

face:toroidal?

face:toroidal?

This Scheme function determines whether a Scheme object is a toroidal face.

Syntax

(face:toroidal? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

filter:type

This Scheme function creates the specified *type-name* as a filter, which specifies the type of entity to be used in another filter operation. If a new type filter is created, then it replaces the previously defined type.

Use this Scheme function to display a list of available filter types.

Syntax

(filter:type *type-name*)

Returns

FILTER

Appendix A: Commands

find-body-id

Arguments

Argument	Description
type-name	Specifies the entity filter to be created. Possible values are: "body?" "edge?" "edge:circular?" "edge:curve?" "edge:elliptical?" "edge:linear?" "edge:spline?" "face?" "face:conical?" "face:cylindrical?" "face:planar?" "face:spherical?" "face:spline?" "face:toroidal?" "mixed-body?" "point?" "solid?" "text?" "vertex?" "wcs?" "wire?" "wire-body?" Argument type: STRING

find-body-id

This Scheme extension finds the entity numbers of a body. It goes through all bodies, (`get-body-list`), and returns the ACIS entity IDs of all bodies that contain the specified position. The position can be on a boundary face or vertex or can be an internal point.

Syntax

`(find-body-id position)`

Returns

BODY LIST

Arguments

Argument	Description
position	Argument type: POSITION

Appendix A: Commands

find-body-id-drs

find-body-id-drs

This Scheme extension finds the entity numbers of a body for a refinement/evaluation (Ref/Eval) window body. It goes through the body list of all defined Ref/Eval window bodies and returns the ACIS entity IDs of all bodies that contain the specified position. The position can be on a boundary face or can be an internal point.

Syntax

(find-body-id-drs position)

Returns

BODY LIST

Arguments

Argument	Description
position	Argument type: POSITION

find-drs-id

This Scheme extension returns the entity numbers of a Ref/Eval window body. It goes through all defined Ref/Eval window bodies and returns the ACIS entity IDs of the Ref/Eval window body that matches the specified DRS name. (The DRS name is a unique identifier of each Ref/Eval window body.)

Syntax

(find-drs-id DRSname)

Returns

BODY

Arguments

Argument	Description
DRSname	Argument type: STRING

Appendix A: Commands

find-edge-id

find-edge-id

This Scheme extension finds the entity numbers of an edge. It goes through the edge list of all defined bodies, (`(entity:edges (get-body-list))`), and returns the ACIS entity IDs of all edges that contain the specified position. The position can be either an end vertex position of the edge or an internal position.

This Scheme extension provides an additional filter for selecting bodies.

Syntax

`(find-edge-id position body)`

Returns

SYNTAX LIST

Arguments

Argument	Description
position	Argument type: POSITION
body	Argument type: BODY BODY LIST

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon" "x1")
(sdegeo:create-cuboid (position 1 0 0) (position 2 1 1) "PolySilicon"
  "x2")
(define chamferedges (find-edge-id (position 1 0.5 1)
  (find-material-id "Silicon")))
(sdegeo:chamfer chamferedges 0.2)
```

Appendix A: Commands

find-edge-id-drs

find-edge-id-drs

This Scheme extension finds the entity numbers of an edge for a Ref/Eval window body. It goes through the edge list of all defined Ref/Eval window bodies and returns the ACIS entity IDs of all edges that contain the specified position. The position can be either an end vertex position of the edge or an internal position.

Syntax

(find-edge-id-drs position)

Returns

EDGE LIST

Arguments

Argument	Description
position	Argument type: POSITION

find-face-id

This Scheme extension finds the entity numbers of a face. It goes through the face list of all defined bodies, (entity:faces (get-body-list)), and returns the ACIS entity IDs of all faces that contain the specified position. If the input argument is a gvector, then all the planar faces with the specified face normal are returned.

Syntax

(find-face-id position | gvector)

Returns

FACE LIST

Arguments

Argument	Description
position	Specifies either a position lying on a boundary edge or an internal position. Argument type: POSITION
gvector	Argument type: GVECTOR

Appendix A: Commands

find-face-id-drs

find-face-id-drs

This Scheme extension finds the entity numbers of a face for a Ref/Eval window body. It goes through the face list of all defined Ref/Eval window bodies and returns the ACIS entity IDs of all faces that contain the specified position.

Syntax

(find-face-id-drs position)

Returns

FACE LIST

Arguments

Argument	Description
position	Specifies either a position lying on a boundary edge or an internal position. Argument type: POSITION

find-material-id

This Scheme extension finds and returns all entities that have the given material attribute. It goes through every region in the model and selects all regions that have the specified `material-name` as the material attribute.

Syntax

(find-material-id material-name)

Returns

BODY | BODY LIST

If only one body is found with the specified `material-name` as the material attribute, then the entity ID of the material is returned. If more than one region has the `material-name` as the material attribute, then a list of all entities is returned. If no entity is found, then an empty list is returned.

Arguments

Argument	Description
material-name	Specifies the name of a material. Argument type: DATEXMAT

Appendix A: Commands

find-region

find-region

This Scheme extension returns the entity IDs for a specific region name. It examines every region in the model and selects all regions that have the specified `region-name` as the region attribute.

Syntax

```
(find-region region-name [partial-match])
```

Returns

BODY | BODY LIST

If only one body is found with the specified `region-name` as the region attribute, then the entity ID of the region is returned. If more than one region has the `region-name` as the region attribute, then a list of all entities is returned. If no entity is found, then an empty list is returned.

Arguments

Argument	Description
region-name	Specifies the name of a region. Argument type: STRING
partial-match	Optional. If this argument is used, then the Scheme extension returns the partially matched region names as well, as long as the partial match criterion is incremental (that is, it is satisfied from the beginning of the string). Argument type: BOOLEAN

Appendix A: Commands

find-region-id

find-region-id

This Scheme extension returns the entity IDs for a specific region name. It goes through every region in the model and selects all regions that have the specified `region-name` as the region attribute.

Syntax

```
(find-region-id region-name)
```

Returns

BODY | BODY LIST

If only one body is found with the specified `region-name` as the region attribute, then the entity ID of the region is returned. If more than one region has the `region-name` as the region attribute, then a list of all entities is returned. If no entity is found, then an empty list is returned.

Arguments

Argument	Description
region-name	Argument type: STRING

find-vertex-id

This Scheme extension finds the entity numbers of a vertex. It goes through the vertex list of all defined bodies, (`entity:vertices (get-body-list)`), and returns the ACIS entity IDs of all vertices that are placed at the specified position.

Syntax

```
(find-vertex-id position)
```

Returns

VERTEX LIST

Arguments

Argument	Description
position	Argument type: POSITION

Appendix A: Commands

find-vertex-id-drs

find-vertex-id-drs

This Scheme extension finds the entity numbers of a vertex for a Ref/Eval window body. It goes through the vertex list of all defined Ref/Eval window bodies and returns the ACIS entity IDs of all vertices that contain the specified position.

Syntax

(find-vertex-id-drs position)

Returns

VERTEX LIST

Arguments

Argument	Description
position	Argument type: POSITION

get-body-list

This Scheme extension returns a list of all 2D and 3D bodies that are defined in the model, except the mask bodies and the doping-related or refinement-related Ref/Eval windows. Since mask bodies and Ref/Eval windows are also sheet bodies, (part:entities (filter:type "solid?")) also returns these entities.

Since most of the geometric operations are performed only on 2D and 3D bodies, and not on masks and Ref/Eval windows, this Scheme extension is used to distinguish easily between masks, Ref/Eval windows, and regular bodies.

Syntax

(get-body-list)

Returns

BODY LIST

Appendix A: Commands

get-drs-list

get-drs-list

This Scheme extension returns a list of all 2D and 3D Ref/Eval windows that are defined in the model.

Syntax

(get-drs-list)

Returns

BODY LIST

get-empty-mask-list

This Scheme extension returns a list of all defined empty masks.

If no empty mask is defined, then an empty list is returned.

Syntax

(get-empty-mask-list)

Returns

LIST

get-mask-list

This Scheme extension returns a list of all defined (geometric) masks.

If no mask is defined, then an empty list is returned.

Syntax

(get-mask-list)

Returns

LIST

Appendix A: Commands

gvector

gvector

This Scheme function creates a new gvector with the specified x-, y-, and z-coordinates, relative to the active coordinate system.

Syntax

```
(gvector x y z [space=model])
```

Returns

GVECTOR

Arguments

Argument	Description
x	Defines the x-coordinate. Argument type: REAL
y	Defines the y-coordinate. Argument type: REAL
z	Defines the z-coordinate. Argument type: REAL
space	<p>Optional. This argument defaults to <code>wcs</code> for the active (working) coordinate system. If no active work plane exists, then <code>space</code> defaults to <code>model</code>. The other optional <code>space</code> arguments return a gvector in the new coordinate system:</p> <ul style="list-style-type: none">The value <code>model</code> means that the x-, y-, and z-values are represented with respect to the model. If the model has an origin other than the active work plane, this returns the position relative to the active coordinate system in rectangular Cartesian coordinates.The value <code>polar</code> or <code>cylindrical</code> means that the x-, y-, and z-values are interpreted as the radial distance from the z-axis, the polar angle in degrees measured from the xz plane (using the right-hand rule), and the z-coordinate, respectively. This returns the x-, y-, and z-terms with respect to the active coordinate system.The value <code>spherical</code> means that the provided x-, y-, and z-values are the radial distance from the origin, the angle of declination from the z-axis in degrees, and the polar angle measured from the xz plane in degrees, respectively. This returns the x-, y-, and z-terms relative to the active coordinate system. <p>Argument type: STRING</p>

Appendix A: Commands

gvector?

gvector?

This Scheme function determines whether a Scheme object is a gvector.

Syntax

(gvector? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

gvector:+

This Scheme function returns the result of adding gvector1 + gvector2 as a gvector.

Syntax

(gvector:+ gvector1 gvector2)

Returns

GVECTOR

Arguments

Argument	Description
gvector1	Defines the first gvector. Argument type: GVECTOR
gvector2	Defines the second gvector. Argument type: GVECTOR

Examples

(gvector:+ (gvector 1 3 2) (gvector 2 2 2))

Appendix A: Commands

gvector:-

gvector:-

This Scheme function returns the result of subtracting gvector1 – gvector2 as a gvector.

Syntax

```
(gvector:- gvector1 gvector2)
```

Returns

GVECTOR

Arguments

Argument	Description
gvector1	Defines the start location for both vectors. Argument type: GVECTOR
gvector2	Defines the end location for both vectors. Argument type: GVECTOR

Examples

```
(gvector:- (gvector 1 3 2) (gvector 2 2 2))
```

gvector:copy

This Scheme function creates a gvector by copying an existing gvector.

Syntax

```
(gvector:copy gvector)
```

Returns

GVECTOR

Arguments

Argument	Description
gvector	Argument type: GVECTOR

Examples

```
(define gvcopy (gvector:copy (gvector 6 5 2)))
```

Appendix A: Commands

gvector:cross

gvector:cross

This Scheme function returns the cross product of two gvectors.

Syntax

```
(gvector:cross gvector1 gvector2)
```

Returns

GVECTOR

Arguments

Argument	Description
gvector1	Argument type: GVECTOR
gvector2	Argument type: GVECTOR

gvector:dot

This Scheme function returns the dot product of two gvectors.

Syntax

```
(gvector:dot gvector1 gvector2)
```

Returns

REAL

Arguments

Argument	Description
gvector1	Argument type: GVECTOR
gvector2	Argument type: GVECTOR

Appendix A: Commands

gvector:from-to

gvector:from-to

This Scheme function returns the gvector between two positions.

Syntax

```
(gvector:from-to position1 position2)
```

Returns

GVECTOR

Arguments

Argument	Description
position1	Defines first position. Argument type: POSITION
position2	Defines second position. Argument type: POSITION

gvector:length

This Scheme function returns the length of a gvector as a real value.

Syntax

```
(gvector:length gvector)
```

Returns

REAL

Arguments

Argument	Description
gvector	Argument type: GVECTOR

Appendix A: Commands

gvector:parallel?

gvector:parallel?

This Scheme function determines whether two gvectors are parallel.

It returns #t if the two gvectors are parallel; otherwise, it returns #f. A zero gvector is not parallel to anything including itself, so the Scheme function returns #t.

Syntax

```
(gvector:parallel? gvector1 gvector2)
```

Returns

BOOLEAN

Arguments

Argument	Description
gvector1	Defines the first gvector. Argument type: GVECTOR
gvector2	Defines the second gvector. Argument type: GVECTOR

gvector:perpendicular?

This Scheme function determines whether two gvectors are perpendicular.

It returns #t if the gvectors are perpendicular; otherwise, it returns #f. A zero gvector is perpendicular to all gvectors including itself, so the Scheme function returns #f.

Syntax

```
(gvector:perpendicular? gvector1 gvector2)
```

Returns

BOOLEAN

Arguments

Argument	Description
gvector1	Defines the first gvector. Argument type: GVECTOR
gvector2	Defines the second gvector. Argument type: GVECTOR

Appendix A: Commands

gvector:reverse

gvector:reverse

This Scheme function reverses the direction of a gvector.

Syntax

```
(gvector:reverse gvector)
```

Returns

GVECTOR

Arguments

Argument	Description
gvector	Argument type: GVECTOR

gvector:scale

This Scheme function multiplies a gvector by a scalar number to produce a new gvector.

Using this Scheme function, the resulting gvector is the original gvector scaled by the number.

Syntax

```
(gvector:scale gvector scale)
```

Returns

GVECTOR

Arguments

Argument	Description
gvector	Argument type: GVECTOR
scale	Argument type: REAL

Appendix A: Commands

gvector:set!

gvector:set!

This Scheme function sets the direction of a gvector given the components of x, y, and z. The coordinates are computed relative to the active coordinate system.

Syntax

```
(gvector:set! gvector {x y z})
```

Returns

GVECTOR

Arguments

Argument	Description
gvector	Specifies the original x-, y-, and z-components. Argument type: GVECTOR
x	Specifies the value to replace the original x-value specified in gvector. Argument type: REAL
y	Specifies the value to replace the original y-value specified in gvector. Argument type: REAL
z	Specifies the value to replace the original z-value specified in gvector. Argument type: REAL

Appendix A: Commands

gvector:set-x!

gvector:set-x!

This Scheme function sets the x-direction component of a gvector. The coordinates are computed relative to the active coordinate system.

This Scheme function returns the x-value as a real.

Syntax

```
(gvector:set-x! gvector x)
```

Returns

REAL

Arguments

Argument	Description
gvector	Specifies the original x-, y-, and z-values. Argument type: GVECTOR
x	Specifies the value to replace the original x-value specified in gvector. Argument type: REAL

Examples

```
(gvector:set-x! vector1 3)
```

Appendix A: Commands

gvector:set-y!

gvector:set-y!

This Scheme function sets the y-direction component of a gvector. The coordinates are computed relative to the active coordinate system.

This Scheme function returns the y-value as a real.

Syntax

```
(gvector:set-y! gvector y)
```

Returns

REAL

Arguments

Argument	Description
gvector	Specifies the original x-, y-, and z-values. Argument type: GVECTOR
y	Specifies the value to replace the original y-value specified in gvector. Argument type: REAL

Examples

```
(gvector:set-y! vector1 3)
```

Appendix A: Commands

gvector:set-z!

gvector:set-z!

This Scheme function sets the z-direction component of a gvector. The coordinates are computed relative to the active coordinate system.

This Scheme function returns the z-value as a real.

Syntax

```
(gvector:set-z! gvector z)
```

Returns

REAL

Arguments

Argument	Description
gvector	Specifies the original x-, y-, and z-values. Argument type: GVECTOR
z	Specifies the value to replace the original z-value specified in gvector. Argument type: REAL

Examples

```
(gvector:set-z! vector1 3)
```

Appendix A: Commands

gvector:transform

gvector:transform

This Scheme function applies a transform to a gvector.

Syntax

```
(gvector:transform gvector transform)
```

Returns

GVECTOR

Arguments

Argument	Description
gvector	Argument type: GVECTOR
transform	Argument type: TRANSFORM

gvector:unitize

This Scheme function creates a new gvector as a unit vector in the same direction as the specified gvector.

Syntax

```
(gvector:unitize gvector)
```

Returns

GVECTOR

Arguments

Argument	Description
gvector	Argument type: GVECTOR

Appendix A: Commands

gvector:x

gvector:x

This Scheme function returns the x-component of a gvector relative to the active coordinate system. The x-coordinate is transformed to the active work plane.

Syntax

(gvector:x gvector)

Returns

REAL

Arguments

Argument	Description
gvector	Argument type: GVECTOR

gvector:y

This Scheme function returns the y-component of a gvector relative to the active coordinate system. The y-coordinate is transformed to the active work plane.

Syntax

(gvector:y gvector)

Returns

REAL

Arguments

Argument	Description
gvector	Argument type: GVECTOR

Appendix A: Commands

gvector:z

gvector:z

This Scheme function returns the z-component of a gvector relative to the active coordinate system. The z-coordinate is transformed to the active work plane.

Syntax

(gvector:z gvector)

Returns

REAL

Arguments

Argument	Description
gvector	Argument type: GVECTOR

journal:abort

This Scheme function terminates the current load without executing the rest of the commands in the file.

Syntax

(journal:abort)

Returns

None

Appendix A: Commands

journal:append

journal:append

This Scheme function opens a journal file and appends additional journal data to the end of the file. The time and date of the append to the file are indicated in the journal.

Syntax

```
(journal:append [filename])
```

Returns

None

Arguments

Argument	Description
filename	Optional. If a file name is not specified, then a unique name is created after reading the current directory. The unique name is sequenced numerically from the last journal file created or named <code>j(last number+1).jrl</code> . Argument type: STRING

Appendix A: Commands

journal:clean

journal:clean

This Scheme function cleans the journal file. It removes all nonexecutable content (such as comments and return values) from the specified script file. Only the executable Scheme commands remain; therefore, parameterizing and debugging the journal file are easier if the saved journal file is cleaned up first.

Syntax

```
( journal:clean jname )
```

Returns

None

Arguments

Argument	Description
jname	Argument type: STRING

Appendix A: Commands

journal:load

journal:load

This Scheme function loads a journal file and runs each command contained in that file. Each line is journaled if journaling is switched on. This Scheme function works like the load primitive, except that the file is evaluated one line at a time instead of all lines simultaneously with the Scheme load primitive.

Encountered errors do not terminate the load operation and are reported in the command-line window. This Scheme function is useful for debugging Scheme files and for rerunning the commands given in another Scheme session.

An error in the loaded file does not terminate the evaluation. This Scheme function permits single stepping through a loaded file, one line at a time. The Scheme function (`(journal:step #t)`) switches it on and should be run before loading the journal file.

In addition, you can use (`(option:set "timing" #t)`) to show the execution time for each command in the command-line window. Before loading, the directory where the load file is found is added to the global variable `part-load-path`.

Syntax

`(journal:load filename)`

Returns

None

Arguments

Argument	Description
<code>filename</code>	Specifies the name of a journal file. Argument type: STRING

journal:off

This Scheme function closes the current journal file and switches off journaling. All Scheme extensions executed after journaling has been switched off are not retained in the journal file.

Syntax

`(journal:off)`

Returns

None

Appendix A: Commands

journal:on

journal:on

This Scheme function closes the current journal file and opens a new journal file. All future commands are journaled to this file.

Syntax

```
(journal:on [filename])
```

Returns

None

Arguments

Argument	Description
filename	Optional. If a file name is not specified, then the Scheme function creates a unique name after reading the current directory. The unique name is sequenced numerically from the last journal file created or named <code>j(last number+1).jrl</code> . If the file exists, then it is truncated to zero length. Argument type: STRING

Examples

```
; Scheme function journal:on
; Close current journal file and open new journal file called new_jrl.
(journal:on "new_jrl")
;; "new_jrl"
; Create solid cuboid.
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10) "Silicon"
    "region_1")
; Switch off journaling.
(journal:off)
;; ()
; Save the resulting journal.
(journal:save "new_jrl")
;; "new_jrl" ; Clear the part.
(sde:clear)
;; #t
; Load the journal file to recreate and redisplay the solid cube.
(journal:load "new_jrl")
;; "new_jrl"
```

Appendix A: Commands

journal:pause

journal:pause

This Scheme function deactivates journaling temporarily but leaves the journal file open. It does not record in the journal file any procedure evaluated while the journal file is paused.

Syntax

(journal:pause)

Returns

None

journal:resume

This Scheme function resumes journaling in the journal file after a pause.

Syntax

(journal:resume)

Returns

None

journal:save

This Scheme function saves the current journal to a file, and it is ignored if journaling is not active. An error is generated if `filename` is the same as the current journal file. See [journal:on on page 408](#).

Syntax

(journal:save filename)

Returns

None

Arguments

Argument	Description
filename	Specifies the name of the file in which the journal is saved. Argument type: STRING

Appendix A: Commands

journal:step

journal:step

This Scheme function sets a flag to control stepping through the journal file. When stepping is *switched on*, the system waits for input after printing, but before executing each line. A single return causes the line to be executed. Anything else is evaluated and the system waits for more input. This allows you to set up demonstrations and to debug Scheme scripts one line at a time.

To run a demonstration, enter `(journal:step #f)`, but include `(journal:step #t)` at points in the script where you want to interact with the graphical user interface (GUI) or enter commands in the command-line window. Otherwise, press the Return key a few times to step through it slowly. Use `(journal:step #f)` to return to the free-running mode.

Syntax

`(journal:step value)`

Returns

None

Arguments

Argument	Description
value	Argument type: BOOLEAN

Appendix A: Commands

loop?

loop?

This Scheme function determines whether a Scheme object is a loop.

It returns #t if the object is a loop; otherwise, it returns #f.

Syntax

(loop? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

loop:external?

This Scheme function determines whether a loop is internal or external.

Syntax

(loop:external? loop)

Returns

BOOLEAN

Arguments

Argument	Description
loop	Argument type: LOOP

Errors

Returns an error if the argument `loop` is not a loop.

Appendix A: Commands

lump?

lump?

This Scheme function determines whether a Scheme object is a lump.

It returns #t if the object is a lump; otherwise it returns #f.

Syntax

(lump? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies a Scheme object to be queried. Argument type: SCHEME OBJECT

Appendix A: Commands

mask-refevalwin-extract-2d

mask-refevalwin-extract-2d

This Scheme extension extracts a 2D Ref/Eval window from the specified mask. It assumes that the domain boundary was set previously. (This information is needed for "dark" mask extraction.)

If the mask consists of several disjoint lumps, then a separate Ref/Eval window is extracted from each disjoint lump. In this case, the specified `rwnname` is appended with a counter `_N` where `N` is the lump counter (from 1 onwards).

Caution:

The domain boundary must be set before this Scheme extension is called.

Syntax

```
(mask-refevalwin-extract-2d rwnname maskname polarity)
```

Returns

ENTITY IDs of the created Ref/Eval windows

Arguments

Argument	Description
<code>rwnname</code>	Specifies the name of the extracted mask. Argument type: STRING
<code>maskname</code>	Specifies the name (and not the entity ID) of the used mask. Argument type: STRING
<code>polarity</code>	Specifies the mask polarity that is used for the extraction. The options are "light" and "dark". The default is "light". Argument type: STRING

Examples

```
(sde:clear)
(sdepe:define-pe-domain 0 0 40 30)
(sdepe:generate-mask "M1" (list (list 0 0 10 10)
    (list 20 0 30 0 30 10 25 15 20 10))))
(mask-refevalwin-extract-2d "RW1" "M1" "light")
(mask-refevalwin-extract-2d "RW2" "M1" "dark")
```

Appendix A: Commands

mask-refevalwin-extract-3d

mask-refevalwin-extract-3d

This Scheme extension extracts a 3D Ref/Eval window from the specified mask. It assumes that the domain boundary was set previously. (This information is needed for "dark" mask extraction.)

The 3D Ref/Eval window is placed between `minz` and `maxz` in the 3D space. If the mask consists of several disjoint lumps, then a separate Ref/Eval window is extracted from each disjoint lump. In this case, the specified `rwname` is appended with a counter `_N` where `N` is the lump counter (from 1 onwards).

Caution:

The domain boundary must be set before this Scheme extension is called.

Syntax

```
(mask-refevalwin-extract-3d rwname maskname polarity minz maxz)
```

Returns

ENTITY IDs of the created Ref/Eval windows

Arguments

Argument	Description
<code>rwname</code>	Specifies the name of the extracted mask. Argument type: STRING
<code>maskname</code>	Specifies the name (and not the entity ID) of the used mask. Argument type: STRING
<code>polarity</code>	Specifies the mask polarity that is used for the extraction. The options are "light" and "dark". The default is "light". Argument type: STRING
<code>minz</code>	Argument type: REAL
<code>maxz</code>	Argument type: REAL

Examples

```
(sde:clear)
(sdepe:define-pe-domain 0 0 40 30)
(sdepe:generate-mask "M1" (list (list 0 0 10 10)
    (list 20 0 30 0 30 10 25 15 20 10))))
(mask-refevalwin-extract-3d "RW1" "M1" "light" -0.1 0.5)
(mask-refevalwin-extract-3d "RW2" "M1" "dark" -0.1 0.5)
```

Appendix A: Commands

member?

member?

This Scheme extension checks whether the specified element is part of the specified list.

It returns #t if the specified element is part of the specified list and, if not, #f is returned.

Syntax

```
(member? element list)
```

Returns

BOOLEAN

Arguments

Argument	Description
element	Argument type: VARIABLE (INTEGER REAL STRING ...)
list	Argument type: LIST (Scheme list)

Examples

```
(member? 2 (list 1 2 3 4))  
#t
```

```
(member? 0 (list 1 2 3 4))  
#f
```

```
(member? "apple" (list 1 2 "apple" 3 4))  
#t
```

Appendix A: Commands

merge-collinear-edges-2d

merge-collinear-edges-2d

This Scheme extension merges all collinear edges of the specified bodies.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(merge-collinear-edges-2d body-list [angular-tolerance])
```

Returns

None

Arguments

Argument	Description
body-list	Argument type: BODY LIST
angular-tolerance	Optional. If this argument is specified, then besides perfectly collinear edges, the edges that close an angle, which is larger than the specified threshold, also merge (that is, the two edges are replaced by one edge, connecting the other endpoints). Argument type: REAL

Appendix A: Commands

part:entities

part:entities

This Scheme function returns a list of all top-level entities in a part.

Syntax

```
(part:entities [filter=NULL])
```

Returns

ENTITY LIST

Arguments

Argument	DescriptionArgument Type
filter	Optional. Controls the types of entity that are returned by this Scheme function. This argument is an <code>entity-filter</code> , which is a procedural object that selects entities from an <code>entity-list</code> . Argument type: FILTER

Examples

```
(sde:clear)
(sdgeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silver"
  "region_3")
(part:entities)
(part:entities (filter:type "solid?"))
```

Appendix A: Commands

part:load

part:load

This Scheme function loads a part from a file into the active part. It is a merge function, which means that the restore does not replace entities; it adds them to the current working session. A list of entities is returned.

If the global variable `part-load-path` is set to a list of strings, then the strings are interpreted as directories to be searched for the part file: `part-load-path` is to `part-load` as `load-path` is to `load`. Set `part-load-path` in an initialization file. When a part is saved using `part:save` and a filename, the filename becomes the new name for the part.

Syntax

```
(part:load filename [textmode=#t] [part=active] [with-history=#f])
```

Returns

ENTITY LIST

Arguments

Argument	Description
filename	Specifies a file name in the current working directory or a path that includes the file name. Argument type: STRING
textmode	Optional. Describes the type of file to be loaded: <ul style="list-style-type: none">If <code>textmode</code> is <code>#t</code>, then the data in <code>filename</code> is saved in text mode as a <code>.sat</code> file.If <code>textmode</code> is <code>#f</code>, then <code>filename</code> is loaded as a binary file.If <code>textmode</code> is not specified, then the mode is determined by the extension of the file name. If the file name string ends in <code>.sab</code> or <code>.SAB</code>, then the file is saved in binary mode; otherwise, the file is saved in text mode. Argument type: BOOLEAN
part	Optional. If this argument is not specified, then the entities in the part file merge into the active part; otherwise, they merge into the specified part. Argument type: PART
with-history	Optional. After specifying the text mode, this argument specifies whether to restore the rollback history data. History data can only be restored into an empty part. The default, <code>#f</code> , does not restore the history. Argument type: BOOLEAN

Appendix A: Commands

part:save

part:save

This Scheme function saves all entities in a part to a file.

When a part is saved using `part:save` and a `filename`, the `filename` becomes the new name for the part. Part files can be saved in the format of an earlier version by setting the global option `save_version`.

Syntax

```
(part:save [filename=partname.sat] [textmode=#t] [part=active]
           [with-history=#f] [mainline-only=#f])
```

Returns

BOOLEAN

Arguments

Argument	Description
filename	Optional. Specifies a file name to be saved in the current working directory or specifies a path that includes the file name to be saved to a directory. If no file name is specified, then the Scheme function uses the name given to the part with the <code>part:set-name</code> function. Argument type: STRING
textmode	Optional. Describes the type of file to be loaded: <ul style="list-style-type: none">• If <code>textmode</code> is <code>#t</code>, then the data in <code>filename</code> is saved in text mode as a <code>.sat</code> file.• If <code>textmode</code> is <code>#f</code>, then <code>filename</code> is loaded as a binary file.• If <code>textmode</code> is not specified, then the mode is determined by the extension of the file name. If the file name string ends in <code>.sab</code> or <code>.SAB</code>, then the file is saved in binary mode; otherwise, the file is saved in text mode. Argument type: BOOLEAN
part	Optional. Specifies the part to save; the default is the active part. Argument type: PART
with-history	Optional. After specifying the text mode, this argument specifies whether to save entity-specific the rolled back history data (that is, the history attached to the entity through an attribute). The default, <code>#f</code> , does not save the history. Argument type: BOOLEAN
mainline-only	Optional. Specifies whether to save rolled back branches. The default, <code>#f</code> , indicates that the entire history should be saved. The saved file can be restored at any time. Argument type: BOOLEAN

Appendix A: Commands

part:save-selection

part:save-selection

This Scheme function saves a list of entities to a file.

Part files can be saved in the format of an earlier version by setting the global option `save_version`.

Syntax

```
(part:save-selection ent-list filename [textmode=#f] [with-history=#f])
```

Returns

BOOLEAN

Arguments

Argument	Description
ent-list	Specifies a list of entities to be saved in the output file. Argument type: ENTITY ENTITY LIST
filename	Specifies a file name to be saved in the current working directory or specifies a path that includes the file name to be saved to a directory. Argument type: STRING
textmode	Optional. Describes the type of file to be loaded: <ul style="list-style-type: none">If <code>textmode</code> is <code>#t</code>, then the data in <code>filename</code> is saved in text mode as a <code>.sat</code> file.If <code>textmode</code> is <code>#f</code>, then <code>filename</code> is loaded as a binary file.If <code>textmode</code> is not specified, then the mode is determined by the extension of the file name. If the file name string ends in <code>.sab</code> or <code>.SAB</code>, then the file is saved in binary mode; otherwise, the file is saved in text mode. The saved file can be restored at any time. Argument type: BOOLEAN
with-history	Optional. This argument allows you to save entity-specific history (that is, history attached to the entity through an attribute). Argument type: BOOLEAN

Appendix A: Commands

part:set-name

part:set-name

This Scheme function assigns a name to the specified part, or to the active part if `part` is omitted. If the part is saved without specifying a file name, then the `part:save` Scheme extension uses the identification name assigned to the part as the file name of the saved part.

Syntax

```
(part:set-name [name] [part=active])
```

Returns

None

Arguments

Argument	Description
name	Optional. Specifies the required string identification of the part. Note: If <code>name</code> is omitted, then the name of the part is cleared. Argument type: STRING
part	Optional. Specifies the name of a part. Argument type: PART

Appendix A: Commands

position

position

This Scheme function creates a new position with x-, y-, and z-coordinates.

If the model has an origin other than the active work plane, this Scheme function returns the position relative to the active coordinate system in rectangular Cartesian coordinates.

A position is not saved with the part, but it is used to help define geometry. Positions are not displayed in Scheme.

A point is an entity that differs from a vertex in that it has no edge associations. Use `env:set-point-size` and `env:set-point-style` to change its appearance.

Syntax

```
(position x y z [space="model"])
```

Returns

BOOLEAN

Arguments

Argument	Description
x	Defines the x-coordinate relative to the active coordinate system. Argument type: REAL
y	Defines the y-coordinate relative to the active coordinate system. Argument type: REAL
z	Defines the z-coordinate relative to the active coordinate system. Argument type: REAL

Appendix A: Commands

position?

Argument	Description
space	<p>Optional. This argument defaults to <code>wcs</code> for the active (working) coordinate system. If no active work plane exists, then <code>space</code> defaults to <code>model</code>. The other optional <code>space</code> arguments return a gvector in the new coordinate system:</p> <ul style="list-style-type: none">The value <code>model</code> means that the x-, y-, and z-values are represented with respect to the model. If the model has an origin other than the active work plane, this returns the position relative to the active coordinate system in rectangular Cartesian coordinates.The value <code>polar</code> or <code>cylindrical</code> means that the x-, y-, and z-values are interpreted as the radial distance from the z-axis, the polar angle in degrees measured from the xz plane (using the right-hand rule), and the z-coordinate, respectively. This returns the x-, y-, and z-terms with respect to the active coordinate system.The value <code>spherical</code> means that the provided x-, y-, and z-values are the radial distance from the origin, the angle of declination from the z-axis in degrees, and the polar angle measured from the xz plane in degrees, respectively. This returns the x-, y-, and z-terms relative to the active coordinate system. <p>Argument type: STRING</p>

position?

This Scheme function determines whether a Scheme object is a position.

Syntax

`(position? object)`

Returns

BOOLEAN

Arguments

Argument	Description
object	<p>Specifies a Scheme object to be queried.</p> <p>Argument type: SCHEME OBJECT</p>

Appendix A: Commands

position:+

position:+

This Scheme function adds a position or gvector to a position or gvector, and returns the sum as a position.

Syntax

(position:+ arg1 arg2)

Returns

POSITION

Arguments

Argument	Description
arg1	Argument type: POSITION GVECTOR
arg2	Argument type: POSITION GVECTOR

position:-

This Scheme function subtracts a position or gvector from a position or gvector, and returns the result as a position.

Syntax

(position:- arg1 arg2)

Returns

POSITION

Arguments

Argument	Description
arg1	Argument type: POSITION GVECTOR
arg2	Argument type: POSITION GVECTOR

Appendix A: Commands

position:distance

position:distance

This Scheme function returns the distance between two positions.

Syntax

```
(position:distance position1 {position2 | ray})
```

Returns

REAL

Arguments

Argument	Description
position1	Defines the start location. Argument type: POSITION
position2	Defines the end location. Argument type: POSITION
ray	Defines the ray to calculate the distance. Argument type: RAY

Appendix A: Commands

position:set!

position:set!

This Scheme function sets the x-, y-, and z-components of a position.

Syntax

```
(position:set! position1 {x y z} | {position2})
```

Returns

POSITION

Arguments

Argument	Description
position1	Specifies the position to be set. Argument type: POSITION
x	Specify the x-value. If this value is specified, then it is copied into position1. Argument type: REAL
y	Specifies the y-value. If this value is specified, then it is copied into position1. Argument type: REAL
z	Specifies the z-value. If this value is specified, then it is copied into position1. Argument type: REAL
position2	If this argument is specified, then its position value is copied into position1. Argument type: POSITION

Appendix A: Commands

position:set-x!

position:set-x!

This Scheme function sets the x-component of a position. The coordinates are computed relative to the active coordinate system.

Syntax

```
(position:set-x! position x)
```

Returns

REAL

Arguments

Argument	Description
position	Identifies the original y- and z-values. Argument type: POSITION
x	Specifies the value to replace the original x-value specified in position. Argument type: REAL

position:set-y!

This Scheme function sets the y-component of a position. The coordinates are computed relative to the active coordinate system.

Syntax

```
(position:set-y! position y)
```

Returns

REAL

Arguments

Argument	Description
position	Identifies the original x- and z-values. Argument type: POSITION
y	Specifies the value to replace the original y-value specified in position. Argument type: REAL

Appendix A: Commands

position:set-z!

position:set-z!

This Scheme function sets the z-component of a position. The coordinates are computed relative to the active coordinate system.

Syntax

```
(position:set-z! position z)
```

Returns

REAL

Arguments

Argument	Description
position	Identifies the original x- and y-values. Argument type: POSITION
z	Specifies the value to replace the original z-value specified in position. Argument type: REAL

position:x

This Scheme function returns the x-component of a position relative to the active coordinate system.

Syntax

```
(position:x position)
```

Returns

REAL

Arguments

Argument	Description
position	Specifies a position. Argument type: POSITION

Appendix A: Commands

position:y

position:y

This Scheme function returns the y-component of a position relative to the active coordinate system.

Syntax

(position:y position)

Returns

REAL

Arguments

Argument	Description
position	Specifies a position. Argument type: POSITION

position:z

This Scheme function returns the z-component of a position relative to the active coordinate system.

Syntax

(position:z position)

Returns

REAL

Arguments

Argument	Description
position	Specifies a position. Argument type: POSITION

Appendix A: Commands

protect-all-contacts

protect-all-contacts

This Scheme function protects all 2D and 3D contacts against merging and deleting. If contacts are assigned to edges and faces using any of the `sdegeo` contact assignment commands, these contacts are protected. The 2D boundary simplification commands do not remove or merge contact edges, and also the 3D boundary regularization command (`(sde:bool-regularise)`) does not merge coplanar contact faces with adjoining faces.

If an application defines the edge or face contacts directly, by attaching the "2d-contact" or "3d-contact" attributes to the contact edges or faces, these contacts are not protected. In this case, you can use the `(protect-all-contacts)` command to add the necessary *no merge* attribute to the contact edges or faces.

Syntax

`(protect-all-contacts)`

Returns

BOOLEAN

Appendix A: Commands

random-sd

random-sd

This Scheme function returns a seeded random number. The seeded random number generator always generates the same random number sequence (normalized in [0-1]). This Scheme function uses a hard-coded, fixed seed. A different seed can be set by adding the initial seed to the function argument, for example, (`random-sd 987654321`). However, this must be done only once.

The next random number, which uses this seed, is generated by (`random-sd`). If you specify the seed, for example (`random-sd 987654321`), this always resets the random number sequence to the first value, which is generated by the given seed.

Syntax

```
(random-sd [initial-seed])
```

Returns

REAL random number, (normalized in [0-1])

Arguments

Argument	Description
initial-seed	Optional. Argument type: INTEGER

Examples

```
(random-sd)
```

Appendix A: Commands

remove-body-ABA

remove-body-ABA

This Scheme extension subtracts the specified input body from all the other existing bodies if there is an overlap between the input body and the other bodies.

The result is similar to the case where the input body is created when the “new replaces old” (“ABA”) automatic overlap removal rule is used. Overlapping bodies are not allowed in the final tessellated model. If overlaps are created, the overlaps must be removed manually.

Overlapping bodies can be created only if the automatic overlap handling is disabled or if some direct geometry manipulation is used, which is not detected by the automatic overlap removal procedure.

Syntax

(remove-body-ABA rbody)

Returns

None

Arguments

Argument	Description
rbody	Argument type: ENTITY

Appendix A: Commands

remove-body-BAB

remove-body-BAB

This Scheme extension subtracts all other existing bodies from the specified input body if there is an overlap between the input body and the other bodies.

The result is similar to the case where the input body is created when the “old replaces new” (“BAB”) automatic overlap removal rule is used. Overlapping bodies are not allowed in the final tessellated model. If overlaps are created, the overlaps must be removed manually.

Overlapping bodies can be created only if the automatic overlap handling is disabled or if some direct geometry manipulation is used, which is not detected by the automatic overlap removal procedure.

Syntax

```
(remove-body-BAB rbody)
```

Returns

None

Arguments

Argument	Description
rbody	Argument type: ENTITY

Appendix A: Commands

render:rebuild

render:rebuild

This Scheme function refreshes the display list for a view.

Use this Scheme function after changing viewing parameters.

Syntax

```
(render:rebuild [clear])
```

Returns

None

Arguments

Argument	Description
clear	Optional. If <code>clear</code> is <code>#t</code> or not present, then the view window is cleared and the wire frame is redisplayed from the display list contents. If <code>clear</code> is <code>#f</code> , then the view window is not cleared before redisplaying the display list (that is, no ‘new page’ is written to the file). Argument type: BOOLEAN

roll

This Scheme function can undo or redo geometry-related and doping-related, or refinement-related operations.

Syntax

```
(roll [num])
```

Returns

INTEGER (number of steps rolled)

Arguments

Argument	Description	Argument Type
num	Optional. Specifies the number of states the model can roll. A negative number means to roll to a previous state, and a positive number means to roll to a later state. Argument type: INTEGER	

Appendix A: Commands

sde:add-material

sde:add-material

This Scheme extension assigns the material and region attributes to a body.

Syntax

```
(sde:add-material body material-name region-name)
```

Returns

None

Arguments

Argument	Description
body	Defines the body. This is the first argument. Argument type: ACIS ENTITY
material-name	This argument must be a valid DATEX name, which is defined in the datexcodes.txt file. This argument is assigned as a material attribute to the body. When the tessellated boundary output is generated, the material attribute is the material of the region. Argument type: DATEXMAT
region-name	Defines the region name in the tessellated boundary output. Argument type: STRING

Appendix A: Commands

sde:back-coord

sde:back-coord

This Scheme extension returns the back coordinate of the specified body or body list. The back coordinate depends on the coordinate system used:

- For the unified coordinate system (UCS), the back coordinate is the minimum z-coordinate.
- For the DF–ISE coordinate system, the back coordinate is the minimum x-coordinate.

Syntax

(sde:back-coord body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY
body-list	Specifies a body list. Argument type: BODY LIST

sde:bg-image-transparency

This Scheme extension sets the transparency of the loaded image.

Syntax

(sde:bg-image-transparency transparency-value)

Returns

None

Arguments

Argument	Description
transparency-value	Specifies the value of the transparency. It must be between 0 and 1. Argument type: REAL

Appendix A: Commands

sde:bool-regularise

sde:bool-regularise

This Scheme function merges the coplanar faces of the specified 3D bodies. After (sde:bool-regularise) is applied, it is recommended to call (sde:fix-imprint) to ensure a conformal model.

Syntax

(sde:bool-regularise body-list)

Returns

None

Arguments

Argument	Description
body-list	Specifies a list of bodies. Argument type: BODY LIST

sde:bottom-coord

This Scheme extension returns the bottom coordinate of the specified body or body list. The bottom coordinate depends on the coordinate system used:

- For the UCS, the bottom coordinate is the maximum x-coordinate.
- For the DF–ISE coordinate system, the bottom coordinate is the minimum z-coordinate.

Syntax

(sde:bottom-coord body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY
body-list	Specifies a body list. Argument type: BODY LIST

Appendix A: Commands

sde:build-mesh

sde:build-mesh

This Scheme extension generates a 2D or 3D TDR tessellated boundary output and a mesh command file, and calls Sentaurus Mesh using a system command call. Sentaurus Structure Editor then waits for the meshing to be completed. No further interaction is possible during this time, until the meshing is completed.

The input argument string `file-basename` is first stripped from the file extension (if it is part of the input argument string). Then, if the stripped `file-basename` ends with `_msh`, that is removed as well. The TDR boundary file is saved as `file-basename_bnd.tdr` and the mesh command file is saved as `file-basename_msh.cmd`.

Note:

The default name of the TDR boundary file is `file-basename_bnd.tdr`.

The default name of the mesh command file is `file-basename_msh.cmd`.

Syntax

(sde:build-mesh `file-basename`)

Returns

Prints a message showing a successful or failed meshing.

Returns `#t` or `#f` to indicate success or failure, respectively.

Arguments

Argument	Description
<code>file-basename</code>	Argument type: STRING

sde:change-datex-color-scheme

This Scheme extension changes the DATEX color scheme used for rendering in the Sentaurus Structure Editor GUI. See [Selecting the DATEX Color Scheme on page 67](#).

Syntax

```
(sde:change-datex-color-scheme color-scheme)
```

Returns

BOOLEAN

Arguments

Argument	Description
color-scheme	Specifies the color scheme to use. The <code>datexcodes.txt</code> file contains two different color schemes for each DATEX material. Possible values are "Vivid" and "Classic". Argument type: STRING

Examples

Change to the Vivid color scheme:

```
(sde:change-datex-color-scheme "Vivid")
```

Change to the Classic color scheme:

```
(sde:change-datex-color-scheme "Classic")
```

sde:check-3d-license-status

This Scheme extension checks the status of the 3D license. It returns #t or #f depending on whether a 3D license is checked out.

Syntax

```
(sde:check-3d-license-status)
```

Returns

BOOLEAN

Appendix A: Commands

sde:check-model

sde:check-model

This Scheme extension checks a model for possible issues that are reported in a report file including:

- Mixed dimensional models
- Overlaps
- Multiple region names
- Entity check issues

Syntax

(sde:check-model [filename])

Returns

BODY LIST

Arguments

Argument	Description
filename	Optional. Specifies the name of a file. The default file name is sde-model-debug.log. Argument type: STRING

sde:checkout-3d-license

This Scheme extension checks out a 3D license explicitly. By default, Sentaurus Structure Editor starts with a 2D license and only checks out (automatically) a 3D license when it is needed (for 3D solid creation, 2D to 3D operations, process emulation operations, and so on).

Syntax

(sde:checkout-3d-license)

Returns

BOOLEAN

Appendix A: Commands

sde:clear

sde:clear

This Scheme extension clears the database of Sentaurus Structure Editor and restores all settings to their initial values, and it should always be used as the first call when a new script is executed that requires a clean database. This Scheme extension also restores the view window to its initial settings.

Caution:

This Scheme extension does not clear the Scheme interpreter. If global Scheme variables are used, then they are still initialized and valid after an (`sde:clear`) call.

Syntax

(`sde:clear`)

Returns

None

Appendix A: Commands

sde:create-bg-image

sde:create-bg-image

This Scheme extension loads an image file (GIF). The image is not shown automatically; this Scheme extension must be called to show the image.

Note:

The image loader supports only images in the older GIF format, GIF87a. Most current graphic programs use the GIF format GIF89a. See [Background Image Loader on page 333](#).

Syntax

```
(sde:create-bg-image filename horizontal-size vertical-size)
```

Returns

None

Arguments

Argument	Description
filename	Specifies the name of a file. Argument type: STRING
horizontal-size	Defines the GUI size of the image – horizontal value. Argument type: REAL
vertical-size	Defines the GUI size of the image – vertical value. Argument type: REAL

Appendix A: Commands

sde:create-dialog

sde:create-dialog

This Scheme extension creates a dialog box and registers a dialog object, and returns the dialog entity ID. (The dialog box is handled as a Scheme object.)

All subsequent sde:dialog functions use this dialog ID to add input fields, a bitmap image, and an **OK** button to the dialog box.

Syntax

```
(sde:create-dialog dlg-name)
```

Returns

Dialog ID

Arguments

Argument	Description
dlg-name	Specifies the name of the dialog box. Argument type: STRING

Examples

```
(define dlg (sde:create-dialog "2D MOSFET DEVICE"))
```

Appendix A: Commands

sde:define-parameter

sde:define-parameter

The Scheme extension defines a Scheme parameter. The defined parameter can be used later in any Scheme command. The parameter value can be accessed by typing `parameter-name`. The parameter can hold any Scheme type variable. Typically, it is a `REAL` or `STRING` type. For `REAL` values, two optional values also can be added: `min-value` and `max-value`. This might be useful when you write a parametric script, where a given parameter can be used and modified only within a certain range.

If `parameter-name` is `width`, for example, `width_min` will contain `min-value` and `width_max` will contain `max-value`. A Scheme variable also can be defined using the Scheme command `define`.

The difference is that the Scheme extension `sde:save-parameters` saves only those Scheme variables that were defined using the Scheme extension `sde:define-parameter`.

Syntax

```
(sde:define-parameter parameter-name value [min-value max-value])
```

Returns

BOOLEAN

Arguments

Argument	Description
<code>parameter-name</code>	Argument type: STRING
<code>value</code>	Argument type: REAL STRING
<code>min-value</code>	Optional. Argument type: REAL
<code>max-value</code>	Optional. Argument type: REAL

Appendix A: Commands

sde:delay-graphics-update

sde:delay-graphics-update

This Scheme extension switches on or off the refreshing or updating of the view window. By default, automatic refreshing of the view window is switched on. When a new body is created or the model is changed in any way that affects the graphical representation of the model, the view window is refreshed automatically.

In this way, the display always reflects the actual state of the database of the modeler. This behavior can be changed and the graphical update can be suppressed using this Scheme extension. By switching on the update option, the buffer is flushed and the screen is updated.

Syntax

```
(sde:delay-graphics-update flag)
```

Returns

None

Arguments

Argument	Description
flag	Argument type: BOOLEAN

sde:delay-graphics-update?

This Scheme extension returns the status of the `sde:delay-graphics-update` Scheme extension.

Syntax

```
(sde:delay-graphics-update?)
```

Returns

BOOLEAN

Appendix A: Commands

sde:delete-bg-image

sde:delete-bg-image

This Scheme extension removes the previously loaded image from memory.

Syntax

```
(sde:delete-bg-image)
```

Returns

None

sde:delete-materials

This Scheme extension deletes the geometric bodies with the specified DATEX material.

Syntax

```
(sde:delete-materials materials)
```

Returns

None

Arguments

Argument	Description
materials	Argument type: DATEXMAT DATEXMAT LIST

Appendix A: Commands

sde:delete-parameter

sde:delete-parameter

This Scheme extension deletes a previously defined parameter.

The parameter must be previously defined using `sde:define-parameter` (see [sde:define-parameter on page 444](#)).

Syntax

```
(sde:delete-parameter parameter-name)
```

Returns

BOOLEAN

Arguments

Argument	Description
parameter-name	Argument type: STRING

Appendix A: Commands

sde:dialog-add-input

sde:dialog-add-input

This Scheme extension adds an input field to a dialog box. See [sde:create-dialog on page 443](#).

Caution:

No default value can be assigned to the input fields and variables.

Syntax

```
(sde:dialog-add-input dlgid varname inputstrfield inputtype  
[default-value])
```

Returns

None

Arguments

Argument	Description
dlgid	Specifies the ID of the dialog box. Argument type: STRING DIALOG TYPE (object returned by sde:create-dialog)
varname	Specifies the Scheme variable name that is passed to the executed Scheme script. Argument type: STRING
inputstrfield	The value of this argument is displayed in the input dialog box. Argument type: STRING
inputtype	Specifies the type of input. Possible values are INTEGER, REAL, and STRING. Argument type: STRING
default-value	Optional. Specifies a default value for the field. Argument type: VALUE (according to inputtype)

Examples

```
(sde:dialog-add-input dlg "width1" "Base Width" "real")
```

Appendix A: Commands

sde:dialog-add-pixmap

sde:dialog-add-pixmap

This Scheme extension adds a predefined bitmap image to a dialog box. The bitmap image must be scaled to the proper size before it is added to the dialog box. See [sde:create-dialog on page 443](#).

Syntax

```
(sde:dialog-add-pixmap dlgid bmpfilename)
```

Returns

None

Arguments

Argument	Description
dlgid	Specifies the ID of the dialog box. Argument type: STRING DIALOG TYPE (object returned by sde:create-dialog)
bmpfilename	Specifies the name of a bitmap image. Argument type: STRING

Examples

```
(sde:dialog-add-pixmap dlg "mosfet2d.bmp")
```

Appendix A: Commands

sde:dialog-delete

sde:dialog-delete

This Scheme extension removes a predefined dialog box from memory. See [sde:create-dialog on page 443](#).

Syntax

(sde:dialog-delete dlgid)

Returns

None

Arguments

Argument	Description
dlgid	Specifies the ID of the dialog box. Argument type: STRING DIALOG TYPE (object returned by sde:create-dialog)

Examples

(sde:dialog-delete dlg)

Appendix A: Commands

sde:dialog-ok-command

sde:dialog-ok-command

When the **OK** button of the dialog box is clicked, this Scheme extension is called with the supplied argument list. See [sde:create-dialog on page 443](#).

Syntax

```
(sde:dialog-ok-command dlgid schemefnname argument-list)
```

Returns

None

Arguments

Argument	Description
dlgid	Specifies the ID of the dialog box. Argument type: STRING DIALOG TYPE (object returned by sde:create-dialog)
schemefnname	Specifies the name of a Scheme function. Argument type: STRING
argument-list	Specifies an argument list in double quotation marks. Argument type: STRING LIST

Examples

```
(sde:dialog-ok-command dlg "create-2d-mosfet"  
"width1 width2 width3 height1 height2 height3")
```

Appendix A: Commands

sde:dialog-show

sde:dialog-show

This Scheme extension displays a predefined dialog box. See [sde:create-dialog on page 443](#).

Syntax

(sde:dialog-show dlgid)

Returns

None

Arguments

Argument	Description
dlgid	Specifies the ID of the dialog box. Argument type: STRING DIALOG TYPE (object returned by sde:create-dialog)

Examples

(sde:dialog-show dlg)

sde:display

This Scheme extension displays the specified string in the command-line window. A new line character is required to send the contents of the sde:display buffer to the command-line window.

This Scheme extension is useful if you write your own Scheme scripts and the function needs to display some information.

Syntax

(sde:display string)

Returns

None

Arguments

Argument	Description
string	Specifies a string to be displayed. Argument type: STRING

Appendix A: Commands

sde:display-err

sde:display-err

This Scheme extension writes a specified string in a terminal window. A new line character is added to the argument string.

Syntax

(sde:display-err string)

Returns

None

Arguments

Argument	Description
string	Specifies a string to be displayed. Argument type: STRING

Appendix A: Commands

sde:display-std

sde:display-std

This Scheme extension writes a specified string in a terminal window. A new line character is added to the argument string.

For the design-of-experiments (DoE) parameter extraction of Sentaurus Workbench, this Scheme extension must be used to send the information to Sentaurus Workbench. (Sentaurus Workbench collects the DoE-marked output from standard output.)

Syntax

(sde:display-std string)

Returns

None

Arguments

Argument	Description
string	Specifies a string to be displayed. Argument type: STRING

Examples

For the DoE parameter extraction, the output string must contain the `DOE:` string, followed by the variable name and the value of the variable:

(sde:display-std "DOE: VARNAME VARVALUE")

For example:

```
(sde:display-std "DOE: width 2.5")
(sde:display-std "DOE: doping @value@")
```

Note:

You must initialize the `VARNAME` variable in the Sentaurus Workbench script to avoid preprocessing errors. For example:

```
#set width 0
#set doping 0
```

Appendix A: Commands

sde:draw-ruler

sde:draw-ruler

This Scheme extension draws a user-defined ruler that can be displayed in the GUI.

You can hide user-defined rulers in the GUI using the `sde:hide-ruler` Scheme extension. See [sde:hide-ruler on page 471](#).

Note:

If you change the parameters of a user-defined ruler, then you must use the `sde:hide-ruler` Scheme extension *first* to erase the previously defined ruler.

After this step, the ruler can be redrawn with the new parameters using the `sde:draw-ruler` Scheme extension.

Syntax

```
(sde:draw-ruler "startpos" startpos "alength" alength "adir" adir  
    "startval" startval "endval" endval  
    ["numdiv" numdiv] ["numdiv2" numdiv2]  
    ["npos" npos] ["acolor" acolor]  
    ["tcolor" tcolor])
```

Returns

BOOLEAN

Arguments

Argument	Description
startpos	Defines the starting position of the ruler in 3D space. Argument type: POSITION
alength	Specifies the length of the ruler. Argument type: REAL
adir	Specifies the ruler direction, where: <ul style="list-style-type: none">• "adir" 0 specifies the global x-axis direction.• "adir" 1 specifies the y-axis direction.• "adir" 2 specifies the z-axis direction. Argument type: INTEGER
startval	Defines the start value of the coordinate axes. Argument type: REAL
endval	Defines the end value of the coordinate axes. Argument type: REAL
numdiv	Optional. Specifies the number of main tick divisions. The default is 5. Argument type: INTEGER

Appendix A: Commands

sde:draw-ruler

Argument	Description
numdiv2	Optional. Specifies the number of subdivisions in each main tick division, where the ruler will draw the subdivision ticks. The default is 2. Argument type: INTEGER
npos	Optional. Specifies the position of the displayed numbers: For an x-axis ruler ("adir" 0): <ul style="list-style-type: none">"npos" 1 specifies that the numbers are drawn below the axis."npos" 0 specifies that the numbers are drawn above the axis. For a y-axis ruler ("adir" 1): <ul style="list-style-type: none">"npos" 1 indicates the right y-axis."npos" 0 indicates the left y-axis. The default is 1. Argument type: INTEGER
acolor	Optional. Specifies the axis color and takes a color:rgb value. This color can also be set by specifying BLACK, BLUE, CYAN, GREEN, MAGENTA, RED, WHITE, or YELLOW. Argument type: COLOR
tcolor	Optional. Specifies the text color and takes a color:rgb value. This color can also be set by specifying BLACK, BLUE, CYAN, GREEN, MAGENTA, RED, WHITE, or YELLOW. Argument type: COLOR

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0)(position 5 4 2) "Silicon" "xx")
(sde:draw-ruler "startpos" (position 0 4 0) "alength" 6 "adir" 0
    "startval" 0 "endval" 6 "numdiv" 6 "numdiv2" 5 "npos" 1
    "acolor" GREEN "tcolor" BLUE)
(sde:draw-ruler "startpos" (position 5 0 0) "alength" 5 "adir" 1
    "startval" 0 "endval" 5 "numdiv" 5 "numdiv2" 5 "npos" 1
    "acolor" GREEN "tcolor" BLUE)
(sde:draw-ruler "startpos" (position 5 4      0) "alength" 3 "adir" 2
    "startval" 0 "endval" 3 "numdiv" 3 "numdiv2" 5 "npos" 1
    "acolor" GREEN "tcolor" BLUE)
```

Appendix A: Commands

sde:dump-non-default-options

sde:dump-non-default-options

This Scheme extension saves a text file containing all the ACIS options that are set to nondefault values. This helps to debug and report problems because, in some cases, nondefault option settings can cause problems or can trigger a different behavior than expected.

Syntax

```
(sde:dump-non-default-options filename)
```

Returns

BOOLEAN

Arguments

Argument	Description
filename	Specifies a file name. Argument type: STRING

sde:exact-coords?

This Scheme extension checks the status of the Exact Coordinates mode.

If exact coordinates are used, then this Scheme extension returns #t.

If the Exact Coordinates mode is switched off, then it returns #f.

Syntax

```
(sde:exact-coords?)
```

Returns

BOOLEAN

Appendix A: Commands

sde:extract-tdr-boundary

sde:extract-tdr-boundary

This Scheme extension extracts the boundary and saves it in a TDR boundary file from a TDR grid file. This Scheme extension uses Sentaurus Mesh through a system call to extract the boundary.

Syntax

```
(sde:extract-tdr-boundary fname [file-name])
```

Returns

BOOLEAN

Arguments

Argument	Description
fname	Specifies the name of a TDR boundary file. Argument type: STRING
file-name	Optional. Specifies a file name for the extracted TDR boundary. Argument type: STRING

sde:fix-imprint

This Scheme extension generates a conformal model by imprinting the model edges and faces to neighboring bodies.

The generated model must be tessellated before meshing, and several different conformity requirements must be satisfied. One of the most important requirements is that neighboring bodies share the boundary edges (in two dimensions) and the boundary faces (in three dimensions). Before the tessellated model is generated, Sentaurus Structure Editor automatically imprints the edges and faces of the neighboring bodies to each other. During model generation, the modeler tries to create a conformal model in each modeling step, but several operations can destroy model conformity by merging collinear edges and coplanar faces. In this case, the sde:fix-imprint Scheme extension restores model conformity.

Syntax

```
(sde:fix-imprint)
```

Returns

None

Appendix A: Commands

sde:fix-orientation

sde:fix-orientation

This Scheme extension modifies the orientation of 3D regions, with surface normals pointing ‘inside’ the body. Mainly native ACIS models, generated by other CAD tools and imported as .sat files, might have this problem. Direct-geometry generation functions in Sentaurus Structure Editor always generate 3D bodies with positive volume and surface normals pointing *outside*.

Syntax

```
(sde:fix-orientation)
```

Returns

None

sde:front-coord

This Scheme extension returns the front coordinate of the specified body or body list. The front coordinate depends on the coordinate system used:

- For the UCS, the front coordinate is the maximum z-coordinate.
- For the DF–ISE coordinate system, the front coordinate is the maximum x-coordinate.

Syntax

```
(sde:front-coord body | body-list)
```

Returns

REAL

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY
body-list	Specifies a list of bodies. Argument type: BODY LIST

Appendix A: Commands

sde:ft_scalar

sde:ft_scalar

This Scheme extension writes the specified string or string–value pair in a terminal window. A new line character is added to the argument string.

For the design-of-experiments (DoE) parameter extraction of Sentaurus Workbench, this Scheme extension must be used to send the information to Sentaurus Workbench. (Sentaurus Workbench collects the DoE-marked output from standard output.)

Syntax

```
(sde:ft_scalar string [value])
```

Returns

None

Arguments

Argument	Description
string	Specifies a string to be written in a terminal window. Argument type: STRING
value	Optional. Specifies a string–value pair to be written in a terminal window. Argument type: STRING REAL

Examples

```
(sde:ft_scalar "aval 3")
```

```
(sde:ft_scalar "aval" 3)
```

sde:get-backwards-compatibility

This Scheme extension returns the value of the given backward-compatibility parameter. Backward-compatibility parameters are specific, internally defined, parameters that are used to reproduce certain defined functionality from previous releases of Sentaurus Structure Editor.

For the available backward-compatibility parameters, see [sde:set-backwards-compatibility on page 499](#).

Syntax

```
(sde:get-backwards-compatibility backcompat-param)
```

Returns

Value of the given backward-compatibility parameter `backcompat-param`; otherwise, returns `#f` if the given `backcompat-param` does not exist

Arguments

Argument	Description
<code>backcompat-param</code>	Specifies a backward-compatibility parameter. Argument type: STRING

sde:get-datex-color-scheme

This Scheme extension returns the current setting of the DATEX color scheme.

Syntax

```
(sde:get-datex-color-scheme)
```

Returns

"Vivid" or "Classic"

Appendix A: Commands

sde:get-default-material

sde:get-default-material

This Scheme extension returns the selected DATEX material in a string. The DATEX material can be selected from the GUI using the material list or the Scheme extension sde:set-default-material.

Syntax

(sde:get-default-material)

Returns

None

sde:get-view-params

This Scheme extension takes no arguments and returns a list, which contains all the necessary data to restore the GUI view to the recorded state.

The Scheme extension sde:set-view-params can be used to restore the settings (see [sde:set-view-params on page 506](#)).

Syntax

(sde:get-view-params)

Returns

A list containing:

eye position	POSITION
target position	POSITION
up-vector	GVECTOR
width	REAL
height	REAL
projection	STRING

Examples

```
(define myview (sde:get-view-params))
(sde:set-view-params myview)
```

Appendix A: Commands

sde:gui-get-integer

sde:gui-get-integer

This Scheme extension displays a dialog box that can be used to enter an integer variable.

When the **OK** button is clicked, the function returns the entered integer, or #f if incorrect input was given or the **Cancel** button was clicked.

Note:

This Scheme extension can be used in GUI mode only.

Syntax

```
(sde:gui-get-integer variable-name default-value [dialog-name])
```

Returns

INTEGER OR #F

Arguments

Argument	Description
variable-name	Specifies the name of a variable to be displayed. Argument type: STRING
default-value	Specifies the default value of the variable. Argument type: INTEGER
dialog-name	Optional. Defines the name of the input dialog box. Argument type: STRING

Appendix A: Commands

sde:gui-get-real

sde:gui-get-real

This Scheme extension displays a dialog box that can be used to enter a real variable.

When the **OK** button is clicked, the function returns the entered integer, or #f if incorrect input was given, or the **Cancel** button was clicked.

Note:

This Scheme extension can be used in GUI mode only.

Syntax

```
(sde:gui-get-real variable-name default-value [dialog-name])
```

Returns

REAL OR #f

Arguments

Argument	Description
variable-name	Specifies the name of a variable to be displayed. Argument type: STRING
default-value	Specifies the default value of the variable. Argument type: INTEGER
dialog-name	Optional. Defines the name of the input dialog box. Argument type: STRING

Appendix A: Commands

sde:gui-get-string

sde:gui-get-string

This Scheme extension displays a dialog box that can be used to enter a string variable.

When the **OK** button is clicked, the function returns the entered string value, or #f if incorrect input was given or the **Cancel** button was clicked.

Note:

This Scheme extension can be used in GUI mode only.

Syntax

```
(sde:gui-get-string variable-name default-value [dialog-name])
```

Returns

STRING or #f

Arguments

Argument	Description
variable-name	Specifies the name of a variable to be displayed. Argument type: STRING
default-value	Specifies the default value of the variable. Argument type: STRING
dialog-name	Optional. Defines the name of the input dialog box. Argument type: STRING

Appendix A: Commands

sde:hide

sde:hide

This Scheme extension removes the specified entity or entities from the view window, and the view window is refreshed. If the entities are already removed from the view window, then there is no change in the view window.

Syntax

```
(sde:hide entity | entity-list)
```

Returns

None

Arguments

Argument	Description
entity	Specifies an entity to be removed. Argument type: ENTITY
entity-list	Specifies a list of entities to be removed. Argument type: ENTITY LIST

Examples

```
(sde:clear)
(define r1 (sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
                                      "Photoresist" "r_1"))
(define r2 (sdegeo:create-rectangle (position 2 0 0) (position 3 1 0)
                                      "Silicon" "r_2"))
(define r3 (sdegeo:create-rectangle (position 4 0 0) (position 5 1 0)
                                      "Resist" "r_3"))
(define r4 (sdegeo:create-rectangle (position 6 0 0) (position 7 1 0)
                                      "PolySilicon" "r_4"))
(sde:hide r1)
(sde:hide (list r3 r4))
```

sde:hide-bg-image

This Scheme extension hides the previously loaded image.

Syntax

```
(sde:hide-bg-image)
```

Returns

None

Appendix A: Commands

sde:hide-contact

sde:hide-contact

This Scheme extension removes from the view window all regions that have contacts defined with the specified contact attributes. Contacts are identified with the attribute name `contact`. In two dimensions, contacts are edges and, in three dimensions, contacts are faces.

Syntax

```
(sde:hide-contact contact-name | contact-name-list)
```

Returns

None

Arguments

Argument	Description
contact-name	Argument type: STRING
contact-name-list	Argument type: STRING LIST

sde:hide-interface

This Scheme extension removes from the view all regions that have interfaces defined with the specified interface attributes. Interfaces are identified with the attribute name `interface`. In two dimensions, interfaces are edges between neighboring regions and, in three dimensions, interfaces are faces between neighboring regions.

Syntax

```
(sde:hide-interface interface-name | interface-name-list)
```

Returns

None

Arguments

Argument	Description
interface-name	Argument type: STRING
interface-name-list	Argument type: STRING LIST

Appendix A: Commands

sde:hide-mask

sde:hide-mask

This Scheme extension removes the specified masks from the view window. Masks are planar wire bodies that have the specified mask names as *mask name* attributes.

Syntax

```
(sde:hide-mask mask-name | mask-name-list)
```

Returns

None

Arguments

Argument	Description
mask-name	Specifies the name of a mask. Argument type: STRING
mask-name-list	Specifies a list of mark names. Argument type: STRING_LIST

Appendix A: Commands

sde:hide-material

sde:hide-material

This Scheme extension removes regions that have the specified material names as material attributes from the view window. The region is removed from the view window, and the view window is refreshed.

If the region is already removed from the view window, then there is no change in the view window.

Syntax

```
(sde:hide-material datex-material | datex-material-list)
```

Returns

None

Arguments

Argument	Description
datex-material	Specifies a DATEX material. Argument type: DATEXMAT
datex-material-list	Specifies a list of DATEX materials. Argument type: DATEXMAT LIST

Examples

```
(sde:clear)
(sdgeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Photoresist" "region_1")
(sdgeo:create-rectangle (position 2 0 0) (position 3 1 0) "Silicon"
    "region_2")
(sdgeo:create-rectangle (position 4 0 0) (position 5 1 0) "Resist"
    "region_3")
(sdgeo:create-rectangle (position 6 0 0) (position 7 1 0)
    "PolySilicon" "region_4")
(sde:hide-material "Silicon")
```

Appendix A: Commands

sde:hide-region

sde:hide-region

This Scheme extension removes the specified regions from the view window. The bodies that have the specified region names as region attributes are removed from the view window, and the view window is refreshed.

If the region is already removed from the view window, there is no change in the view window.

Syntax

```
(sde:hide-region region-name | region-name-list)
```

Returns

None

Arguments

Argument	Description
region-name	Specifies a region name, which must be an existing region name. Argument type: STRING
region-name-list	Specifies a list of region names, which must be existing region names. Argument type: STRING LIST

Examples

```
(sde:clear)
(sdgeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Photoresist" "region_1")
(sdgeo:create-rectangle (position 2 0 0) (position 3 1 0) "Silicon"
    "region_2")
(sdgeo:create-rectangle (position 4 0 0) (position 5 1 0) "Resist"
    "region_3")
(sdgeo:create-rectangle (position 6 0 0) (position 7 1 0)
    "PolySilicon" "region_4")
(sde:hide-region "region_1")
(sde:hide-region (list "region_3" "region_4"))
```

Appendix A: Commands

sde:hide-ruler

sde:hide-ruler

This Scheme extension hides all user-defined rulers in the GUI. See [sde:draw-ruler](#) on page 455.

Syntax

(sde:hide-ruler)

Returns

BOOLEAN

sde:info

This Scheme extension returns the attributes of Sentaurus Structure Editor attached to the specified bodies. It displays the entity number, and material and region attributes that are attached to the body.

Syntax

(sde:info entity-list)

Returns

STRING

Arguments

Argument	Description
entity-list	Argument type: ENTITY LIST

Appendix A: Commands

sde:left-coord

sde:left-coord

This Scheme extension returns the left coordinate of the specified body or body list. The left coordinate is the minimum y-coordinate.

Syntax

(sde:left-coord body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body to be queried. Argument type: BODY
body-list	Specifies a list of bodies to be queried. Argument type: BODY LIST

sde:load-sat

This Scheme extension loads a native ACIS .sat file, with correct overlap handling.

The difference between `part:load` and `sde:load-sat` is that the latter handles overlapping regions correctly, according to the active Boolean setting. If a .sat file is loaded into the modeler, which already has some bodies defined, then the overlaps are resolved according to the Boolean setting.

Syntax

(sde:load-sat file-name)

Returns

None

Arguments

Argument	Description
file-name	Argument type: STRING

Appendix A: Commands

sde:material-type

sde:material-type

This Scheme extension returns the group type from the `datexcodes.txt` file for the given material. The group type can be Semiconductor, or Conductor, or Insulator.

Syntax

(`sde:material-type` `material-name`)

Returns

group-name or an empty string if the specified material is not found in the `datexcodes.txt` file

Arguments

Argument	Description
<code>material-name</code>	Argument type: DATEXMAT

sde:max-x

This Scheme extension returns the maximum x-dimension of the specified body or body list.

Syntax

(`sde:max-x` `body` | `body-list`)

Returns

REAL

Arguments

Argument	Description
<code>body</code>	Specifies a body to be queried. Argument type: BODY
<code>body-list</code>	Specifies a list of bodies to be queried. Argument type: BODY LIST

Appendix A: Commands

sde:max-y

sde:max-y

This Scheme extension returns the maximum y-dimension of the specified body or body list.

Syntax

(sde:max-y body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body to be queried. Argument type: BODY
body-list	Specifies a list of bodies to be queried. Argument type: BODY LIST

sde:max-z

This Scheme extension returns the maximum z-dimension of the specified body or body list.

Syntax

(sde:max-z body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body to be queried. Argument type: BODY
body-list	Specifies a list of bodies to be queried. Argument type: BODY LIST

Appendix A: Commands

sde:merge-materials

sde:merge-materials

This Scheme extension merges (unites) a copy of the geometric bodies with the specified DATEX materials, and the newmaterial DATEX material is assigned to the new united body.

Note:

The new united body overlaps with the original bodies. The entity:delete Scheme function must be used to delete the original bodies, or the overlaps must be removed by other operations before the tessellated boundary file is created.

Syntax

```
(sde:merge-materials materials newmaterial)
```

Returns

None

Arguments

Argument	Description
materials	Specifies a DATEX material or a list of DATEX materials. Argument type: DATEXMAT DATEXMAT LIST
newmaterial	Specifies a DATEX material to assign to the new united body. Argument type: DATEXMAT

Appendix A: Commands

sde:min-x

sde:min-x

This Scheme extension returns the smallest x-dimension of the specified body or body list.

Syntax

(sde:min-x body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body to be queried. Argument type: BODY
body-list	Specifies a list of bodies to be queried. Argument type: BODY LIST

sde:min-y

This Scheme extension returns the smallest y-dimension of the specified body or body list.

Syntax

(sde:min-y body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body to be queried. Argument type: BODY
body-list	Specifies a list of bodies to be queried. Argument type: BODY LIST

Appendix A: Commands

sde:min-z

sde:min-z

This Scheme extension returns the smallest z-dimension of the specified body or body list.

Syntax

(sde:min-z body | body-list)

Returns

REAL

Arguments

Argument	Description
body	Specifies a body to be queried. Argument type: BODY
body-list	Specifies a list of bodies to be queried. Argument type: BODY LIST

sde:new-region-name

This Scheme extension returns an automatically generated region name `region_%N`, where `%N` is the actual region counter. If the region counter is set correctly, then this Scheme extension always returns a unique region name.

Syntax

(sde:new-region-name)

Returns

STRING

sde:off-lights

This Scheme extension switches off the lights in the view window.

Syntax

(sde:off-lights)

Returns

None

Appendix A: Commands

sde:offset-mask

sde:offset-mask

This Scheme extension offsets (or biases) the specified mask.

Syntax

```
(sde:offset-mask mask-name offset-distance)
```

Returns

None

Arguments

Argument	Description
mask-name	Specifies the name of a mask. Argument type: STRING
offset-distance	Specifies the offset distance. Argument type: REAL

sde:on-lights

This Scheme extension switches on the lights in the view window.

Syntax

```
(sde:on-lights)
```

Returns

None

Appendix A: Commands

sde:open-model

sde:open-model

This Scheme extension loads a complete Sentaurus Structure Editor model.

Syntax

(sde:open-model filename)

Returns

None

Arguments

Argument	Description
filename	Specifies the name of a file to load. Argument type: STRING

Description

This Scheme extension is the counterpart of `sde:save-model`. It restores the complete state of the modeler. First, `filename.sat` is loaded if it exists. It is a native ACIS `.sat` file that contains the geometric description of the model, including Ref/Eval windows, with all the attributes and contacts of Sentaurus Structure Editor that are attached.

The `.sat` file is loaded using the `sde:part-load` command, so the automatic overlap resolution works the same way as for the `sde:part-load` command. Check the usage of the global Boolean variable `aut-overlap-control` in [sde:part-load on page 480](#).

If the working directory does not include a `filename.sat` file, the loader checks for the existence of a TDR boundary file, `filename_bnd.tdr` (or if it is not found, the loader checks for `filename.tdr`). If a TDR boundary file is found, this file is imported into Sentaurus Structure Editor.

A `filename.scm` file also is loaded if it exists. This file contains all of the defined parameters. Finally, a mesh command file, `filename_msh.cmd`, is loaded if it exists. If this file does not exist, the loader checks for `filename.cmd` and loads it if it is found. This file contains all refinement-related and doping-related information.

See Also

Sentaurus™ Mesh User Guide, Chapter 2

Appendix A: Commands

sde:part-load

sde:part-load

This Scheme extension loads a native ACIS .sat file, with correct automatic overlap handling.

Syntax

```
(sde:part-load filename [loadoption])
```

Returns

None

Arguments

Argument	Description
filename	Specifies the name of .sat file. Argument type: STRING
loadoption	Optional. Argument type: BOOLEAN

Description

A global Boolean variable, aut-overlap-control, determines whether to load .sat files with automatic overlap control. If it is #f (default value), the newly loaded .sat file is loaded without checking for overlaps. If it is set to #t, the newly loaded .sat file is loaded in such a way that the default Boolean overlap setting will be taken into consideration and the overlaps will be resolved.

Note:

Use (sdegeo:get-default-boolean) to check the current status of the automatic overlap-handling setting (see [sdegeo:get-default-boolean on page 661](#)).

Since the automatic overlap control slows down model loading considerably, it is helpful to use the default #f value for aut-overlap-control if it is known that the loaded files contain only conformal, non-overlapping regions.

When aut-overlap-control #t is set, the overlaps are checked and resolved within the loaded model.

Appendix A: Commands

sde:pick-point-on-wp

sde:pick-point-on-wp

This Scheme extension selects a point on the view window.

The global xyz coordinates of the selected point are returned as a position entity. The point is always selected in the active work plane. (This Scheme extension draws a cross-hair cursor on the screen, and a horizontal line and vertical line also help the positioning.) In addition, the exact coordinates and snap-to-grid options can be used together with this Scheme extension.

Syntax

(sde:pick-point-on-wp)

Returns

POSITION

sde:pick-two-points-on-wp

This Scheme extension selects two points on the view window.

The global xyz coordinates of the selected points are returned as a list of two position entities. The points are always selected in the active work plane. (This Scheme extension draws a cross-hair cursor on the screen, and a horizontal and vertical line also help the positioning.)

In addition, the exact coordinates and snap-to-grid options can be used together with this Scheme extension. The first point is selected by pressing the mouse button; drag to draw a line from the first point. When the mouse button is released, the two positions are returned.

Syntax

(sde:pick-two-points-on-wp)

Returns

POSITION LIST

Appendix A: Commands

sde:post-message

sde:post-message

This Scheme extension displays a message in a dialog box. The **OK** button of the dialog box must be clicked to cancel the message.

Syntax

(sde:post-message message)

Returns

None

Arguments

Argument	Description
message	Specifies a message to be displayed in a dialog box. Argument type: STRING

Examples

(sde:post-message "Hello World")

sde:project-name

This Scheme extension returns the project name, which is used in the Build Mesh dialog box to save the created mesh file. The project name can be set using the sde:set-project-name command. See [sde:set-project-name on page 502](#).

Syntax

(sde:project-name)

Returns

STRING

Appendix A: Commands

sde:refresh

sde:refresh

This Scheme extension refreshes and rebuilds the view window. It discards the existing entity display list and regenerates it from the entity data. This is useful when the view zooms in on entities, because the display might appear jagged.

This Scheme extension destroys the existing display list, regenerates it from the entities at the new zoom level, and redisplays the entities in all views associated with the part.

If only GUI actions are used to manipulate the view and modeler, then the view is always updated and there is no need to call (sde:refresh).

Syntax

(sde:refresh)

Returns

None

Appendix A: Commands

sde:rename-regions

sde:rename-regions

This Scheme extension automatically renames regions based on spatial criteria.

The geometric bodies are ordered, based on the specified direction, and a region name sequence is attached to the bodies.

Syntax

```
(sde:rename-regions entity-list direction)
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies a list of entities. Argument type: ENTITY LIST
direction	Specifies the direction in which the geometric bodies are ordered. Options are "+x" "-x" "+y" "-y" "+z" "-z". For example, if the direction is "+x", then bodies are ordered based on their minimal bounding box x-coordinates. Then, a "region_RC" region name is attached to each body, where RC is a numeric region counter, starting from 0. The "region_0" region name is attached to the body with the smallest minimal bounding box x-coordinate, "region_1" is the body with the second minimal bounding box x-coordinate, and so on. If the direction is "-x", then the largest bounding box x-coordinates are ordered in a decreasing sequence. Similar rules apply to "+y", "-y", "+z", and "-z".

sde:restore-cursor

This Scheme extension reverts the pointer to the default pointer.

Syntax

```
(sde:restore-cursor)
```

Returns

None

Appendix A: Commands

sde:right-coord

sde:right-coord

This Scheme extension returns the right coordinate of the specified body or body list. The right coordinate is the maximum y-coordinate.

Syntax

```
(sde:right-coord body | body-list)
```

Returns

REAL

Arguments

Argument	Description
body	Specifies a body to be queried. Argument type: BODY
body-list	Specifies a list of bodies to be queried. Argument type: BODY LIST

Appendix A: Commands

sde:save-model

sde:save-model

This Scheme extension saves all the information about a model.

Syntax

```
(sde:save-model filename)
```

Returns

None

Arguments

Argument	Description
filename	Specifies the name of a file. Argument type: STRING

Description

This Scheme extension performs a complex task. It calls several other functions to save the complete model. The Scheme extension `sde:open-model` is the counterpart of this Scheme extension that loads the model and restores the original model completely. It saves a native ACIS `.sat` file to store the model geometry and Ref/Eval windows. The `.sat` file stores the curved boundaries and all the attributes of Sentaurus Structure Editor that are attached to the model.

This Scheme extension also saves a mesh command file that contains all refinement-related and doping-related information. A Scheme file also is saved containing all of the defined parameters.

These three files are sufficient to save and restore the complete state of the modeler. In addition, a TDR boundary file is saved. This file contains a tessellated model, so it cannot be used to restore the complete geometric model if curved boundaries are present.

Appendix A: Commands

sde:save-parameters

sde:save-parameters

This Scheme extension saves all defined parameters (Scheme variables) that were created using sde:define-parameter.

The saved Scheme (.scm) file can be edited and used for later sessions to restore the complete previous state of the modeler. Some information about the GUI (such as size and background color) is recorded as well.

Syntax

```
(sde:save-parameters filename)
```

Returns

BOOLEAN

Arguments

Argument	Description
filename	Specifies the name of a file in which to save all of the defined parameters. Argument type: STRING

Appendix A: Commands

sde:save-tcl-parameters

sde:save-tcl-parameters

This Scheme extension saves the defined contact colors to a Tcl file, using the contact color assignment syntax of Sentaurus Visual. The saved Tcl file can be loaded into Sentaurus Visual, after a TDR file is loaded. The contacts of the TDR file will have the same colors in Sentaurus Visual as in Sentaurus Structure Editor.

Syntax

```
(sde:save-tcl-parameters filename)
```

Returns

BOOLEAN

Arguments

Argument	Description
filename	Specifies the name of a Tcl file. Argument type: STRING

Examples

Example 1:

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Silicon" "x1")
(sdegeo:create-rectangle (position 0.3 1 0) (position 0.7 1.5 0)
    "PolySilicon" "x2")
(sdegeo:define-contact-set "red" 4 (color:rgb 1 0 0) "##")
(sdegeo:define-contact-set "green" 4 (color:rgb 0 1 0) "##")
(sdegeo:define-contact-set "blue" 4 (color:rgb 0 0 1) "##")
(sdegeo:define-contact-set "black" 4 (color:rgb 0 0 0) "##")

(sdegeo:set-contact (list (car (find-edge-id (position 0.1 1 0))))
    "red")
(sdegeo:set-contact (list (car (find-edge-id (position 0.5 1.5 0))))
    "green")
(sdegeo:set-contact (list (car (find-edge-id (position 0.9 1 0))))
    "blue")
(sdegeo:set-contact (list (car (find-edge-id (position 0.1 0 0))))
    "black")

(sdeio:save-tdr-bnd (get-body-list) "xx2d.tdr")
(sde:save-tcl-parameters "xx2d.tcl")
```

Appendix A: Commands

sde:save-tcl-parameters

The file xx2d.tcl contains the following:

```
set_region_prop {red} -color #FF0000
set_region_prop {green} -color #00FF00
set_region_prop {blue} -color #0000FF
set_region_prop {black} -color #000000
```

Example 2:

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 0.4)
    "Silicon" "x1")
(sdegeo:create-cuboid (position 0.3 0 0.4) (position 0.7 1 0.6)
    "PolySilicon" "x2")
(sdegeo:define-contact-set "red" 4 (color:rgb 1 0 0) "##")
(sdegeo:define-contact-set "green" 4 (color:rgb 0 1 0) "##")
(sdegeo:define-contact-set "blue" 4 (color:rgb 0 0 1) "##")
(sdegeo:define-contact-set "black" 4 (color:rgb 0 0 0) "##")
(sdegeo:set-contact (list (car (find-face-id (position 0.1 0.1 0.4))))
    "red")
(sdegeo:set-contact (list (car (find-face-id (position 0.5 0.1 0.6))))
    "green")
(sdegeo:set-contact (list (car (find-face-id (position 0.9 0.1 0.4))))
    "blue")
(sdegeo:set-contact (list (car (find-face-id (position 0.1 0.1 0)))))
    "black")
(sdeio:save-tdr-bnd (get-body-list) "xx3d.tdr")
(sde:save-tcl-parameters "xx3d.tcl")
```

The file xx3d.tcl is the same as xx2d.tcl.

Appendix A: Commands

sde:scale-scene

sde:scale-scene

This Scheme extension scales the model in the view window. This is only a view scale. The model coordinates do not change. The operation can be used to *magnify*, for example, thin layers.

Syntax

```
(sde:scale-scene xs ys zs)
```

Returns

None

Arguments

Argument	Description
xs	Argument type: REAL
ys	Argument type: REAL
zs	Argument type: REAL

Examples

To return to the original view, use:

```
(sde:scale-scene 1 1 1)
```

sde:scmwin-get-font-families

This Scheme extension returns the available font types that are used in the command-line window.

Syntax

```
(sde:scmwin-get-font-families)
```

Returns

STRING

sde:scmwin-get-font-family

This Scheme extension returns the font type used in the command-line window.

Syntax

(sde:scmwin-get-font-family)

Returns

STRING

sde:scmwin-get-font-size

This Scheme extension returns the font size used in the command-line window.

Syntax

(sde:scmwin-get-font-size)

Returns

INTEGER

sde:scmwin-get-font-style

This Scheme extension returns the font style used in the command-line window. It returns a pair:

- The first value is 0 (Normal), or 1 (Italic), or 2 (Oblique).
- The second value is either #t (Bold) or #f.

Syntax

(sde:scmwin-get-font-style)

Returns

PAIR

Appendix A: Commands

sde:scmwin-get-window-height

sde:scmwin-get-window-height

This Scheme extension returns the height of the command-line window in terms of number of lines (or rows).

Syntax

```
(sde:scmwin-get-window-height)
```

Returns

INTEGER

sde:scmwin-select-font

This Scheme extension displays the Select Font dialog box, which is used to select the font, the font style, and the size of the command-line window.

Syntax

```
(sde:scmwin-select-font)
```

Returns

BOOLEAN

sde:scmwin-set-font-family

This Scheme extension sets the font type to be used in the command-line window.

Syntax

```
(sde:scmwin-set-font-family font-type)
```

Returns

BOOLEAN

Arguments

Argument	Description
font-type	Specifies the font type to be used. Argument type: STRING

Appendix A: Commands

sde:scmwin-set-font-size

sde:scmwin-set-font-size

This Scheme extension sets the font size to be used in the command-line window.

Syntax

```
(sde:scmwin-set-font-size font-size)
```

Returns

BOOLEAN

Arguments

Argument	Description
font-size	Specifies the font size to be used. Argument type: INTEGER

sde:scmwin-set-prefs

This Scheme extension sets the preferences of the command-line window.

The Scheme extension `sde:scmwin-select-font` displays the Select Font dialog box, which can be used to change font styles.

Syntax

```
(sde:scmwin-set-prefs font-type font-style font-size cmdwin-height)
```

Returns

BOOLEAN

Arguments

Argument	Description
font-type	Specifies the command-line window font. Use <code>sde:scmwin-get-font-families</code> for the available fonts and <code>sde:scmwin-get-font-family</code> for the active setting. Argument type: STRING
font-style	Sets the actual font style. Argument type: STRING
font-size	Sets the font size. Argument type: INTEGER
cmdwin-height	Specifies the height of the command-line window. Argument type: INTEGER

Appendix A: Commands

sde:scmwin-set-window-height

sde:scmwin-set-window-height

This Scheme extension sets the height (in pixels) of the command-line window.

Syntax

```
(sde:scmwin-set-window-height height)
```

Returns

BOOLEAN

Arguments

Argument	Description
height	Specifies the height of the command-line window. Argument type: INTEGER

sde:scmwin-suppress-output

This Scheme extension shows or hides messages in the command-line window.

Syntax

```
(sde:scmwin-suppress-output bool)
```

Returns

None

Arguments

Argument	Description
bool	Specifies whether to show or hide messages. Argument type: BOOLEAN

Appendix A: Commands

sde:selected-entities

sde:selected-entities

This Scheme extension returns the currently active selected entities in a Scheme list. The selection tools of the GUI (Select mode) can be used to select geometric entities (body, face, edge, vertex). The selected entities are highlighted.

Syntax

(sde:selected-entities)

Returns

ENTITY LIST

sde:selected-refeval-windows

This Scheme extension returns the names of the currently selected Ref/Eval windows in a Scheme list.

The selection tools of the GUI (Select mode) can be used to select previously defined Ref/Eval windows. The selected Ref/Eval windows are highlighted.

Syntax

(sde:selected-refeval-windows)

Returns

ENTITY LIST

Appendix A: Commands

sde:separate-lumps

sde:separate-lumps

This Scheme extension separates the lumps in a body.

Syntax

```
(sde:separate-lumps [NULL | body | body-list])
```

Returns

ENTITY LIST

Arguments

Argument	Description
NULL	Specifies an empty list. Argument type: STRING
body	Specifies the name of a body. Argument type: BODY
body-list	Specifies a list of bodies. Argument type: BODY LIST

Description

Geometric operations can result in models with bodies that have multiple lumps (disjoint parts of a body). Since each body has a unique region-name attribute, these separate lumps in a body will have the same region name. As the Synopsys meshing engines require a unique region name for each disjoint region, bodies with multiple lumps must be separated before a TDR boundary file is generated.

This Scheme extension separates lumps and is also called automatically before a TDR boundary file is generated. For example, if the original region name was `region_1` and it has three lumps before `sde:separate-lumps` is called, this Scheme extension *breaks* the body into three parts and assigns a unique region name to each. A `_lump_N` string is added to each lump, where `N` is the lump counter, starting from 1. Therefore, three new bodies are generated with the region names `region_1_lump_1`, `region_1_lump_2`, and `region_1_lump_3`.

If the argument list is empty, then all geometric bodies are separated. Alternatively, either a single body or a body list can be specified in the argument list, in which case, only the specified entities are separated.

This Scheme extension returns the newly created body list (which contains the entity IDs of the separated lumps).

Appendix A: Commands

sde:separate-lumps

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 10 3 0) "Silver"
 "region_1")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position 4 -1 0) (position 6 4 0) "Silicon"
 "region_2")
; region_1 will have two lumps !
(get-body-list)
; returns the entity ids of the two bodies
(sde:separate-lumps)
(get-body-list)
; returns the entity ids of the three bodies
; region_1 was separated into two bodies, with region names
; region_1_lump_1 and region_1_lump_2
```

Appendix A: Commands

sde:set-background-color

sde:set-background-color

This Scheme extension sets the background color of the main window of the GUI.

The RGB colors for the top and bottom must be specified (0–255). If the top and bottom colors differ, then a graded background is used. If the top and bottom colors match, then the background is uniform.

Syntax

```
(sde:set-background-color rtop gtop btop rbottom gbottom bbottom)
```

Returns

None

Arguments

Argument	Description
rtop	Specifies the red value for the top of the background color. Argument type: INTEGER
gtop	Specifies the green value for the top of the background color. Argument type: INTEGER
btop	Specifies the blue value for the top of the background color. Argument type: INTEGER
rbottom	Specifies the red value for the bottom of the background color. Argument type: INTEGER
gbottom	Specifies the green value for the bottom of the background color. Argument type: INTEGER
bbottom	Specifies the blue value for the bottom of the background color. Argument type: INTEGER

Appendix A: Commands

sde:set-backwards-compatibility

sde:set-backwards-compatibility

This Scheme extension sets the value of the overall backward compatibility or the given backward-compatibility parameter. See [sde:get-backwards-compatibility on page 461](#).

This Scheme extension, if called with one parameter, sets the general backward compatibility to the given release version. If one or more pairs of parameters are given, then the corresponding backward-compatibility parameters are set to the corresponding release values.

Syntax

```
(sde:set-backwards-compatibility { backcompat-release |  
    backcompat-param1 backcompat-release1  
    [backcompat-param2 backcompat-release2 ...] })
```

Returns

Returns #t upon successful setting of overall backward compatibility or backward-compatibility parameters

Arguments

Argument	Description
backcompat-release	Specifies the release version. For example: "S-2021.06" Argument type: STRING
backcompat-param	Specifies backward-compatibility parameters, which are specific, internally defined, parameters that are used to reproduce certain defined functionality from previous releases of Sentaurus Structure Editor. The available backward-compatibility parameters are: <ul style="list-style-type: none">"Save Part at End" Defaults to #t, such that when saving the model, the .sat file is saved after changes to the structure are made, which are needed to save the .cmd file and the .scm file. This parameter is #f for compatibility with releases E-2010.12 and earlier, where the .sat file is saved first, before the model is modified for saving the .cmd file and the .scm file."Z Align Polygons" Defaults to #f. This parameter is #t in Version E-2010.12 and earlier, leading to the automatic reversal of polygon direction for coplanar polygons whose face normal is not parallel to the z-axis in the active coordinate system. Argument type: STRING

Appendix A: Commands

sde:set-default-material

sde:set-default-material

This Scheme extension sets the active DATEX material to the specified value.

This Scheme extension also changes the selected DATEX material in the DATEX list.

Syntax

```
(sde:set-default-material datex-material)
```

Returns

None

Arguments

Argument	Description
datex-material	Specifies one of the DATEX colors, specified in the datexcodes.txt file. Argument type: DATEXMAT

sde:set-menubar-font-size

This Scheme extension changes the font size of the menu bar of the GUI.

You can also use the menu command **View > GUI Font Size** to change the font size.

Syntax

```
(sde:set-menubar-font-size font-size)
```

Returns

None

Arguments

Argument	Description
font-size	Specifies the font size. Suggested values [10-14]. Argument type: INTEGER

sde:set-process-up-direction

This Scheme extension specifies the up-direction convention of the coordinate system to be used during process emulation.

If you use the Sentaurus Process–Sentaurus Structure Editor interface to perform process emulation, then the Scheme script generated by Sentaurus Process will contain instructions to select the correct coordinate system convention automatically.

Syntax

```
(sde:set-process-up-direction up-direction)
```

Returns

BOOLEAN

Arguments

Argument	Description
up-direction	Specifies the up direction. Options are: <ul style="list-style-type: none">• If you specify "-x", then the up direction changes to -x (default).• If you specify "+z", then the up direction is set to +z. Argument type: STRING

Examples

```
(sde:set-process-up-direction "+z")
```

Appendix A: Commands

sde:set-project-name

sde:set-project-name

This Scheme extension sets the project name, which is used in the Build Mesh dialog box as the name of the saved mesh.

Syntax

```
(sde:set-project-name project-name)
```

Returns

STRING

Arguments

Argument	Description
project-name	Specifies the name of a project. Argument type: STRING

sde:set-rendering-mode

This Scheme extension sets the rendering mode.

Syntax

```
(sde:set-rendering-mode rendering-mode)
```

Returns

Previous setting (STRING)

Arguments

Argument	Description
rendering-mode	Specifies the rendering mode. Options are facet, flat, gouraud (or alternatively, smooth), hidden, or wire. Argument type: STRING

Appendix A: Commands

sde:set-selection-level

sde:set-selection-level

This Scheme extension sets the selection filter to the specified value. All subsequent selection operations will use the specified filter to select only those entities that correspond to the specified type.

Syntax

```
(sde:set-selection-level filter-type)
```

Returns

None

Arguments

Argument	DescriptionArgument Type
filter-type	Specifies the filter type. Options are "body" "face" "edge" "vertex" "other". Argument type: STRING

Examples

```
(sde:set-selection-level "edge")
```

Appendix A: Commands

sde:set-translucency

sde:set-translucency

This Scheme extension sets the translucency of the main window of the GUI.

Syntax

```
(sde:set-translucency body(list) tflag)
```

Returns

None

Arguments

Argument	Description
body(list)	Specifies an entity or a list of entities. Argument type: ENTITY ENTITY LIST
tflag	Specifies whether the main window is translucent: <ul style="list-style-type: none">• true means to make the main window translucent.• false means to make the main window opaque. Argument type: BOOLEAN

Appendix A: Commands

sde:set-view-mode

sde:set-view-mode

This Scheme extension sets the view to the specified value, and it also performs a zoom extent operation.

Syntax

```
(sde:set-view-mode view-mode)
```

Returns

None

Arguments

Argument	Description
view-mode	Specifies the view mode from one of the following: <ul style="list-style-type: none">ISO specifies the isometric view.XY sets the view to the xy plane.XZ sets the view to the xz plane.YZ sets the view to the yz plane.X-Y sets the view window in such a way that the x-axis points horizontally and the y-axis points downwards. Argument type: STRING

sde:set-view-operator

This Scheme extension manipulates the toolbar, by setting the active actions to the specified mode.

Syntax

```
(sde:set-view-operator op-mode)
```

Returns

Previous setting (STRING)

Arguments

Argument	Description
op-mode	Specifies the operation mode. Options are orbit, pan, select, or zoom. Argument type: STRING

Appendix A: Commands

sde:set-view-params

sde:set-view-params

This Scheme extension sets the GUI view (rotation, zoom, and so on) to a previously recorded state.

The order of parameters is not significant, but `eye` must precede `target` and, if given, `width` must precede `height`. Existing settings are extracted from the current camera position if the optional values are not given.

Syntax

```
(sde:set-view-params view-params)
```

Returns

None

Arguments

Argument	Description
view-params	Specifies a list containing the same parameters that are returned by <code>sde:get-view-params</code> , but with some less rigid constraints. The parameters are: <ul style="list-style-type: none">• <code>eye</code>, Argument type: POSITION• <code>target</code>, Argument type: POSITION• <code>up-vector</code> (optional), Argument type: GVECTOR• <code>width</code> (optional), Argument type: REAL• <code>height</code> (optional), Argument type: REAL• <code>projection</code> (optional), Argument type: STRING

Examples

```
(define myview (sde:get-view-params))  
(sde:set-view-params myview)
```

Appendix A: Commands

sde:set-window-position

sde:set-window-position

This Scheme extension positions the main window of the GUI.

Syntax

```
(sde:set-window-position xpos ypos)
```

Returns

None

Arguments

Argument	Description
xpos	Specifies the horizontal placement of the GUI. Argument type: INTEGER
ypos	Specifies the vertical placement of the GUI. Argument type: INTEGER

sde:set-window-size

This Scheme extension resizes the main window of the GUI.

Syntax

```
(sde:set-window-size xsize ysize)
```

Returns

None

Arguments

Argument	Description
xsize	Specifies the horizontal size (in pixels) of the GUI. Argument type: INTEGER
ysize	Specifies the vertical size (in pixels) of the GUI. Argument type: INTEGER

Appendix A: Commands

sde:set-window-style

sde:set-window-style

This Scheme extension sets the GUI style.

Syntax

```
(sde:set-window-style wstyle)
```

Returns

None

Arguments

Argument	Description
wstyle	Sets the GUI style. The available styles are Windows, Motif, Cleanlooks, Plastique, and CDE. Argument type: STRING

Appendix A: Commands

sde:setrefprops

sde:setrefprops

This Scheme extension controls the refinement properties of the internal faceter of Sentaurus Structure Editor, which is an important link between Sentaurus Structure Editor and other Synopsys tools. When Sentaurus Structure Editor contains a model with curved boundaries, this internal curved representation must be tessellated before a TDR boundary file is generated.

Note:

This Scheme extension is deprecated. Instead, use the aspect-ratio, max-edge-length, normal-tolerance, and surface-tolerance arguments of the sdeio:save-tdr-bnd Scheme extension to control the properties of the faceter (see [sdeio:save-tdr-bnd on page 745](#)).

Syntax

```
(sde:setrefprops surface-tolerance normal-tolerance  
[aspect-ratio max-edge-length])
```

Returns

None

Arguments

Argument	Description
surface-tolerance	Specifies the surface tolerance. It is an absolute number. It is the allowable maximum distance between the original model and tessellated output. By specifying a smaller value, the tessellated model deviates less from the original model. Argument type: REAL
normal-tolerance	Specifies the normal tolerance. It is the allowable maximum difference between the surface normals of the original and tessellated models. Both values are global and apply to the complete model. In the case of a thin curved channel, the refinement based on surface tolerance is not recommended. It is better to control the normal tolerance. Argument type: REAL
aspect-ratio	Optional. Controls the maximum aspect ratio of the triangles in the tessellated boundary. Using an aspect ratio control can prevent the creation of sliver faces (triangles with a close to zero angle). The recommended aspect ratio value is [2-4]. Argument type: REAL
max-edge-length	Optional. Controls the maximum-allowable edge length in the tessellated boundary. In some cases, a more regular tessellation can be created using max-edge-length, which helps meshing. A small max-edge-length value might cause unnecessary overrefinement. Argument type: REAL

Appendix A: Commands

sde:setup-grid

sde:setup-grid

This Scheme extension sets the basic parameters for the grid.

Syntax

```
(sde:setup-grid x-spacing y-spacing edge-weight edge-pattern)
```

Returns

None

Arguments

Argument	Description
x-spacing	Specifies the horizontal grid size. Argument type: REAL
y-spacing	Specifies the vertical grid size. Argument type: REAL
edge-weight	Specifies the line thickness. Argument type: INTEGER
edge-pattern	Specifies the line pattern that is used to draw the grid. Possible values are: <ul style="list-style-type: none">• A solid line: "---• A simple dashed line: "- -"• A dotted line: "..."• Dash and dot alternating: "-.-."• Dash and two dots alternating: "-...-.."• Dash and three dots alternating: "-...-...-.."• Long dashes: "---- -----"• Very long dash and short dash alternating: "center"• Very long dash and two short dashes alternating: "phantom" Argument type: STRING

Examples

```
(sde:setup-grid 1 1 2 "...")  
(sde:show-grid #t)
```

Appendix A: Commands

sde:show

sde:show

This Scheme extension is used to show the specified entity or entities in the view window. The bodies are added to the view window, and the view window is refreshed. If the entity is already visible, then there is no change in the view window.

Syntax

```
(sde:show entity | entity-list)
```

Returns

None

Arguments

Argument	Description
entity	Specifies an entity. Argument type: ENTITY
entity-list	Specifies a list of entities. Argument type: ENTITY LIST

Examples

```
(sde:clear)
(define r1 (sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
                                      "Photoresist" "r_1"))
(define r2 (sdegeo:create-rectangle (position 2 0 0) (position 3 1 0)
                                      "Silicon" "r_2"))
(define r3 (sdegeo:create-rectangle (position 4 0 0) (position 5 1 0)
                                      "Resist" "r_3"))
(define r4 (sdegeo:create-rectangle (position 6 0 0) (position 7 1 0)
                                      "PolySilicon" "r_4"))
(sde:xshow-material "")
(sde:show r1)
(sde:show (list r3 r4))
```

Appendix A: Commands

sde:showattribs

sde:showattribs

This Scheme extension shows the attached attributes for the specified entity list.

This Scheme extension can display the ACIS part number, and the material and region attributes of Sentaurus Structure Editor that are attached to the bodies.

Syntax

```
(sde:showattribs entity-list | "all")
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies a list of entities. Alternatively, you can specify the keyword "all", in which case, all ACIS bodies are shown. Argument type: ENTITY LIST

sde:show-bg-image

This Scheme extension displays the previously loaded image.

Syntax

```
(sde:show-bg-image)
```

Returns

None

Appendix A: Commands

sde:show-contact

sde:show-contact

This Scheme extension adds to the view all regions that have contacts defined with the specified contact attributes. Contacts are identified with the attribute name `contact`. In two dimensions, contacts are edges and, in three dimensions, contacts are faces.

Syntax

```
(sde:show-contact contact-name | contact-name-list)
```

Returns

None

Arguments

Argument	Description
contact-name	Specifies the name of a contact. Argument type: STRING
contact-name-list	Specifies a list of contact names. Argument type: STRING LIST

sde:show-grid

This Scheme extension switches on and off visualization of the grid in the view window.

Syntax

```
(sde:show-grid on-off)
```

Returns

BOOLEAN

Arguments

Argument	Description
on-off	Specifies whether to switch on or off visualization of the grid. Argument type: BOOLEAN

Appendix A: Commands

sde:show-interface

sde:show-interface

This Scheme extension adds to the view window all regions that have interfaces defined with the specified interface attributes. Interfaces are identified with the attribute name `interface`.

In two dimensions, interfaces are edges between neighboring regions. In three dimensions, interfaces are faces between neighboring regions.

Syntax

```
(sde:show-interface interface-name | interface-name-list)
```

Returns

None

Arguments

Argument	Description
interface-name	Specifies the name of an interface. Argument type: STRING
interface-name-list	Specifies a list of interface names. Argument type: STRING LIST

sde:show-mask

This Scheme extension adds the specified masks to the view window. Masks are planar wire bodies that have the specified mask names as *mask name* attributes.

Syntax

```
(sde:show-mask mask-name | mask-name-list)
```

Returns

None

Arguments

Argument	Description
mask-name	Specifies the name of a mask. Argument type: STRING
mask-name-list	Specifies a list of mask names. Argument type: STRING LIST

Appendix A: Commands

sde:show-material

sde:show-material

This Scheme extension is used to show the regions that have the specified material names as material attributes. The regions are added to the view window, and the view window is refreshed. If a region is already visible, there is no change in the view window.

Syntax

```
(sde:show-material datex-material | datex-material-list)
```

Returns

None

Arguments

Argument	Description
datex-material	Specifies a DATEX material. Argument type: DATEXMAT
datex-material-list	Specifies a list of DATEX materials. Argument type: DATEXMAT LIST

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Photoresist" "region_1")
(sdegeo:create-rectangle (position 2 0 0) (position 3 1 0) "Silicon"
    "region_2")
(sdegeo:create-rectangle (position 4 0 0) (position 5 1 0) "Resist"
    "region_3")
(sdegeo:create-rectangle (position 6 0 0) (position 7 1 0)
    "PolySilicon" "region_4")
(sde:xshow-material "")
(sde:show-material "Silicon")
```

Appendix A: Commands

sde:show-pcurves

sde:show-pcurves

This Scheme extension visualizes the parametric curves for each entity, defined in the database.

Note:

Only the wire frame view is affected by this Scheme extension, by rendering the parameter lines.

Syntax

```
(sde:show-pcurves rendering-flag [u-params [v-params]])
```

Returns

None

Arguments

Argument	Description
rendering-flag	Specifies whether to switch on or off the parameter line visualization. Argument type: BOOLEAN
u-params	Optional. If this argument is not specified, then the default is 5. Argument type: INTEGER
v-params	Optional. If this argument is not specified, then the default is 5. Argument type: INTEGER

Appendix A: Commands

sde:show-region

sde:show-region

This Scheme extension shows the specified regions in the view window. The bodies that have the specified region name as a region attribute are added to the view window, and the view window is refreshed. If the region is already visible, there is no change in the view window.

Syntax

```
(sde:show-region region-name | region-name-list)
```

Returns

None

Arguments

Argument	Description
region-name	Specifies the name of a region. It must be an existing region name. Argument type: STRING
region-name-list	Specifies a list of region names. They must be existing region names. Argument type: STRING LIST

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Photoresist" "region_1")
(sdegeo:create-rectangle (position 2 0 0) (position 3 1 0) "Silicon"
    "region_2")
(sdegeo:create-rectangle (position 4 0 0) (position 5 1 0) "Resist"
    "region_3")
(sdegeo:create-rectangle (position 6 0 0) (position 7 1 0)
    "PolySilicon" "region_4")
(sde:xshow-material "")
(sde:show-region "region_1")
(sde:show-region (list "region_3" "region_4"))
```

Appendix A: Commands

sde:split-solid

sde:split-solid

This Scheme extension splits the specified solid (2D and 3D) into separate bodies.

The new split bodies can be treated as high-level entities. They can be deleted, new material and region properties can be assigned to them, and any geometry operations can be applied to them that can be applied to high-level entities only.

Note:

After a solid entity is split into parts, region names are no longer unique. All split parts will have the same region name, therefore, renaming of these parts might be necessary.

Syntax

```
(sde:split-solid solid-body base-position plane-normal)
```

Returns

BODY LIST

Arguments

Argument	Description
solid-body	Specifies a body to be split. Argument type: BODY
base-position	Defines the base position (that is, a position variable) for the cutplane. Argument type: POSITION
plane-normal	Defines the normal vector (that is, a gvector variable) for the cutplane. Argument type: GVECTOR

Examples

```
(sde:clear)
(define my_cube (sdegeo:create-cuboid (position -1 -1 -1)
                                         (position 1 1 1) "Silver" "xx"))
(sde:split-solid my_cube (position 0 0 0) (gvector 1 1 0))
(sde:split-solid my_cube (position 0 0 0) (gvector 1 -1 0))
```

Appendix A: Commands

sde:stripextension

sde:stripextension

This Scheme extension locates the last dot (.) in a specified string and removes all characters that follow. The stripped string is returned. If the input string does not have a dot, then the input string is returned.

Syntax

(sde:stripextension name)

Returns

STRING

Arguments

Argument	Description
name	Specifies an input string. Argument type: STRING

sde:substring

This Scheme extension returns a substring of the input string. If the length of the input string is smaller than `max`, then an empty string is returned. (The `string-length` command returns the length of a string.)

Syntax

(sde:substring string min max)

Returns

STRING

Arguments

Argument	Description
string	Specifies an input string. Argument type: STRING
min	Specifies a number that denotes the first character position (starting from zero). Argument type: INTEGER
max	Specifies a number that denotes the last character position. Argument type: INTEGER

Appendix A: Commands

sde:test-entity

sde:test-entity

This Scheme extension returns the entity types.

Syntax

```
(sde:test-entity entities)
```

Returns

Possible return values are:

"unknown"	
"solid"	Solid with a positive volume (manifold)
"non-manifold"	Solid with zero volume (non-manifold)
"mask"	Mask (same conditions as for "non-manifold", but also the <code>maskname</code> attribute is defined)

Arguments

Argument	Description
entities	Specifies an entity or a list of entities. Argument type: ENTITY ENTITY LIST

sde:toggle-lights

This Scheme extension switches on or off the lights of the view window, that is, this Scheme extension behaves like a light switch.

Syntax

```
(sde:toggle-lights)
```

Returns

None

Appendix A: Commands

sde:top-coord

sde:top-coord

This Scheme extension returns the top coordinate of the specified body or body list. The top coordinate depends on the coordinate system used:

- For the UCS, the top coordinate is the minimum x-coordinate.
- For the DF–ISE coordinate system, the top coordinate is the maximum z-coordinate.

Syntax

```
(sde:top-coord body | body-list)
```

Returns

REAL

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY
body-list	Specifies a list of bodies. Argument type: BODY LIST

sde:tr-get

This Scheme extension returns the transformation that was used to place the 3D cut on the xy plane, using sdegeo:3d-cut.

The transformation is composed of a rotation component (3×3 matrix) and a translation component (3D vector). The extension returns 12 real numbers in a list. The first nine numbers represent the 3×3 rotation matrix (A) and the last three numbers (T) represent the xyz part of the translation component. The new position of any point in the cut can be calculated by $P_{\text{cut}} = P_{\text{original}} \times A + T$.

Syntax

```
(sde:tr-get)
```

Returns

LIST

Appendix A: Commands

sde:view-filter-reset

sde:view-filter-reset

This Scheme extension resets the view window to the default value. All regions and contacts are shown, and interfaces are hidden.

Syntax

```
(sde:view-filter-reset)
```

Returns

None

sde:view-set-light-intensity

This Scheme extension sets the light intensity of the view window.

Syntax

```
(sde:view-set-light-intensity lintensity)
```

Returns

None

Arguments

Argument	Description
lintensity	Sets the light intensity to a value between 0 and 1. Argument type: INTEGER

Appendix A: Commands

sde:view-set-visible-area

sde:view-set-visible-area

This Scheme extension resizes the GUI drawing area explicitly. If the bounding box of the created device is known, then it might be convenient to preset the view area to that size.

Syntax

```
(sde:view-set-visible-area xmin xmax ymin ymax)
```

Returns

None

Arguments

Argument	Description
xmin	Argument type: REAL
xmax	Argument type: REAL
ymin	Argument type: REAL
ymax	Argument type: REAL

sde:wait-cursor

This Scheme extension changes the cursor to an hourglass pointer.

Syntax

```
(sde:wait-cursor)
```

Returns

None

Appendix A: Commands

sde:window-select-2d

sde:window-select-2d

This Scheme extension selects all entities (of the specified type, `stype`) lying in the specified 2D window. It returns a list containing all vertex, edge, or body entities that are inside the specified 2D window. A similar Scheme extension `sde:window-select-3d` applies to 3D models.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sde:window-select-2d x-left y-left x-right y-right elist stype)
```

Returns

LIST

Arguments

Argument	Description
x-left	Defines the rectangular window to be used during the selection. Argument type: REAL
y-left	Defines the rectangular window to be used during the selection. Argument type: REAL
x-right	Defines the rectangular window to be used during the selection. Argument type: REAL
y-right	Defines the rectangular window to be used during the selection. Argument type: REAL
elist	Specifies a list of bodies. Alternatively, if you specify "all", then all entities in the model are used for the selection. Argument type: BODY LIST
stype	Specifies the selection criterion, that is, the entity type. Possible values are "vertex" "edge" "body". If "vertex" is specified, then a vertex list is returned. If "edge" is specified, then an edge list is returned. If "body", is specified, then a body list is returned. If no entity is found, then an empty list is returned. Argument type: STRING

Appendix A: Commands

sde:window-select-3d

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "AB")
(sdegeo:create-rectangle (position 0 0 0) (position 2 2 0) "Silicon"
    "region_1")
(sdegeo:create-circle (position 1 2 0) 1 "Silicon" "region_2")
(define vlist (sde:window-select-2d -1 -1 3 4 "all" "vertex"))
(define elist (sde:window-select-2d -1 -1 3 4 "all" "edge"))
(define blist (sde:window-select-2d -1 -1 3 4 "all" "body"))
```

sde:window-select-3d

This Scheme extension selects all entities (of the specified type, *stype*) lying in the specified 3D window. It returns a list containing all vertex, edge, or face entities that are inside the specified 3D window. A similar Scheme extension sde:window-select-2d applies to 2D models.

Note:

This Scheme extension applies to 3D models only.

Syntax

```
(sde:window-select-3d pmin pmax elist stype)
```

Returns

LIST

Arguments

Argument	Description
pmin	Defines the 3D window (cuboid) to be used during the selection. Argument type: POSITION
pmax	Defines the 3D window (cuboid) to be used during the selection. Argument type: POSITION
elist	Specifies a list of bodies. Alternatively, if you specify "all", then all entities in the model are used for the selection. Argument type: BODY LIST

Appendix A: Commands

sde:xshow

Argument	Description
stype	Specifies the selection criterion, that is, the entity type. Possible values are "vertex" "edge" "face" "body". If "vertex" is specified, then a vertex list is returned. If "edge" is specified, then an edge list is returned. If "face" is specified, then a face list is returned. If "body", is specified, then a body list is returned. If no entity is found, then an empty list is returned. Argument type: STRING

sde:xshow

This Scheme extension shows the specified entities exclusively in the view window. The entities are added to the view window. All other entities are removed from the view window, and the view window is refreshed.

Syntax

```
(sde:xshow entity | entity-list)
```

Returns

None

Arguments

Argument	Description
entity	Specifies an entity. Argument type: ENTITY
entity-list	Specifies a list of entities. Argument type: ENTITY LIST

Examples

```
(sde:clear)
(define r1 (sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
                                      "Photoresist" "r_1"))
(define r2 (sdegeo:create-rectangle (position 2 0 0) (position 3 1 0)
                                      "Silicon" "r_2"))
(define r3 (sdegeo:create-rectangle (position 4 0 0) (position 5 1 0)
                                      "Resist" "r_3"))
(define r4 (sdegeo:create-rectangle (position 6 0 0) (position 7 1 0)
                                      "PolySilicon" "r_4"))
(sde:xshow r1)
(sde:xshow (list r3 r4))
```

Appendix A: Commands

sde:xshow-contact

sde:xshow-contact

This Scheme extension exclusively adds all regions to the view that have contacts defined with the specified contact attributes and removes all other regions. Contacts are identified with the attribute name `contact`. In two dimensions, contacts are edges and, in three dimensions, contacts are faces.

Syntax

```
(sde:xshow-contact contact-name | contact-name-list)
```

Returns

None

Arguments

Argument	Description
contact-name	Specifies the name of a contact. Argument type: STRING
contact-name-list	Specifies a list of contact names. Argument type: STRING LIST

Appendix A: Commands

sde:xshow-interface

sde:xshow-interface

This Scheme extension exclusively adds all regions to the view window, which have interfaces defined, with the specified interface attributes, and it removes all other interfaces.

Interfaces are identified with the attribute name `interface`. In two dimensions, interfaces are edges between neighboring regions and, in three dimensions, interfaces are faces between neighboring regions.

Syntax

```
(sde:xshow-interface interface-name | interface-name-list)
```

Returns

None

Arguments

Argument	Description
interface-name	Specifies the name of an interface. Argument type: STRING
interface-name-list	Specifies a list of interface names. Argument type: STRING LIST

Appendix A: Commands

sde:xshow-mask

sde:xshow-mask

This Scheme extension exclusively adds the specified masks to the view window, that is, the specified masks are added and all other masks are removed. Masks are planar wire bodies that have the specified mask names as *mask name* attributes.

Syntax

```
(sde:xshow-mask mask-name | mask-name-list)
```

Returns

None

Arguments

Argument	Description
mask-name	Specifies the name of a mask. Argument type: STRING
mask-name-list	Specifies a list of mask names. Argument type: STRING LIST

Appendix A: Commands

sde:xshow-material

sde:xshow-material

This Scheme extension shows exclusively the regions that have the specified material names as material attributes in the view window. The region is added to the view window, all other regions are removed from the view window, and the view window is refreshed.

Syntax

```
(sde:xshow-material datex-material | datex-material-list)
```

Returns

None

Arguments

Argument	Description
datex-material	Specifies the name of a DATEX material. Argument type: DATEXMAT
datex-material-list	Specifies a list of DATEX materials. Argument type: DATEXMAT LIST

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Photoresist" "region_1")
(sdegeo:create-rectangle (position 2 0 0) (position 3 1 0) "Silicon"
    "region_2")
(sdegeo:create-rectangle (position 4 0 0) (position 5 1 0) "Resist"
    "region_3")
(sdegeo:create-rectangle (position 6 0 0) (position 7 1 0)
    "PolySilicon" "region_4")
(sde:xshow-material "Silicon")
```

Appendix A: Commands

sde:xshow-region

sde:xshow-region

This Scheme extension shows the specified regions exclusively in the view window. The bodies, which have the specified region names as region attributes, are added to the view window. All other bodies are removed from the view window, and the view window is refreshed.

Syntax

```
(sde:xshow-region region-name | region-name-list)
```

Returns

None

Arguments

Argument	Description
region-name	Specifies the name of a region. The region name must exist. Argument type: STRING
region-name-list	Specifies a list of region names. The region names must exist. Argument type: STRING LIST

Examples

```
(sde:clear)
(sdgeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Photoresist" "region_1")
(sdgeo:create-rectangle (position 2 0 0) (position 3 1 0) "Silicon"
    "region_2")
(sdgeo:create-rectangle (position 4 0 0) (position 5 1 0) "Resist"
    "region_3")
(sdgeo:create-rectangle (position 6 0 0) (position 7 1 0)
    "PolySilicon" "region_4")
(sde:xshow-region "region_1")
(sde:xshow-region (list "region_3" "region_4"))
```

Appendix A: Commands

sde:zoom-all

sde:zoom-all

This Scheme extension fits the model to the view window.

Syntax

(sde:zoom-all)

Returns

None

sdedr:append-cmd-file

This Scheme extension appends a text file to the end of a mesh command file. The mesh command file is not created by this Scheme extension.

You must call `sdedr:write-cmd-file` after `sdedr:append-cmd-file` to create the mesh command file.

Syntax

(sdedr:append-cmd-file filename)

Returns

None

Arguments

Argument	Description
filename	Specifies the name of a text file to be appended. Argument type: STRING

Appendix A: Commands

sdedr:clear

sdedr:clear

This Scheme extension clears all doping-related and refinement-related data from Sentaurus Structure Editor. A subsequent `sdedr:write-cmd-file` call creates an empty command file. (Choose **Mesh > Clear All > Profiles and Definitions.**)

Syntax

`(sdedr:clear)`

Returns

None

sdedr:clear-multibox-definitions

This Scheme extension deletes all multibox definition-related data from the doping and refinement container. (Choose **Mesh > Clear All > Multibox Definitions.**)

Syntax

`(sdedr:clear-multibox-definitions)`

Returns

None

sdedr:clear-multibox-placements

This Scheme extension deletes all multibox placement-related data from the doping and refinement container. (Choose **Mesh > Clear All > Multibox Placements.**)

Syntax

`(sdedr:clear-multibox-placements)`

Returns

None

sdedr:clear-profile-definitions

This Scheme extension deletes all profile definition-related data from the doping and refinement container. (Choose **Mesh > Clear All > Profile Definitions.**)

Syntax

```
(sdedr:clear-profile-definitions)
```

Returns

None

sdedr:clear-profile-placements

This Scheme extension deletes all profile placement-related data from the doping and refinement container. (Choose **Mesh > Clear All > Profile Placements.**)

Syntax

```
(sdedr:clear-profile-placements)
```

Returns

None

sdedr:clear-ref-windows

This Scheme extension deletes all previously defined Ref/Eval windows from the database. (Choose **Mesh > Clear All > Ref/Eval Windows.**)

Syntax

```
(sdedr:clear-ref-windows)
```

Returns

None

sdedr:clear-refinement-definitions

This Scheme extension deletes all refinement definition–related data from the doping and refinement container. The mesh command files contain all refinement-related data. The command file includes two data sections that are related to doping and refinement: Definitions and Placements.

The Definitions section contains the definitions of the refinement entities (definition name, refinement size information), and the Placements section contains the named placements for refinements (associated refinement windows). This Scheme extension deletes all refinement-related data *only* from the Definitions section. (Choose **Mesh > Clear All > Refinement Definitions**.)

Syntax

```
(sdedr:clear-refinement-definitions)
```

Returns

None

sdedr:clear-refinement-placements

This Scheme extension deletes all refinement placement–related data from the doping and refinement container. The mesh command files contain all refinement-related data. The command file includes two data sections that are related to doping and refinement: Definitions and Placements.

The Definitions section contains the definitions of the refinement entities (definition name, refinement size information), and the Placements section contains the named placements for refinements (associated refinement windows). This Scheme extension deletes all refinement-related data *only* from the Placements section. (Choose **Mesh > Clear All > Refinement Placements**.)

Syntax

```
(sdedr:clear-refinement-placements)
```

Returns

None

Appendix A: Commands

sdedr:clear-submesh-placement-transform

sdedr:clear-submesh-placement-transform

This Scheme extension clears all transform operations (Reflect, Rotation, and ShiftVector) as defined for the named external profile placement through the Scheme extension sdedr:transform-submesh-placement. See [sdedr:transform-submesh-placement on page 582](#).

Syntax

```
(sdedr:clear-submesh-placement-transform placement-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Specifies a placement name. Argument type: STRING

sdedr:convert-mask-to-drs-body

This Scheme extension converts the specified mask into a Ref/Eval window.

If the mask is not found, then this Scheme extension returns #f. If the conversion is successful, then it returns #t.

Syntax

```
(sdedr:convert-mask-to-drs-body maskname refwindowname)
```

Returns

BOOLEAN

Arguments

Argument	Description
maskname	Specifies the name of a mask. Argument type: STRING
refwindowname	Specifies the name of a Ref/Eval window. Argument type: STRING

Appendix A: Commands

sdedr:define-1d-external-profile

sdedr:define-1d-external-profile

This Scheme extension creates a 1D external profile in the mesh command file.

The default unit is micrometer. The lateral profile can be given as a Gaussian function or an error function. The "Range" keyword with its parameters range-from and range-to is optional.

See *Sentaurus™ Mesh User Guide*, Defining Analytic Profiles, for detailed descriptions of parameters.

Syntax

```
(sdedr:define-1d-external-profile name filename "Scale" scale  
    "DataScale" data-scale ["Range" range-from range-to]  
    [lateral-function])
```

Returns

None

Arguments

Argument	Description
name	Argument type: STRING
filename	Argument type: STRING
scale	Argument type: REAL
data-scale	Argument type: REAL
range-from	Optional. Argument type: REAL
range-to	Optional. Argument type: REAL
lateral-function	Optional. Specifies the lateral function. Possible values are "Erf", "Eval", or "Gauss". For example: "Erf" {"Factor" lateral-factor "Length" length} or: "Eval" {eval-init eval-function} or: "Gauss" {"Factor" lateral-factor "Length" length "StdDev" standard-deviation} Argument type: STRING

Appendix A: Commands

sdedr:define-1d-external-profile

Argument	Description
lateral-factor	Argument type: REAL
length	Argument type: REAL
standard-deviation	Argument type: REAL
eval-init	Argument type: STRING
eval-function	Argument type: STRING

Appendix A: Commands

sdedr:define-analytical-profile

sdedr:define-analytical-profile

This Scheme extension creates a user-defined analytic function that describes doping in the mesh command file.

The optional definition of lateral diffusion is not available if the definition in the primary direction is the analytic type "General". The "General" type includes both the primary and lateral distributions in its definition and, therefore, it is not intended to have a separate lateral diffusion.

See *Sentaurus™ Mesh User Guide*, Defining Analytic Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-analytical-profile name species initialization function
    start-value analytic-type
    [lateral-type-gauss-erf lateral-parameter lateral-value |
     "Eval" lateral-eval-init lateral-eval-func])
```

Returns

None

Arguments

Argument	Description
name	Argument type: STRING
species	Argument type: STRING
initialization	Argument type: STRING
function	Argument type: STRING
start-value	Argument type: REAL
analytic-type	Specifies the analytic type. Options are "General" "Eval". Argument type: STRING
lateral-type-gauss-erf	Optional. Specifies the lateral function. Options are "Gauss" "Erf". Argument type: STRING
lateral-parameter	Optional. Specifies the lateral parameter. Options are "factor" "length" "stddev". Argument type: STRING
lateral-value	Optional. Specifies the lateral value. Argument type: REAL

Appendix A: Commands

sdedr:define-analytical-profile-placement

Argument	Description
lateral-eval-init	Optional. Argument type: STRING
lateral-eval-func	Optional. Argument type: STRING

sdedr:define-analytical-profile-placement

This Scheme extension creates a placement for an analytic profile definition in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Placing Analytic Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-analytical-profile-placement placement-name
    definition-name ref-eval-window symmetry replacement evaluation
    eval-window decay-length [ "Gauss" ] eval-window-type)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
ref-eval-window	Argument type: STRING
symmetry	Specifies the symmetry. Options are "Both" "Positive" "Negative". Argument type: STRING
replacement	Specifies the type of replacement. Options are "Replace" "NoReplace" "LocalReplace". Argument type: STRING
evaluation	Specifies the type of evaluation. Options are "Eval" "NoEval". Specifying "NoEval" causes Sentaurus Structure Editor to write the "notevalline" keyword to the AnalyticalProfile statement of the Placements section of the Sentaurus Mesh command file. The AnalyticalProfile statement instructs Sentaurus Mesh <i>not</i> to include the base line or base polygons in the evaluation of the analytic doping profiles. Argument type: STRING

Appendix A: Commands

sdedr:define-analytical-profile-placement

Argument	Description
eval-window	Argument type: STRING
decay-length	This argument corresponds to the mesh command file parameter "DecayLength" unless the optional string "Gauss" is also present, in which case, decay-length is interpreted instead as corresponding to the "GaussDecayLength" parameter of the mesh command file. Argument type: REAL
"Gauss"	Optional. Argument type: STRING
eval-window-type	Specifies the type of Ref/Eval window. Options are "region" "material" "evalwin". Argument type: STRING

Appendix A: Commands

sdedr:define-body-interface-refwin

sdedr:define-body-interface-refwin

This Scheme extension defines a refinement window between the interfaces of the specified geometric bodies.

If the refinement window is already defined, then the Scheme extension returns #f. If there is an interface between the input bodies, then a refinement window is created, which conforms to the interface.

Note:

The interface between bodies can be a geometric object that is not supported in the mesh command file and that is not supported by the other sdedr Scheme extensions. For example, if the interface is a polygon (wire body in two dimensions), this entity type is not supported by other sdedr extensions. In this case, the wire body is converted to an edge list, and a refinement window is created from each edge. If the original refinement window name is, for example, "rw", the "_edge_EN" string is appended to the name, where EN is an edge counter (rw_edge_1, rw_edge_2, and so on). Similarly, if the interface is a face list (3D), each face is converted to a polygon and the mesh command file includes these polygons if the interface refinement window is used to define doping/refinement. The refinement window name will have the "_lump_FN" string appended, where FN is a face number counter.

Syntax

```
(sdedr:define-body-interface-refwin body-list ref-eval-window)
```

Returns

BOOLEAN

Arguments

Argument	Description
body-list	Specifies a list of bodies. Argument type: BODY LIST
ref-eval-window	Specifies the name of a refinement window. Argument type: STRING

sdedr:define-constant-profile

This Scheme extension creates a definition for a constant doping profile in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Constant Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-constant-profile definition-name species concentration)
```

Returns

None

Arguments

Argument	Description
definition-name	Argument type: STRING
species	Argument type: STRING
concentration	Argument type: REAL

sdedr:define-constant-profile-material

This Scheme extension creates a constant doping profile placement based on a material in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Constant Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-constant-profile-material placement-name definition-name  
material [decay-length ["Gauss"]] [replace])
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
material	Argument type: DATEXMAT
decay-length	Optional. This argument corresponds to the mesh command file parameter "DecayLength" unless the optional string "Gauss" is also present, in which case, decay-length is interpreted instead as corresponding to the "GaussDecayLength" parameter of the mesh command file. Argument type: REAL
"Gauss"	Optional. Argument type: STRING
replace	Optional. Specifies the replacement method. Options are "Replace" "LocalReplace". Argument type: STRING

Appendix A: Commands

sdedr:define-constant-profile-placement

sdedr:define-constant-profile-placement

This Scheme extension creates a Ref/Eval window placement of a constant doping profile definition in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Constant Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-constant-profile-placement placement-name  
    definition-name ref-win-name [decay-length ["Gauss"]] [replace])
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
ref-win-name	Argument type: STRING
decay-length	Optional. This argument corresponds to the mesh command file parameter "DecayLength" unless the optional string "Gauss" is also present, in which case, decay-length is interpreted instead as corresponding to the "GaussDecayLength" parameter of the mesh command file. Argument type: REAL
"Gauss"	Optional. Argument type: STRING
replace	Optional. Specifies the replacement method. Options are "Replace" "LocalReplace". Argument type: STRING

Appendix A: Commands

sdedr:define-constant-profile-region

sdedr:define-constant-profile-region

This Scheme extension creates a constant doping profile placement based on a region in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Constant Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-constant-profile-region placement-name definition-name  
    region-name [decay-length ["Gauss"]] [replace])
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
region-name	Argument type: STRING
decay-length	Optional. This argument corresponds to the mesh command file parameter "DecayLength" unless the optional string "Gauss" is also present, in which case, decay-length is interpreted instead as corresponding to the "GaussDecayLength" parameter of the mesh command file. Argument type: REAL
"Gauss"	Optional. Argument type: STRING
replace	Optional. Specifies the replacement method. Options are "Replace" "LocalReplace". Argument type: STRING

Appendix A: Commands

sdedr:define-erf-profile

sdedr:define-erf-profile

This Scheme extension creates a definition of an error-function doping profile in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Constant Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-erf-profile definition-name species
  "SymPos" symmetry-position
  "MaxVal" max-value | "Dose" dose "Junction" junction
  "ValueAtDepth" value-at-depth "Depth" depth "Length" length
  "StdDev" standard-deviation [lateral-function])
```

Returns

None

Arguments

Argument	Description
definition-name	Argument type: STRING
species	Argument type: STRING
symmetry-position	Argument type: REAL
max-value	Argument type: REAL
dose	Argument type: REAL
junction	Argument type: REAL
value-at-depth	Argument type: REAL
depth	Argument type: REAL
length	Argument type: REAL
standard-deviation	Argument type: REAL

Appendix A: Commands

sdedr:define-gaussian-profile

Argument	Description
lateral-function	Optional. Specifies the lateral function. Possible values are "Erf", "Eval", or "Gauss". For example: "Erf" {"Factor" lateral-factor "Length" length} or: "Eval" {eval-init eval-function} or: "Gauss" {"Factor" lateral-factor "Length" length "StdDev" standard-deviation} Argument type: STRING
lateral-factor	Argument type: REAL
eval-init	Argument type: STRING
eval-function	Argument type: STRING

sdedr:define-gaussian-profile

This Scheme extension creates a definition of a Gaussian-function doping profile in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Analytic Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-gaussian-profile definition-name species
    "PeakPos" peak-position
    {"PeakVal" peak-concentration | "Dose" dose}
    {"ValueAtDepth" concentration-at-depth "Depth" depth |
     "Length" diffusion-length | "StdDev" standard-deviation}
    [lateral-function])
```

Returns

None

Arguments

Argument	Description
definition-name	Argument type: STRING
species	Argument type: STRING

Appendix A: Commands

sdedr:define-gaussian-profile

Argument	Description
peak-position	Argument type: REAL
peak-concentration	Argument type: REAL
dose	Argument type: REAL
concentration-at-depth	Argument type: REAL
depth	Argument type: REAL
diffusion-length	Argument type: REAL
standard-deviation	Argument type: REAL
lateral-function	Optional. Specifies the lateral function. Possible values are "Erf", "Eval", or "Gauss". For example: "Erf" {"Factor" lateral-factor "Length" length} or: "Eval" {eval-init eval-function} or: "Gauss" {"Factor" lateral-factor "Length" length "StdDev" standard-deviation} Argument type: STRING
lateral-factor	Argument type: REAL
length	Argument type: REAL
eval-init	Argument type: STRING
eval-function	Argument type: STRING

Appendix A: Commands

sdedr:define-multibox-placement

sdedr:define-multibox-placement

This Scheme extension defines the placements part of a multibox definition.

Syntax

```
(sdedr:define-multibox-placement placement-name definition-name  
refwindow-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
refwindow-name	Argument type: STRING

Examples

```
(sdedr:define-multibox-placement "p1" "d1" "refwindow1")
```

Appendix A: Commands

sdedr:define-multibox-size

sdedr:define-multibox-size

This Scheme extension defines the definitions part of a multibox definition.

Syntax

```
(sdedr:define-multibox-size mbox-name max-x max-y [max-z]
                           min-x min-y [min-z] ratio-x ratio-y [ratio-z])
```

Returns

None

Arguments

Argument	Description
mbox-name	Argument type: STRING
max-x	Argument type: REAL
max-y	Argument type: REAL
max-z	Optional. Argument type: REAL
min-x	Argument type: REAL
min-y	Argument type: REAL
min-z	Optional. Argument type: REAL
ratio-x	Argument type: REAL
ratio-y	Argument type: REAL
ratio-z	Optional. Argument type: REAL

Examples

```
(sdedr:define-multibox-size "multibox-2d" 10 10 1 1 2 2)
(sdedr:define-multibox-size "multibox-3d" 10 10 10 1 1 1 2 2)
```

sdedr:define-particle-profile

This Scheme extension creates a particle doping profile in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Particle Profiles, for descriptions of arguments.

Syntax

```
(sdedr:define-particle-profile name file
  [ "AutoScreeningFactor" auto-screening-factor]
  [ "BoundaryExtension" boundary-extension]
  [ "Divisions" divisions]
  [ "DopingAssignment" doping-assignment]
  [ "Normalization" normalization]
  [ "NumberOfThreads" number-of-threads]
  [ "ScreeningFactor" screening-factor]
  [ "ScreeningScalingFactor" screening-scaling-factor]
  [ "Species" species])
```

Returns

BOOLEAN

Arguments

Argument	Description
name	Argument type: STRING
file	Argument type: STRING
auto-screening-factor	Optional. Argument type: BOOLEAN
boundary-extension	Optional. Argument type: REAL
divisions	Optional. Argument type: INTEGER
doping-assignment	Optional. Specifies the doping assignment. Options are "CIC" "NGP" "Sano". Argument type: STRING
normalization	Optional. Argument type: BOOLEAN
number-of-threads	Optional. Argument type: INTEGER
screening-factor	Optional. Argument type: REAL
screening-scaling-factor	Optional. Argument type: BOOLEAN

Appendix A: Commands

sdedr:define-particle-profile-placement

Argument	Description
species	Optional. Argument type: STRING

sdedr:define-particle-profile-placement

This Scheme extension creates a placement for a particle profile definition in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Placing Particle Profiles, for descriptions of arguments.

Note:

Only "Replace" or "LocalReplace" can be set to true at the same time.

Syntax

```
(sdedr:define-particle-profile-placement placement-name
                                         definition-name "EvalWindow" eval-window eval-window-type
                                         ["Replace" replace | "LocalReplace" local-replace])
```

Returns

BOOLEAN

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
eval-window	Argument type: STRING
eval-window-type	Specifies the type of evaluation window. Options are "region" "material" "evalwin". Argument type: STRING
replace	Optional. Argument type: BOOLEAN
local-replace	Optional. Argument type: BOOLEAN

sdedr:define-refeval-window

This Scheme extension defines a geometric region that can be used as a Ref/Eval window.

You can create multiple body shapes by listing multiple single shapes. (Choose **Mesh > Define Ref/Eval Window.**)

The faces of the defined Ref/Eval windows are oriented. The face normal direction follows the right-hand rule (for polygonal Ref/Eval windows, the vertex list must be counterclockwise). In this case, the Ref/Eval windows are used in the [sdedr:define-analytical-profile-placement](#) Scheme extension for reference. The "Positive" and "Negative" arguments in this Scheme extension must be adjusted to the Ref/Eval window normal direction.

Syntax

```
(sdedr:define-refeval-window rfwin-name {multi-shape | single-shape})
```

Returns

ENTITY of the ACIS body Ref/Eval window if it is created successfully; otherwise, #f if the operation failed

Arguments

Argument	Description
rfwin-name	Specifies the name of a Ref/Eval window. Argument type: STRING
multi-shape	Specifies a list of single shapes by using the format: (list single-shape1 single-shape2 ... single-shapen) Argument type: STRING

Appendix A: Commands

sdedr:define-refeval-window

Argument	Description
single-shape	<p>Specifies a single body shape by using the format <code>rfwin-type</code> <code>data</code>, where:</p> <ul style="list-style-type: none"> • <code>rfwin-type</code> can be one of the following: "ComplexPolygon" "Cuboid" "Line" "Point" "Polygon" "Polyhedron" "Rectangle". • <code>data</code> can be one of the following: POSITION POSITION LIST. <p>Note: The <code>rfwin-type</code> object is not case sensitive.</p> <p>For <code>rfwin-type</code>:</p> <ul style="list-style-type: none"> • If set to "ComplexPolygon", which is used for multiconnected polygons, that is, polygons with internal doughnut holes, then <code>data</code> is a list of polygon shapes where the first polygon given is the outer perimeter, and the subsequent polygon shapes in the list represent internal holes. • If set to "Cuboid", "Line", or "Rectangle", then <code>data</code> is POSITION POSITION (the two opposite corners). The keywords "Cuboid" and "Rectangle" are axis aligned. • If set to "Point", then <code>data</code> is a position. • If set to "Polygon", then <code>data</code> is a position list (the first and last positions must be the same). • If set to "Polyhedron", then <code>data</code> is a list of polygon definitions creating a closed polyhedral solid from the union of the polygon faces. <p>Argument type: STRING</p>

Examples

```

; Line
(sdedr:define-refeval-window "rfwin1" "Line" (position 0 0 0)
  (position 1 0 0))

; Rectangle
(sdedr:define-refeval-window "rfwin2" "Rectangle" (position 0 -2 0)
  (position 1 -1 0))

; Polygon
(sdedr:define-refeval-window "rfwin3" "Polygon" (list
  (position 1 2 0.0) (position 0.75 2 0.0) (position 1 2.5 0.0)
  (position 1.25 3 0.0) (position 1.5 3.5 0.0) (position 1.75 4 0.0)
  (position 2 4.25 0.0) (position 2.25 4.5 0.0) (position 2.5 2 0.0)
  (position 1.5 2 0.0) (position 1 2 0.0)))

; Cuboid
(sdedr:define-refeval-window "rfwin4" "Cuboid" (position 0 0 0)
  (position 1 2 3))

; Multiconnected Polygon = ComplexPolygon
(sdedr:define-refeval-window "polygon-multiconnected" "ComplexPolygon"
  (list
    (list (position 0 0 0) (position 0 3 0) (position 3 3 0)
      (position 3 0 0)))

```

Appendix A: Commands

sdedr:define-refeval-window

```
(list (position 1 1 0) (position 1 2 0) (position 2 2 0)
      (position 2 1 0)))

; Polyhedron
(sdedr:define-refeval-window "polyhedron_shape" "Polyhedron" (list
  (list (position 0 0 0) (position 0 3 0) (position 3 3 0)
        (position 3 0 0))
  (list (position 0 0 3) (position 0 3 3) (position 3 3 3)
        (position 3 0 3))
  (list (position 0 0 0) (position 0 0 3) (position 3 0 3)
        (position 3 0 0))
  (list (position 0 3 0) (position 0 3 3) (position 3 3 3)
        (position 3 3 0))
  (list (position 0 0 0) (position 0 3 0) (position 0 3 3)
        (position 0 0 3))
  (list (position 3 0 0) (position 3 3 0) (position 3 3 3)
        (position 3 0 3)))))

; Multiple body
(sdedr:define-refeval-window "multiple_body" (list
  (list "Point" (position 0.15 0.25 0))
  (list "Rectangle" (position 0.1 0.2 0) (position 0.3 0.4 0))
  (list "Line" (position 1.5 1.6 0) (position 1.7 1.8 0))
  (list "Polygon" (list (position 0 0 0) (position 0.1 0.3 0)
    (position 0.2 0.4 0) (position 0.3 0.2 0) (position 0.2 0.0 0)
    (position 0.1 -0.1 0) (position 0 0 0)))
  (list "Cuboid" (position 3.1 3.2 3.3) (position 3.4 3.5 3.6))))
```

Appendix A: Commands

sdedr:define-refinement-function

sdedr:define-refinement-function

This Scheme extension adds a refinement function to the specified refinement.

See *Sentaurus™ Mesh User Guide*, Defining Refinement Regions, for descriptions of arguments.

Syntax

```
(sdedr:define-refinement-function definition-name function-name
  { "MaxLenInt" mat-reg mat-reg value [factor] [ "DoubleSide" ]
    [ "UseRegionNames" ] |
    "MaxInterval" "Variable" dopant-name "Cmin" cmin "Cmax" cmax
    [ "Scaling" scaling] [ "TargetLength" targetLength] [ "Rolloff" ] |
    "MaxGradient" value | "MaxTransDiff" value})
```

Returns

None

Arguments

Argument	Description
definition-name	Argument type: STRING
function-name	Specifies the name of the function to be used for refinement. Argument type: STRING
mat-reg	Specifies the material name or region name defining the interface. Argument type: STRING
value	Argument type: REAL
factor	Optional. Argument type: REAL
"DoubleSide"	Optional. Argument type: STRING
"UseRegionNames"	Optional. Argument type: STRING
dopant-name	Argument type: STRING
cmin	Argument type: REAL REAL LIST
cmax	Argument type: REAL REAL LIST
scaling	Argument type: REAL

Appendix A: Commands

sdedr:define-refinement-material

Argument	Description
targetLength	Optional. Argument type: REAL
"Rolloff"	Optional. Argument type: STRING

sdedr:define-refinement-material

This Scheme extension defines the placements information for a *materialwise* refinement.

Syntax

```
(sdedr:define-refinement-material placement-name definition-name  
material-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
material-name	Argument type: DATEXMAT

Examples

```
(sdedr:define-refinement-material "p1" "d1" "Silicon")
```

Appendix A: Commands

sdedr:define-refinement-placement

sdedr:define-refinement-placement

This Scheme extension creates a refinement placement in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Refinement Regions, for descriptions of arguments.

Syntax

```
(sdedr:define-refinement-placement refinement-name definition-name  
ref-eval-window)
```

Returns

None

Arguments

Argument	Description
refinement-name	Argument type: STRING
definition-name	Argument type: STRING
ref-eval-window	Specifies either the name of a defined refinement window or a list of Ref/Eval windows, materials, and regions in the form: <code>(list "window" "refevalwin1" "window" "refevalwindow2" ... "material" "material1" ... "region" "region1" ...)</code> If a list is specified, then the refinement is performed at the intersection of all the criteria. Argument type: STRING STRING LIST

Appendix A: Commands

sdedr:define-refinement-region

sdedr:define-refinement-region

This Scheme extension defines the placements information for a *regionwise* refinement.

Syntax

```
(sdedr:define-refinement-region placement-name definition-name  
                                 region-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
region-name	Argument type: STRING

Examples

```
(sdedr:define-refinement-region "p1" "d1" "Region_1")
```

sdedr:define-refinement-size

This Scheme extension creates a refinement size definition in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Refinement Regions, for descriptions of arguments.

Syntax

```
(sdedr:define-refinement-size definition-name max-x [max-y] [max-z]
                               min-x [min-y] [min-z])
```

Returns

None

Arguments

Argument	Description
definition-name	Argument type: STRING
max-x	Argument type: REAL
max-y	Optional. Argument type: REAL
max-z	Optional. Argument type: REAL
min-x	Argument type: REAL
min-y	Optional. Argument type: REAL
min-z	Optional. Argument type: REAL

Appendix A: Commands

sdedr:define-submesh

sdedr:define-submesh

This Scheme extension creates a 1D external profile in the mesh command file.

See *Sentaurus™ Mesh User Guide*, Defining Submeshes, for descriptions of arguments.

Syntax

```
(sdedr:define-submesh definition-name geofile-name  
    file1 mode1 file2 mode2 ... filen moden  
    [ (list field1 field2 ... fieldn) ])
```

or:

```
(sdedr:define-submesh definition-name geofile-name  
    file1 mode1 file2 mode2 ... filen moden  
    [ (list field1 conc1 field2 conc2 ... fieldn concn) ])
```

Returns

None

Arguments

Argument	Description
definition-name	Argument type: STRING
geofile-name	Argument type: STRING
file[1...n]	Argument type: STRING STRING LIST
mode[1...n]	Options are "r" "i" "o" "w" "w=outfile-name". Argument type: STRING
field[1...n]	Optional. Argument type: STRING STRING LIST
conc[1...n]	Optional. Argument type: REAL REAL LIST

Appendix A: Commands

sdedr:define-submesh-placement

sdedr:define-submesh-placement

This Scheme extension creates a submesh placement in the mesh command file.

To rotate, shift, or reflect submesh placements, use the Scheme extension
sdedr:transform-submesh-placement.

See *Sentaurus™ Mesh User Guide*, Placements Section, for descriptions of arguments.

Syntax

```
(sdedr:define-submesh-placement placement-name definition-name
    evaluation-window
    ["PlacementType" eval-window-type]
    ["DecayLength" decay-length | "GaussDecayLength" gauss-decay-length]
    ["Replace" | "NoReplace" | "LocalReplace"]
    ["IgnoreMat"] ["MatchMaterialType"])
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING
definition-name	Argument type: STRING
evaluation-window	Specifies the name of a region, material, or Ref/Eval window, depending on eval-window-type. Argument type: STRING
eval-window-type	Optional. Specifies the type of evaluation window. Options are "evalwin" "material" "region". The default is "evalwin". Argument type: STRING
decay-length	Optional. Argument type: REAL
gauss-decay-length	Optional. Argument type: REAL

sdedr:del-selected-drentity

This Scheme extension deletes a specified entity or list of entities.

Syntax

```
(sdedr:del-selected-drentity entity | entity-list)
```

Returns

None

Arguments

Argument	Description
entity	Specifies the name of an entity to be deleted. Argument type: ENTITY
entity-list	Specifies a list of entity names to be deleted. Argument type: ENTITY LIST

sdedr:delete-multibox-placement

This Scheme extension deletes the specified multibox placement from the placements part of the command file.

Syntax

```
(sdedr:delete-multibox-placement placement-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING

Examples

```
(sdedr:delete-multibox-placement "multibox_1")
```

Appendix A: Commands

sdedr:delete-profile-placement

sdedr:delete-profile-placement

This Scheme extension deletes the specified placement.

Syntax

```
(sdedr:delete-profile-placement placement-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING

sdedr:delete-refeval-window

This Scheme extension deletes a Ref/Eval window or list of Ref/Eval windows.

Syntax

```
(sdedr:delete-refeval-window bodies)
```

Returns

None

Arguments

Argument	Description
bodies	Specifies either a Ref/Eval window or a list of Ref/Eval windows to be deleted. Argument type: BODY BODY LIST

Examples

```
(sdedr:define-refeval-window "rw1" "Rectangle" (position 0 0 0)
                               (position 1 1 0))
(sdedr:define-refeval-window "rw2" "Rectangle" (position 1 0 0)
                               (position 2 1 0))
(sdedr:define-refeval-window "rw3" "Rectangle" (position 2 0 0)
                               (position 3 1 0))
(sdedr:delete-refeval-window "rw1") ; or equivalently
```

Appendix A: Commands

sdedr:delete-refinement-placement

```
(sdedr:delete-refeval-window (list "rw1"))
(sdedr:delete-refeval-window (list "rw2" "rw3"))
; Refinement/evaluation windows can also be deleted, by selecting
; them and using (sde:selected-refeval-windows)
(sdedr:delete-refeval-window (sde:selected-refeval-windows))
```

sdedr:delete-refinement-placement

This Scheme extension deletes the specified refinement placement from the `Placements` section of the command file.

Syntax

```
(sdedr:delete-refinement-placement placement-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING

Examples

```
(sdedr:delete-refinement-placement "refinement_1")
```

Appendix A: Commands

sdedr:delete-submesh-placement

sdedr:delete-submesh-placement

This Scheme extension deletes the specified placement.

Syntax

```
(sdedr:delete-submesh-placement placement-name)
```

Returns

None

Arguments

Argument	Description
placement-name	Argument type: STRING

sdedr:get-cmdprecision

This Scheme extension returns the precision value of the mesh command file. The default value is 12.

Syntax

```
(sdedr:get-cmdprecision)
```

Returns

INTEGER

sdedr:get-definition-list

This Scheme extension returns a list of definition names of one of the placement types.

Syntax

```
(sdedr:get-definition-list "multibox" | "profile" | "refinement")
```

Returns

STRING LIST

Appendix A: Commands

sdedr:get-placement-list

sdedr:get-placement-list

This Scheme extension returns a list of placement names of one of the placement types.

Syntax

```
(sdedr:get-placement-list "multibox" | "profile" | "refinement")
```

Returns

STRING LIST

sdedr:hide-mbox

This Scheme extension hides the specified multiboxes.

Syntax

```
(sdedr:hide-mbox mbox-list)
```

Returns

None

Arguments

Argument	Description
mbox-list	Specifies a list of multiboxes to hide. Argument type: STRING LIST

Appendix A: Commands

sdedr:hide-profile

sdedr:hide-profile

This Scheme extension hides the specified analytic doping profiles.

Syntax

```
(sdedr:hide-profile profile-list)
```

Returns

None

Arguments

Argument	Description
profile-list	Specifies a list of doping profiles to hide. Argument type: STRING LIST

sdedr:hide-refinement

This Scheme extension hides the specified refinements.

Syntax

```
(sdedr:hide-refinement refinement-list)
```

Returns

None

Arguments

Argument	Description
refinement-list	Specifies a list of refinements to hide. Argument type: STRING LIST

Examples

```
(sde:clear)
(sdgeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "region_1")
(sdgeo:create-rectangle (position 4 10 0) (position 6 12 0)
    "PolySilicon" "region_2")
(sdedr:define-refinement-size "ref1" 2 2 2 1 1 1)
(sdedr:define-refinement-region "ref1" "ref1" "region_1")
```

Appendix A: Commands

sdedr:hide-rewin

```
(sdedr:show-refinement "ref1")
(sdedr:hide-refinement "ref1")
```

sdedr:hide-rewin

This Scheme extension hides the specified Ref/Eval window or windows.

Syntax

```
(sdedr:hide-rewin rewin-list)
```

Returns

None

Arguments

Argument	Description
rewin-list	Specifies a Ref/Eval window or a list of Ref/Eval windows to hide. Argument type: STRING LIST

Examples

```
(sdedr:define-refeval-window "N1" "Rectangle" (position 0 0 0)
  (position 1 1 0))
(sdedr:define-refeval-window "N2" "Rectangle" (position 2 0 0)
  (position 3 1 0))
(sdedr:define-refeval-window "N3" "Rectangle" (position 4 0 0)
  (position 5 1 0))
(sdedr:define-refeval-window "N4" "Rectangle" (position 0 2 0)
  (position 1 3 0))
(sdedr:define-refeval-window "N5" "Rectangle" (position 2 2 0)
  (position 3 3 0))
(sdedr:define-refeval-window "N6" "Rectangle" (position 4 2 0)
  (position 5 3 0))
(sdedr:define-refeval-window "N7" "Rectangle" (position 0 4 0)
  (position 1 5 0))
(sdedr:define-refeval-window "N8" "Rectangle" (position 2 4 0)
  (position 3 5 0))
(sdedr:define-refeval-window "N9" "Rectangle" (position 4 4 0)
  (position 5 5 0))
(sdedr:hide-rewin (list "N1" "N2" "N3" "N4" "N5" "N6" "N7" "N8" "N9"))
(sdedr:show-rewin "N1")
(sdedr:show-rewin (list "N2" "N3" "N4"))
```

Appendix A: Commands

sdedr:offset-block

sdedr:offset-block

This Scheme extension generates a new offsetting block inside the `Offsetting` section of the mesh command file.

Syntax

```
(sdedr:offset-block "material" material "maxlevel" maxlevel)  
(sdedr:offset-block "region" region "maxlevel" maxlevel)
```

Returns

None

Arguments

Argument	Description
material	Argument type: DATEXMAT
maxlevel	Argument type: INTEGER
region	Argument type: STRING

Appendix A: Commands

sdedr:offset-interface

sdedr:offset-interface

This Scheme extension generates a new offsetting interface block inside the Offsetting section of the mesh command file.

Syntax

```
(sdedr:offset-interface "region" region1 region2  
    "hlocal" hlocal "factor" factor "window" x1 y1 z1 x2 y2 z2)  
  
(sdedr:offset-interface "material" material1 material2  
    "hlocal" hlocal "factor" factor "window" x1 y1 z1 x2 y2 z2)
```

Returns

None

Arguments

Argument	Description
region	Argument type: STRING
material	Argument type: DATEXMAT
hlocal	Argument type: REAL
factor	Argument type: REAL
x1 y1 z1 x2 y2 z2	Argument type: REAL

Appendix A: Commands

sdedr:read-cmd-file

sdedr:read-cmd-file

This Scheme extension parses a specified mesh command file and initializes the doping-related and refinement-related data structures in Sentaurus Structure Editor. The command file entities can be visualized using the Placements Viewer.

The command file entities can be manipulated by using the sdedr Scheme extensions or the Sentaurus Structure Editor GUI menus.

Syntax

```
(sdedr:read-cmd-file cmd-file-name)
```

Returns

None

Arguments

Argument	Description
cmd-file-name	Specifies the name of a mesh command file to load. Argument type: STRING

Appendix A: Commands

sdedr:redefine-refeval-window

sdedr:redefine-refeval-window

This Scheme extension redefines a previously defined geometric Ref/Eval window. It is used mainly in renaming existing Ref/Eval windows.

Syntax

```
(sdedr:redefine-refeval-window rfwin-name new-rfwin-name  
rfwin-type data)
```

Returns

ENTITY ID of the redefined ACIS body Ref/Eval window; otherwise, #f if the operation fails

Arguments

Argument	Description
rfwin-name	Specifies the name of a Ref/Eval window. Argument type: STRING
new-rfwin-name	Specifies the new name of a Ref/Eval window. Argument type: STRING
rfwin-type	Specifies the type of Ref/Eval window. Options are "Cuboid" "Line" "Point" "Polygon" "Rectangle". For rfwin-type: <ul style="list-style-type: none">If set to "Cuboid", "Line", or "Rectangle", then data is POSITION POSITION (the two opposite corners).If set to "Point", then data is a position.If set to "Polygon", then data is a position list (the first and last positions must be the same). Argument type: STRING
data	Specifies the data for a Ref/Eval window. Argument type: POSITION POSITION LIST

Appendix A: Commands

sdedr:refine-box

sdedr:refine-box

This Scheme extension defines a refinement window–style refinement box.

It defines a refinement rectangle in two dimensions and a refinement cuboid in three dimensions.

Syntax

```
(sdedr:refine-box refinement-name pos-min pos-max spacing)
```

Returns

None

Arguments

Argument	Description
refinement-name	Specifies the name of a refinement box. Argument type: STRING
pos-min	Specifies the minimum coordinate of the box. Argument type: POSITION
pos-max	Specifies the maximum coordinate of the box. Argument type: POSITION
spacing	Specifies either a single value, which denotes the mesh spacing in all directions, or a list of three values for spacing in each direction. Argument type: REAL REAL REAL REAL

Appendix A: Commands

sdedr:refine-doping

sdedr:refine-doping

This Scheme extension adds a refinement based on doping
(field:DopingConcentration) in an entire region or material.

Syntax

```
(sdedr:refine-doping spacing | sx sy sz
  [ "MaxTransDiff" asinhdiff]
  [ "material" material-name | "region" region-name])
```

Returns

None

Arguments

Argument	Description
spacing	Specifies either a single value, which denotes the mesh spacing in all directions, or a list of three values for spacing in each direction. Argument type: REAL REAL REAL REAL
sx sy sz	Argument type: REAL REAL REAL
asinhdiff	Optional. Specifies the value for "MaxTransDiff". If you do not specify any value, then 1.0 is used. Argument type: REAL
material-name	Optional. Specifies the name of the material in which the refinement is performed. If you do not specify any value, then material = "Silicon" is used. Argument type: STRING
region-name	Optional. Specifies the name of a region. Argument type: STRING

Appendix A: Commands

sdedr:refine-interface

sdedr:refine-interface

This Scheme extension creates a refined mesh at an interface.

Note:

Following the convention used in Sentaurus Structure Editor and Sentaurus Mesh, but unlike Sentaurus Process, the refinement is placed only in region `r1` or material `m1`, unless you specify a window, in which case, the placement is performed in the entire window.

Syntax

```
(sdedr:refine-interface "region" r1 r2 | "material" m1 m2  
"hlocal" spacing ["factor" factor] ["window" x1 y1 z1 x2 y2 z2])
```

Returns

None

Arguments

Argument	Description
"region" r1 r2	Specifies a region interface, where <code>r1</code> specifies the first region and <code>r2</code> specifies the second region. Argument type: STRING
"material" m1 m2	Specifies a material interface, where <code>m1</code> specifies the first material and <code>m2</code> specifies the second material. Argument type: STRING
spacing	Specifies the initial normal spacing at the interface (scalar). Argument type: REAL
factor	Optional. Determines how fast the spacing increases. The default is 2.0. Argument type: REAL
x1 y1 z1 x2 y2 z2	Optional. Define the window corners if the placement should be restricted to a window. The default is no window; placement in <code>r1</code> or <code>m1</code> . Argument type: REAL

Appendix A: Commands

sdedr:set-cmdprecision

sdedr:set-cmdprecision

This Scheme extension sets the precision value of the mesh command file. The number of digits that is written in the mesh command file for Ref/Eval window coordinates can be controlled. The vertex coordinates of the Ref/Eval windows that are written to the mesh command file can be chopped and rounded.

Rounding the vertex coordinates of the Ref/Eval objects might be needed, for example, when a rigid body rotation is performed on a Ref/Eval window, which will introduce some rounding errors.

The default is 12.

Syntax

```
(sdedr:set-cmdprecision cmdprecision)
```

Returns

None

Arguments

Argument	Description
cmdprecision	Argument type: INTEGER

Appendix A: Commands

sdedr:set-title

sdedr:set-title

This Scheme extension sets the title of the command file.

Information is written to the `Title` section of the command file. If this Scheme extension is not called explicitly, then the title of the command file is set to `Untitled`.

Syntax

```
(sdedr:set-title cmd-file-title)
```

Returns

None

Arguments

Argument	Description
cmd-file-title	Specifies the title of the command file. Argument type: STRING

sdedr:show-mbox

This Scheme extension displays the specified multiboxes.

Syntax

```
(sdedr:show-mbox mbox-list)
```

Returns

None

Arguments

Argument	Description
mbox-list	Specifies a list of multiboxes to be displayed. Argument type: STRING LIST

Appendix A: Commands

sdedr:show-profile

sdedr:show-profile

This Scheme extension displays the specified analytic doping profiles.

Syntax

```
(sdedr:show-profile profile-list)
```

Returns

None

Arguments

Argument	Description
profile-list	Specifies a list of doping profiles to be displayed. Argument type: STRING LIST

sdedr:show-refinement

This Scheme extension displays the specified refinements.

Syntax

```
(sdedr:show-refinement refinement-list)
```

Returns

None

Arguments

Argument	Description
refinement-list	Specifies a list of refinements to be displayed. Argument type: STRING LIST

Examples

```
(sde:clear)
(sdgeo:create-rectangle (position 0 0 0) (position 10 10 0) "Silicon"
    "region_1")
(sdgeo:create-rectangle (position 4 10 0) (position 6 12 0)
    "PolySilicon" "region_2")
(sdedr:define-refinement-size "ref1" 2 2 2 1 1 1)
(sdedr:define-refinement-region "ref1" "ref1" "region_1")
(sdedr:show-refinement "ref1")
```

Appendix A: Commands

sdedr:show-rewin

sdedr:show-rewin

This Scheme extension displays the specified Ref/Eval window or windows.

Syntax

```
(sdedr:show-rewin rewin-list)
```

Returns

None

Arguments

Argument	Description
rewin-list	Specifies a Ref/Eval window or a list of Ref/Eval windows to be displayed. Argument type: STRING LIST

Examples

```
(sdedr:define-refeval-window "N1" "Rectangle" (position 0 0 0)
  (position 1 1 0))
(sdedr:define-refeval-window "N2" "Rectangle" (position 2 0 0)
  (position 3 1 0))
(sdedr:define-refeval-window "N3" "Rectangle" (position 4 0 0)
  (position 5 1 0))
(sdedr:define-refeval-window "N4" "Rectangle" (position 0 2 0)
  (position 1 3 0))
(sdedr:define-refeval-window "N5" "Rectangle" (position 2 2 0)
  (position 3 3 0))
(sdedr:define-refeval-window "N6" "Rectangle" (position 4 2 0)
  (position 5 3 0))
(sdedr:define-refeval-window "N7" "Rectangle" (position 0 4 0)
  (position 1 5 0))
(sdedr:define-refeval-window "N8" "Rectangle" (position 2 4 0)
  (position 3 5 0))
(sdedr:define-refeval-window "N9" "Rectangle" (position 4 4 0)
  (position 5 5 0))
(sdedr:hide-rewin (list "N1" "N2" "N3" "N4" "N5" "N6" "N7" "N8" "N9"))
(sdedr:show-rewin "N1")
(sdedr:show-rewin (list "N2" "N3" "N4"))
```

sdedr:transform-submesh-placement

This Scheme extension defines a transform operation, or a series of transform operations, applied to the named external profile placement. The operation or series of operations is appended to any existing series of operations already applied to the placement.

You can specify any number of operations in this Scheme extension, and you can this Scheme extension any number of times. Operations are performed in the order of calls to this extension and in the order listed within each call.

The Scheme extension `sdedr:clear-submesh-placement-transform` is used to clear the list of transformation operations for a given external profile placement.

Syntax

```
(sdedr:transform-submesh-placement placement-name  
[<transform1> <transform2> ... <transformN>])
```

Returns

None

Arguments

Argument	Description
placement-name	Specifies the name of an external profile placement. Argument type: STRING
<transform1>...<transformN>	Specifies any of the following transformations: <ul style="list-style-type: none">• "Reflect" axis• "Rotation" angle axis• "ShiftVector" (gvector x y z) For these transformations: <ul style="list-style-type: none">• axis specifies the axis of the transformation. Options are "X" "Y" "Z". Argument type: STRING• angle specifies the angle of the transformation. Argument type: REAL• x, y, z specify the values of the gvector. Argument type: REAL

Appendix A: Commands

sdedr:write-cmd-file

sdedr:write-cmd-file

This Scheme extension writes all doping-related and refinement-related data to the specified mesh command file. The command file can be used during a subsequent meshing action, together with the generated boundary file. By using a command file, you can control, for example, local mesh refinements during meshing.

Syntax

```
(sdedr:write-cmd-file cmd-file-name)
```

Returns

None

Arguments

Argument	Description
cmd-file-name	Specifies the name of a mesh command file. Argument type: STRING

sdedr:write-scaled-cmd-file

This Scheme extension writes all doping-related and refinement-related data to the specified mesh command file (after scaling some dimensions), which can be used during a subsequent meshing action, together with the generated boundary file. By using a command file, you can control, for example, local mesh refinements, during meshing.

Syntax

```
(sdedr:write-scaled-cmd-file cmd-file-name scaling-factor)
```

Returns

None

Arguments

Argument	Description
cmd-file-name	Specifies the name of a mesh command file. Argument type: STRING
scaling-factor	Specifies a scaling factor. All position and length parameters are <i>multiplied</i> by this factor. All gradient parameters are <i>divided</i> by this factor. Argument type: REAL

Appendix A: Commands

sdeepi:create-layerstack

sdeepi:create-layerstack

This Scheme extension creates a planar layer stack structure from the CSV input data.

If the global Scheme variable `sdeepi:use-global-vars` is set to `#t` (default: `#f`), then the global and layer variables of the `sdeepi` Scheme extensions also are initialized as global Scheme variables and can be used for subsequent computations.

Syntax

```
(sdeepi:create-layerstack file-name)
```

Returns

None

Arguments

Argument	Description
file-name	Specifies the name of a CSV file. Argument type: STRING

Appendix A: Commands

sdeepi:publish-global-vars

sdeepi:publish-global-vars

This Scheme extension defines the global and layer variables of the sdeepi Scheme extensions as global Scheme variables.

If you call this Scheme extension with a #t argument, then all the published global and layer variables are available as global Scheme variables after the epi layers are created using the sdeepi:create-layerstack Scheme extension. By default, to avoid overwriting existing Scheme variables, no layer stack variables are retained after the stack creation.

Syntax

```
(sdeepi:publish-global-vars make-global-vars)
```

Returns

None

Arguments

Argument	Description
make-global-vars	Specifies whether to define the global and layer variables of sdeepi Scheme extensions as global Scheme variables. Argument type: BOOLEAN

Appendix A: Commands

sdeepi:scm

sdeepi:scm

This Scheme extension generates a Scheme script for creating a planar layer stack structure from CSV input data. This can be useful to investigate the layer stack creation in more detail. The Scheme script is named like `filename` but with `.scm` as the file extension unless it is specified explicitly using the optional `output-filename` argument.

Syntax

```
(sdeepi:scm filename [output-filename])
```

Returns

None

Arguments

Argument	Description
filename	Specifies the name of a Scheme script. Argument type: STRING
output-filename	Optional. Specifies the name of an output script. Argument type: STRING

Appendix A: Commands

sdeepi:tcl

sdeepi:tcl

This Scheme extension creates a Tcl script that contains all layer stack-specific data from the CSV input file in the form of Tcl variables. This file can be easily sourced in Tcl preprocessing blocks or any other Tcl interpreter such as Sentaurus Visual or Inspect to access layer data (see [Global Section on page 289](#) for an example).

The Tcl file is named like `filename` but with `.tcl` as the file extension unless it is specified explicitly using the optional `output-filename` argument.

Syntax

```
(sdeepi:tcl filename [output-filename])
```

Returns

None

Arguments

Argument	Description
<code>filename</code>	Specifies the name of a Tcl file. Argument type: STRING
<code>output-filename</code>	Optional. Specifies the name of an output file. Argument type: STRING

Appendix A: Commands

sdegeo:2d-cut

sdegeo:2d-cut

This Scheme extension extracts a 2D rectangular part from a 2D device.

Syntax

(sdegeo:2d-cut p1 p2)

Returns

None

Arguments

Argument	Description
p1	Specifies one of the opposite corners of the rectangle. Argument type: POSITION
p2	Specifies the other opposite corner of the rectangle. Argument type: POSITION

Appendix A: Commands

sdegeo:3d-cut

sdegeo:3d-cut

This Scheme extension cuts a 2D slice from a 3D device. It transfers 3D face contacts to 2D edge contacts.

Syntax

```
(sdegeo:3d-cut base-position normal-vector  
[tr-xy] [tr-record] [tr-axisaligned])
```

Returns

None

Arguments

Argument	Description
base-position	Specifies the base point used to define the cutplane. Argument type: POSITION
normal-vector	Specifies the normal vector to the cutplane. Argument type: GVECTOR
tr-xy	Optional. Specifies whether to transform the 2D cut to the xy plane. This is needed when the 2D cut is to be meshed. The default is #f. Argument type: BOOLEAN
tr-record	Optional. Specifies whether to record the transform (rotation and translation) that transforms the 2D cut on the xy plane. The default is #f. Argument type: BOOLEAN
tr-axisaligned	Optional. Specifies whether to trigger new transformation rules for axis-aligned cuts. If the new transformation rule is applied, then the positioning of the 2D cut can be calculated explicitly following the rules in Table 41 . The default is #f. Argument type: BOOLEAN

Description

You can use the `sde:tr-get` Scheme extension to return the rotation (3×3 matrix) and translation (vector) components of the transform (see [sde:tr-get on page 521](#)).

The `sdegeo:3d-cut` Scheme extension always creates a well-defined 2D model. If the original 3D model contains multiple touching faces at the cut position in the cutplane (material or region interfaces), then the direction of the normal vector (specified in the argument list) defines which regions to keep in the 2D model.

Appendix A: Commands

sdegeo:3d-cut

The 2D cut will contain only those faces that are visible from the normal plane direction in the 3D model. In this way, the outcome of the `sdegeo:3d-cut` operation is always well defined, and the 2D cut does not contain overlapping faces. If the 2D sheet bodies in the 2D cut contain multiple faces, then these faces are merged to one face.

Transformation Rules for Axis-Aligned Cuts

The model is placed such that the minimum bounding box position (lower-left corner) is the same as the minimum bounding box position of the 3D model at the cutplane position. The axis-aligned transformation rule determines which coordinates to use for `xmin` and `ymin` for the 2D cut from the global `x,y,z` bounding box minimum.

Table 41 Transformation rules for axis-aligned cuts

Cutplane normal	New 2D coordinate axes (x, y) with respect to global coordinate system
(1, 0, 0)	(0,1,0) (0,0,1)
(-1, 0, 0)	(0,1,0) (0,0,1)
(0, 1, 0)	(1,0,0) (0,0,1)
(0, -1, 0)	(1,0,0) (0,0,1)
(0, 0, 1)	(1,0,0) (0,1,0)
(0, 0, -1)	(1,0,0) (0,1,0)

Note:

The + and – face normal directions are not distinguished, so the (1,0,0) cutplane normal generates the same result as the (-1,0,0) normal.

The transformation rules for axis-aligned cuts are:

- If the gvector is parallel to the x-axis, then the 2D cut x-axis is the global y-axis, and the 2D cut y-axis is the global z-axis. The global bounding box minimum coordinates `bbxmin` and `bbzmin` are the minimum coordinates of the 2D cut: `xmin = bbymin`, `ymin = bbzmin`.
- If the gvector is parallel to the y-axis, then the 2D cut x-axis is the global x-axis, and the 2D cut y-axis is the global z-axis. The global bounding box minimum coordinates `bbxmin` and `bbzmin` are the minimum coordinates of the 2D cut: `xmin = bbxmin`, `ymin = bbzmin`.
- If the gvector is parallel to the z-axis, then the 2D cut x-axis is the global x-axis, and the 2D cut y-axis is the global y-axis. The global bounding box minimum coordinates `bbxmin`

Appendix A: Commands

sdegeo:align-horizontal

and `bbymin` are the minimum coordinates of the 2D cut: `xmin = bbxmin`, `ymin = bbymin`.

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0) "Silicon"
    "region_1")
(sdegeo:define-contact-set "cstop" 4 (color:rgb 1 0 0) "##")
(sdegeo:define-contact-set "csbottom-middle" 4 (color:rgb 1 1 0) "||")
(sdegeo:set-contact (find-edge-id (position 5 10 0)) "cstop")
(sdegeo:insert-vertex (position 3 0 0))
(sdegeo:insert-vertex (position 7 0 0))
(sdegeo:set-current-contact-set "csbottom-middle")
(sdegeo:set-contact (find-edge-id (position 5 0 0)) "csbottom-middle")
; 2D model containing edge contacts
(sdegeo:extrude (find-face-id (position 5 5 0)) 6)
; 3D model containing face contacts
(sdegeo:3d-cut (position 5 0 2) (gvector 0 0 1))
; 2D model containing edge contacts
```

sdegeo:align-horizontal

This Scheme extension aligns the specified vertices to a horizontal line.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:align-horizontal vertex-list align-position)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: VERTEX_LIST
align-position	Specifies the position (y-coordinate) of the horizontal line. Argument type: REAL

sdegeo:align-horizontal-aut

This Scheme extension aligns the specified vertices to a horizontal line. The align position (y-coordinate) of the line is defined using the average y-coordinate of the specified vertices, which is calculated automatically.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:align-horizontal-aut vertex-list)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: VERTEX LIST

Appendix A: Commands

sdegeo:align-to-line

sdegeo:align-to-line

This Scheme extension aligns the specified vertices to a line.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:align-to-line vertex-list pos1 pos2)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: VERTEX_LIST
pos1, pos2	Specify the two points of the line. The points must lie on the same work plane as the listed vertices. The vertices are projected on to the specified line. Argument type: POSITION

Appendix A: Commands

sdegeo:align-vertical

sdegeo:align-vertical

This Scheme extension aligns the specified vertices to a vertical line.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:align-vertical vertex-list align-position)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: VERTEX_LIST
align-position	Specifies the new position (x-coordinate) of the vertical line. Argument type: REAL

Appendix A: Commands

sdegeo:align-vertical-aut

sdegeo:align-vertical-aut

This Scheme extension aligns the specified vertices to a vertical line. The align position (x-coordinate) of the line is defined using the average x-coordinate of the specified vertices, which is calculated automatically.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:align-vertical-aut vertex-list)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: VERTEX LIST

Appendix A: Commands

sdegeo:average-edge-length

sdegeo:average-edge-length

This Scheme extension computes the length of each edge in the list and returns the average edge length.

Syntax

```
(sdegeo:average-edge-length edge-list)
```

Returns

REAL

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "AB")
(sdegeo:create-rectangle (position 0 0 0) (position 2 2 0) "Silicon"
"region_1")
(sdegeo:create-circle (position 1 2 0) 1 "Silicon" "region_2")
(define elist (entity:edges (get-body-list)))
(sdegeo:average-edge-length elist)
```

Appendix A: Commands

sdegeo:body-trim

sdegeo:body-trim

This Scheme extension trims the model to the specified box. All parts of the geometric bodies that lie outside of the specified box are removed.

Syntax

```
(sdegeo:body-trim x0 y0 z0 x1 y1 z1)
```

Returns

None

Arguments

Argument	Description
x0	Argument type: REAL
y0	Argument type: REAL
z0	Argument type: REAL
x1	Argument type: REAL
y1	Argument type: REAL
z1	Argument type: REAL

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 5) (position 10 10 10)
    "Silicon" "x1")
(sdegeo:create-cuboid (position 0 0 10) (position 10 10 11)
    "PolySilicon" "x2")
(sdegeo:create-cuboid (position 2 2 11) (position 8 8 14) "Copper"
    "x3")
(sdegeo:create-cylinder (position 5 5 14) (position 5 5 18) 3 "Silver"
    "x4")
(sdegeo:body-trim 5 5 5 20 20 20)
```

Appendix A: Commands

sdegeo:bool-intersect

sdegeo:bool-intersect

This Scheme extension performs a Boolean intersection operation, that is, it intersects the first body from the specified body list with the rest of the bodies from the list. The resulting body inherits the DATEX material attribute from the first body. Similarly, the region-name attribute is inherited from the first entity in the body list.

This Scheme extension applies to both 2D and 3D bodies.

If the specified body list contains empty bodies, then the Scheme extension returns #f.

If the result of the Boolean operation is an empty body, then the empty body is removed from the entity list, and #t is returned.

Syntax

```
(sdegeo:bool-intersect body-list)
```

Returns

ENTITY | BOOLEAN

Arguments

Argument	Description
body-list	Specifies a list of bodies. Argument type: BODY LIST

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "XX")
(define b1 (sdegeo:create-rectangle (position 0 0 0)
                                      (position 10 10 0) "Silicon" "region_1"))
(define b2 (sdegeo:create-rectangle (position 5 5 0)
                                      (position 12 12 0) "PolySilicon" "region_2"))
(sdegeo:bool-intersect (list b2 b1))
```

Appendix A: Commands

sdegeo:bool-subtract

sdegeo:bool-subtract

This Scheme extension performs a Boolean subtraction operation, that is, it subtracts the first body from the specified body list with the rest of the bodies from the list. The resulting body inherits the DATEX material attribute of the first body. Similarly, the `region-name` attribute is inherited from the first entity in the body list.

This Scheme extension applies to both 2D and 3D bodies.

If the specified body list contains empty bodies, then the Scheme extension returns #f.

If the result of the Boolean operation is an empty body, then the empty body is removed from the entity list, and #t is returned.

Syntax

```
(sdegeo:bool-subtract body-list)
```

Returns

ENTITY | BOOLEAN

Arguments

Argument	Description
body-list	Specifies a list of bodies. Argument type: BODY LIST

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "XX")
(define b1 (sdegeo:create-rectangle (position 0 0 0)
                                      (position 10 10 0) "Silicon" "region_1"))
(define b2 (sdegeo:create-rectangle (position 5 5 0)
                                      (position 12 12 0) "Silicon" "region_2"))
(sdegeo:bool-subtract (list b2 b1))
```

Appendix A: Commands

sdegeo:bool-unite

sdegeo:bool-unite

This Scheme extension performs a Boolean unite operation on the specified body list. The newly *united* body inherits the DATEX material attribute of the first body from the specified list of bodies. Similarly, the `region-name` attribute is inherited from the first entity in the body list.

This Scheme extension applies to both 2D and 3D bodies.

If the specified body list contains empty bodies, then the Scheme extension returns #f.

If the result of the Boolean operation is an empty body, then the empty body is removed from the entity list, and #t is returned.

Syntax

```
(sdegeo:bool-unite body-list)
```

Returns

ENTITY | BOOLEAN

Arguments

Argument	Description
body-list	Specifies a list of bodies. Argument type: BODY LIST

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position -10 -8 0) (position -5 -3 0)
    "Silicon" "region_1")
(sdegeo:create-rectangle (position -6 -7 0) (position 1 -4 0)
    "Silicon" "region_2")
(sdegeo:create-rectangle (position -1 -9 0) (position 3 -3 0)
    "PolySilicon" "region_3")
(sdegeo:create-polygon (list (position 1 -7 0) (position 5 -9 0)
    (position 10 -9 0) (position 11 -4 0) (position 3 -1 0)
    (position 1 -7 0)) "Silver" "region_4")
(sdegeo:bool-unite (get-body-list))
```

Appendix A: Commands

sdegeo:break-nearly-axis-aligned-edges

sdegeo:break-nearly-axis-aligned-edges

This Scheme extension changes nearly axis-aligned edges to axis-aligned edges by inserting additional vertices and staircase edges to the model. This step might be needed before meshing because some meshing algorithms (typically, quadtree-based meshing engines) might have difficulties handling nearly axis-aligned edges.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:break-nearly-axis-aligned-edges edge-list [angular-tolerance])
```

Returns

None

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST
angular-tolerance	Optional. Specifies the minimal angle, in degrees, that is permitted between model edges and between the x-axis and y-axis of the model. If the angle between the edge and the horizontal and vertical direction is smaller than angular-tolerance, then the edge is split into horizontal and vertical components. If this argument is not specified, then the default (5°) is used. Argument type: REAL

Appendix A: Commands

sdegeo:chamfer

sdegeo:chamfer

This Scheme extension performs edge or vertex chamfering operations for 3D bodies.

Syntax

```
(sdegeo:chamfer edge-list | vertex-list  
    chamfer-dist [adaptive-chamfering])
```

Returns

None

Arguments

Argument	Description
edge-list vertex-list	Specifies either a list of edges or a list of vertices to be chamfered. Argument type: EDGE LIST VERTEX LIST
chamfer-dist	Specifies the chamfering distance used for the operation. Argument type: REAL
adaptive-chamfering	Optional. Specifies whether to perform adaptive chamfering. If you set this argument to #t, then if the chamfering operation fails using the original chamfer-dist value, then the operation is repeated with a sequence of chamfering operations, using an adaptive approach, to set the chamfer distance to smaller values until the operation succeeds. Argument type: BOOLEAN

Description

You can use the GUI to select the edges to be operated on, or some other method can be used to find the edge entity IDs. For example, you can use `find-edge-id` to find the edge IDs.

Chamfering is a complex operation and can fail for several reasons. A common issue is using an incorrect chamfer distance, which is usually too large. The chamfer distance must be selected so that the resulting model (after performing the chamfering operation) is physically (topologically) correct. Another typical issue is the incorrect selection of the edges to be chamfered. The edge or vertex list must contain all tangent continuous edges; otherwise, the operation might fail.

Chamfering changes the model topology; new faces (and edges) are created, and old faces (edges) are removed from the model. When chamfering is performed in several steps, the order of performing the `sdegeo:chamfer` operation can be important.

Appendix A: Commands

sdegeo:chamfer

If a vertex list is specified, then all the edges that are connected in the specified vertices are chamfered.

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10) "Silver"
  "region_1")
(define edge1 (find-edge-id (position 5 0 0)))
(sdegeo:chamfer edge1 2)
(define facel (find-face-id (position 5 5 10)))
(sdegeo:chamfer (entity:edges facel) 1)
(define facel (find-face-id (position 5 0 5)))
(sdegeo:chamfer (entity:edges facel) 3)
```

Appendix A: Commands

sdegeo:chamfer-2d

sdegeo:chamfer-2d

This Scheme extension performs a 2D chamfering operation on a specified vertex or a list of vertices. When you use the GUI, this operation is applied to the selected entities. The Scheme extension (`sde:selected-entities`) returns the selected entity list. (Choose **Edit > Edit 2D > Chamfer.**)

Note:

When the chamfering is to be performed on a vertex (or vertices) shared by neighboring regions, all vertices must be added to the argument list.

Syntax

```
(sdegeo:chamfer-2d vertex chamfer-dist)
```

Returns

None

Arguments

Argument	Description
vertex	Specifies a vertex or a list of vertices to be chamfered. Argument type: VERTEX VERTEX LIST
chamfer-dist	Specifies the chamfering distance used for the operation. Note: The chamfering distance must be selected so that the operation does not result in an invalid geometry. Argument type: REAL

Examples

```
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "region_1")
(define mybody (car (reverse (part:entities))))
define myvertices (entity:vertices mybody))
sdegeo:chamfer-2d (list (list-ref myvertices 0)
    (list-ref myvertices 1)) 3)
```

Appendix A: Commands

sdegeo:check-overlap

sdegeo:check-overlap

This Scheme extension checks for possible overlaps between geometric bodies.

If the body list is specified by using the (`get-body-list`) command, then the entire model is checked for overlaps. The default overlap check looks only for a possible bounding box overlap. This operation is very fast but can report false overlaps. To provide a more reliable result, set `use-bbox-check` to `#f`.

If no overlap is found, then the Scheme extension returns null (an empty list). If overlaps are found, then it returns a list with all overlapping body pairs. If overlaps are found, these overlaps must be removed manually before the tessellated boundary output is generated; otherwise, meshing will not work correctly.

Note:

This Scheme extension applies to both 2D and 3D models.

Syntax

```
(sdegeo:check-overlap body-list [use-bbox-check])
```

Returns

BODY LIST

Arguments

Argument	Description
body-list	Specifies a list of bodies. Note: This list should contain only geometric bodies. DRS entities (Ref/Eval windows) are ignored, since Ref/Eval windows can overlap. Existing overlaps are not reported for these entities. Argument type: BODY LIST
use-bbox-check	Optional. Specify <code>#f</code> for this argument to provide a more reliable result. The default is <code>#t</code> . Argument type: BOOLEAN

Appendix A: Commands

sdegeo:chop-domain

sdegeo:chop-domain

This Scheme extension performs a 3D domain boundary cut.

If the base of the device is placed at the xy plane, then the device can be cut off to cover only the specified polygonal base.

Note:

The device must be placed at the -y-plane, and the cut is performed in the z-direction.

Syntax

```
(sdegeo:chop-domain point-pairs)
```

Returns

None

Arguments

Argument	Description
point-pairs	Specifies a list of x,y point pairs. Argument type: REAL REAL

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 5) (position 10 10 10) "Silicon"
  "x1")
(sdegeo:create-cuboid (position 0 0 10) (position 10 10 11)
  "PolySilicon" "x2")
(sdegeo:create-cuboid (position 2 2 11) (position 8 8 14) "Copper"
  "x3")
(sdegeo:create-cylinder (position 5 5 14) (position 5 5 18) 3 "Silver"
  "x4")
(sdegeo:chop-domain (list 0 0 2 2 6 0 10 4 3 10 0 4))
```

Appendix A: Commands

sdegeo:chull2d

sdegeo:chull2d

This Scheme extension computes the 2D convex hull of the specified points (position list). The 2D convex hull is returned in a POSITION LIST.

Note:

The input position list must be defined at the xy plane, and the z-coordinate must be 0.

Syntax

```
(sdegeo:chull2d position-list)
```

Returns

POSITION LIST

Arguments

Argument	Description
position-list	Argument type: POSITION LIST

Examples

```
(sde:clear)
(define pl1 (list
  (position 8 4 0) (position 0 0 0) (position 5 -4 0)
  (position 7 -1 0) (position 10 0 0) (position 10 5 0)
  (position 2 1 0) (position 4 1 0) (position 4 3 0)
  (position 8 3 0) (position 9 4 0) (position 9 3 0)
  (position 8.5 3.5 0) (position 2 3 0) (position 0 5 0)
  (position 5 -2 0)))
(sdegeo:create-polygon (sdegeo:chull2d pl1) "Silicon" "xx")
```

sdegeo:contact-sets

This Scheme extension returns the names of all the defined contact sets.

Syntax

```
(sdegeo:contact-sets)
```

Returns

A Scheme list containing the names of the defined contact sets

Appendix A: Commands

sdegeo:create-circle

sdegeo:create-circle

This Scheme extension adds a 2D circular region to the model. The circle is defined by specifying the center position and the radius. If the generated polygon overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted region and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

If you specify `start-angle` and `end-angle`, then a circular arch is generated.

Syntax

```
(sdegeo:create-circle center-pos radius region-material region-name  
[start-angle end-angle])
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
center-pos	Argument type: POSITION
radius	Argument type: REAL
region-material	Specifies the name of a material. Argument type: DATEXMAT
region-name	Specifies the name of a region. Argument type: STRING
start-angle	Optional. Argument type: REAL
end-angle	Optional. Argument type: REAL

Examples

```
(sdegeo:create-circle (position 0 0 0) 10 "Silicon" "Region_1")
```

Appendix A: Commands

sdegeo:create-cone

sdegeo:create-cone

This Scheme extension creates a circular cone.

Syntax

```
(sdegeo:create-cone {{start-axis end-axis} | {start-axis-x  
start-axis-y start-axis-z end-axis-x end-axis-y end-axis-z}}  
base-radius top-radius [ratio=1 [position3 | {x3 y3 z3}]]  
region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
start-axis	Specifies the start axis position or base of the cone. See <i>Description</i> . Argument type: POSITION
end-axis	Specifies the end axis position or top of the cone. See <i>Description</i> . Argument type: POSITION
start-axis-x	Argument type: REAL
start-axis-y	Argument type: REAL
start-axis-z	Argument type: REAL
end-axis-x	Argument type: REAL
end-axis-y	Argument type: REAL
end-axis-z	Argument type: REAL
base-radius	Specifies the radius at start-axis (must be greater than zero). Argument type: REAL
top-radius	Specifies radius at end-axis, which can be zero. Argument type: REAL
ratio	Optional. Argument type: REAL
position3	Optional. Argument type: POSITION
x3 y3 z3	Optional. Argument type: REAL

Appendix A: Commands

sdegeo:create-cone

Argument	Description
region-material	Argument type: DATEXMAT
region-name	Argument type: STRING

Description

Two syntax formats are available for defining the `start-axis` and `end-axis` positional arguments:

- The first (original) syntax format defines all positional arguments by placing them in ‘position’ statements enclosed in parentheses.
- The second syntax format defines positional arguments without using the ‘position’ statement or the additional set of parentheses.

Otherwise, the two formats are identical. The format selected must be used for all three positional arguments.

If you specify `ratio` and `position3`, an elliptical cone is created. If you specify `ratio`, the ratio between the major and minor axes of the ellipse is used.

If `position3` (or `x3 y3 z3`) is specified, then the vector from the projection of `position3` (or `x3 y3 z3`) onto the axis of the cone to `position3` (or `x3 y3 z3`) specifies the major axis. If `position3` (or `x3 y3 z3`) is not specified, then the major axis is defined by the projection of the `x-axis` of the active coordinate system onto the plane that is defined by `start-axis` and the vector from `start-axis` to `end-axis`.

Note:

The argument `position3` cannot lie on the axis of the cone; otherwise, an error occurs.

If the generated body overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The `material` attribute is set to `region-material` and the `region` attribute is set to `region-name`.

Examples

```
(sdegeo:create-cone (position -20 -5 -9) (position 15 20 10) 10 2  
    "Gold" "")  
(sdegeo:create-cone -20 -5 -9 5 15 7.5 10 2 3 "Silver" "")  
(sdegeo:create-cone -2 -5 -9 15 20 10 10 2 3 17 22 12 "Copper" "")
```

Appendix A: Commands

sdegeo:create-cuboid

sdegeo:create-cuboid

This Scheme extension creates a solid block. The block is oriented with respect to the current work plane.

If the generated body overlaps already existing regions, then the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-cuboid diagonal1 diagonal2 region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
diagonal1	Specifies the first diagonal corner of the block. Argument type: POSITION
diagonal2	Specifies the second diagonal corner of the block. Argument type: POSITION
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Examples

```
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1)
(sde:get-default-material) "r1")
```

Appendix A: Commands

sdegeo:create-cylinder

sdegeo:create-cylinder

This Scheme extension creates a cylinder.

Syntax

```
(sdegeo:create-cylinder {{start-pos end-pos} |
  {x-start y-start z-start x-end y-end z-end}} radius
  [ratio=1 {position3 | {x3 y3 z3}}] region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
start-pos	Specifies the start position of the cylinder. Argument type: POSITION
end-pos	Specifies the end position of the cylinder. Argument type: POSITION
x-start y-start z-start	Specify the start position of the cylinder. Argument type: REAL
x-end y-end z-end	Specify the end position of the cylinder. Argument type: REAL
radius	Specifies the radii for the base and top. Argument type: REAL
ratio	Optional. Argument type: REAL
position3	Optional. Argument type: POSITION
x3 y3 z3	Optional. Argument type: REAL
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Description

Two syntax formats are available for defining the start and end positions for creating a cylinder:

- The first (original) syntax format defines the positional arguments by placing them in ‘position’ statements enclosed in parentheses.

Appendix A: Commands

sdegeo:create-cylinder

- The second syntax format defines the positional arguments without using the ‘position’ statement or the additional set of parentheses.

Otherwise, the two formats are identical. The format selected must be used for all three positional arguments.

If you specify `ratio` and `position3` (or `x3 y3 z3`), an elliptical cylinder is created. If you specify `ratio`, the ratio between the major and minor axes of the ellipse is used.

If `position3` (or `x3, y3, and z3`) is specified, the vector from the projection of `position3` (or `x3 y3 z3`) on to the axis of the cylinder to `position3` (or `x3 y3 z3`) specifies the major axis. If `position3` (or `x3 y3 z3`) is not specified, the major axis is defined by the projection of the `x-axis` of the active coordinate system onto the plane that is defined by `position1` and the vector from `position1` to `position2`.

Note:

The argument `position3` cannot lie on the axis of the cylinder; otherwise, an error occurs.

If the generated body overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The `material` attribute is set to `region-material` and the `region` attribute is set to `region-name`.

Examples

```
(sde:clear)
(sdegeo:create-cylinder (position 0 0 0) (position 25 25 0) 30
    "Gold" "")
(sdegeo:create-cylinder 2 2 2 -20 -20 0 15 3 "Gold" "")
(sdegeo:create-cylinder 2 2 2 -20 -20 0 15 3 -5 -5 0 "Gold" "")
```

Appendix A: Commands

sdegeo:create-ellipse

sdegeo:create-ellipse

This Scheme extension creates a 2D elliptical region (ellipse).

It generates the ellipse on the active work plane, using the active coordinate system. The default Boolean setting is respected by the Scheme extension.

Syntax

```
(sdegeo:create-ellipse position1 position2 ratio lmat lreg)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
position1	Specifies the center position of an ellipse. Argument type: POSITION
position2	Specifies the endpoint of the major axis. Argument type: POSITION
ratio	Specifies the ratio of the major and minor axes. The ratio value can be greater than 1, in which case, the minor axis becomes the major axis. Argument type: REAL
lmat	Specifies a DATEX material. Argument type: DATEXMAT
lreg	Specifies the name of the region. Argument type: STRING

Examples

```
; Scheme Extension
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-ellipse (position 0 0 0) (position 10 0 0) 0.5
    "PolySi" "x1")
(sdegeo:create-ellipse (position 0 0 0) (position 3 0 0) 3
    "Silver" "x2")
(sdegeo:create-ellipse (position 0 0 0) (position 8 8 0) 0.2
    "Oxide" "x3")
```

Appendix A: Commands

sdegeo:create-ellipsoid

sdegeo:create-ellipsoid

This Scheme extension creates a 3D ellipsoid.

It generates the ellipse on the active work plane, using the active coordinate system. The default Boolean setting is respected by the Scheme extension.

Syntax

```
(sdegeo:create-ellipsoid position1 position2 ratio lmat lreg)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
position1	Specifies the center position of an ellipse. Argument type: POSITION
position2	Specifies the endpoint of the major axis. Argument type: POSITION
ratio	Specifies the ratio of the major and minor axes. The ratio value can be greater than 1, in which case, the minor axis becomes the major axis. Argument type: REAL
lmat	Specifies a DATEX material. Argument type: DATEXMAT
lreg	Specifies the name of the region. Argument type: STRING

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-ellipsoid (position 0 0 0) (position 10 0 0) 0.5
    "PolySi" "x1")
(sdegeo:create-ellipsoid (position 0 0 0) (position 3 0 0) 3
    "Silver" "x2")
(sdegeo:create-ellipsoid (position 0 0 0) (position 8 8 0) 0.2
    "Oxide" "x3")
```

Appendix A: Commands

sdegeo:create-ellipsoid-d

sdegeo:create-ellipsoid-d

This Scheme extension creates a polyhedral ellipsoid with triangular faces.

It generates the triangulated ellipsoid on the active work plane, using the active coordinate system. The default Boolean setting is respected by the Scheme extension.

Syntax

```
(sdegeo:create-ellipsoid-d position1 position2 ratio lmat lreg  
    nseg1 nseg2)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
position1	Specifies the center position of the ellipse. Argument type: POSITION
position2	Specifies the endpoint of the major axis. Argument type: POSITION
ratio	Specifies the ratio of the major and minor axes. The ratio value can be greater than 1, in which case, the minor axis becomes the major axis. Argument type: REAL
lmat	Specifies a DATEX material. Argument type: DATEXMAT
lreg	Specifies the name of the region. Argument type: STRING
nseg1	Specifies the number of segments (planar faces) along the major axis. Argument type: INTEGER
nseg2	Specifies the number of segments about the major axis. Argument type: INTEGER

Examples

```
(sde:clear)  
(sdegeo:create-ellipsoid-d (position 0 0 0) (position 10 0 0) 0.25  
    "Silicon" "xx" 10 20)
```

Appendix A: Commands

sdegeo:create-linear-edge

sdegeo:create-linear-edge

This Scheme extension creates a linear edge.

Syntax

```
(sdegeo:create-linear-edge pos1 pos2)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
pos1 pos2	Specify the end positions of the line. Argument type: POSITION

sdegeo:create-ot-ellipsoid

This Scheme extension creates an oriented and truncated (ot) ellipsoid.

If the generated body overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-ot-ellipsoid center-position major-axis-position ratio  
truncate-angle1 truncate-angle2 region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
center-position	Specifies the center position of the ellipsoid. Argument type: POSITION
major-axis-position	Specifies the endpoint of the major axis of the untruncated ellipsoid. Argument type: POSITION
ratio	Specifies the ratio of the major and minor axes. Argument type: REAL

Appendix A: Commands

sdegeo:create-ot-ellipsoid

Argument	Description
truncate-angle1 truncate-angle2	Determine the cut-off angles where the ellipsoidal top and bottom are replaced by a flat top and bottom. Argument type: REAL
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Examples

```
(sde:clear)
(define mb
  (sdegeo:create-polygon (list (position 0 3 0) (position 1 3 0)
                                (position 2 2 0) (position 3 2 0) (position 4 3 0)
                                (position 5 3 0) (position 5 0 0) (position 0 0 0))
                            "Silicon" "xx"))
(sdegeo:extrude (list mb) 2)
(do ((i 0 (+ i 1))) ((>= i 5))
    (define mf (caar (sdegeo:ray-test mb (ray
                                         (position (+ i 0.5) 5 1.0) (gvector 0 -1 0)) 0.01)))
    (define mp (cdar (sdegeo:ray-test mb (ray
                                         (position (+ i 0.5) 5 1.0) (gvector 0 -1 0)) 0.01)))
    (define mn (face:plane-normal mf))
    (define sf 0.5)
    (define mp1 (position (+ (position:x mp) (* sf (gvector:x mn)))
                           (+ (position:y mp) (* sf (gvector:y mn))))
                           (+ (position:z mp) (* sf (gvector:z mn)))))
    (define sf 1.2)
    (define mp2 (position (+ (position:x mp) (* sf (gvector:x mn)))
                           (+ (position:y mp) (* sf (gvector:y mn))))
                           (+ (position:z mp) (* sf (gvector:z mn)))))
    (sdegeo:create-ot-ellipsoid mp1 mp2 0.5 20 80 "Gold" "xx")
)
```

Appendix A: Commands

sdegeo:create-ot-sphere

sdegeo:create-ot-sphere

This Scheme extension creates an oriented and truncated (ot) sphere.

If the generated body overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The material attribute is set to region-material and the region attribute is set to region-name.

Syntax

```
(sdegeo:create-ot-sphere center-position orientation radius
    truncate-angle1 truncate-angle2 region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
center-position	Specifies the center position of the sphere. Argument type: POSITION
orientation	Specifies the normal vector of the truncated top and bottom. The sphere is oriented in such a way that the poles of the sphere are placed in the orientation direction. Argument type: GVECTOR
radius	Specifies the size of the sphere. Argument type: REAL
truncate-angle1 truncate-angle2	Determine the cut-off angles where the spherical top and bottom are replaced by a flat top and bottom. Argument type: REAL
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Examples

```
(sde:clear)
(sdegeo:create-ot-sphere (position 0 0 0) (gvector 1 0 0) 1 65 65
    "Silicon" "x1")
(sdegeo:create-ot-sphere (position 0 0 3) (gvector 1 1 0) 1 50 50
    "PolySilicon" "x2")
(sdegeo:create-ot-sphere (position 0 0 6) (gvector 0 1 0) 1 40 40
    "Copper" "x3")
(sdegeo:create-ot-sphere (position 0 0 9) (gvector -1 1 0) 1 30 30
    "Oxide" "x4")
```

Appendix A: Commands

sdegeo:create-polygon

sdegeo:create-polygon

This Scheme extension adds a closed 2D polygonal region to the model.

The polygon is defined by specifying the vertex list of the polygon (either closed or not closed). If the last vertex differs from the first vertex, then the first vertex closes the polygon.

If the generated polygon overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted region and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-polygon vertex-list region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: POSITION LIST
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

sdegeo:create-polyline-wire

This Scheme extension defines a wire body consisting of linear edges only. The vertex positions of the wire are specified in the argument list as a position list. The wire body does not have an attached material or `region-name` attribute and is not written to the output boundary file. The wire body can be used, for example, during a sweep operation when the sweep along a wire option is used.

Syntax

```
(sdegeo:create-polyline-wire vertex-list)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description	Argument Type
vertex-list	Specifies a list of vertices. Argument type: POSITION LIST	

Appendix A: Commands

sdegeo:create-prism

sdegeo:create-prism

This Scheme extension creates an n -sided prism where n is greater than or equal to three.

If the generated body overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-prism center-position height major-radius minor-radius  
    nsides region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
center-position	Specifies the center position of the prism. Argument type: POSITION
height	Specifies the height of the prism along the z-axis. If set to zero, then the resulting body consists of only one polygonal-sided sheet face, lying in the xy plane. Argument type: POSITION
major-radius	Specifies the major radius along the x-axis of the prism. Argument type: REAL
minor-radius	Specifies the minor radius along the y-axis of the prism. Argument type: REAL
nsides	Specifies the number of sides of the prism. The number of sides must be greater than or equal to three. Argument type: INTEGER
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Examples

```
(sdegeo:create-prism (position 0 0 0) 20 40 40 6 "PolySilicon" "")
```

Appendix A: Commands

sdegeo:create-pyramid

sdegeo:create-pyramid

This Scheme extension creates a solid pyramid.

The pyramid is centered about the origin:

- If the up direction is set to "`-x`" (UCS), then its `height` is along the x-axis, the `major-radius` is along the z-axis, and the `minor-radius` is along the y-axis.
- If the up direction is set to "`+z`", then its `height` is along the z-axis, the `major-radius` is along the x-axis, and the `minor-radius` is along the y-axis.

If the generated body overlaps already existing regions, then the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The `material` attribute is set to `region-material` and the `region` attribute is set to `region-name`.

Syntax

```
(sdegeo:create-pyramid center-position height major-radius  
minor-radius nsides top region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
<code>center-position</code>	Specifies the center position of the pyramid. Argument type: POSITION
<code>height</code>	Specifies the height of the pyramid. If set to zero, then the resulting body consists of only one polygonal-sided sheet face, lying in the xy plane. Argument type: REAL
<code>major-radius</code>	Specifies the major radius of the pyramid. Argument type: REAL
<code>minor-radius</code>	Specifies the minor radius of the pyramid. Argument type: REAL
<code>nsides</code>	Specifies the number of sides of the pyramid. The number of sides must be greater than or equal to three. Argument type: INTEGER
<code>top</code>	Specifies the major axis length at the top of the pyramid. Argument type: REAL
<code>region-material</code>	Specifies the material of the region. Argument type: DATEXMAT
<code>region-name</code>	Specifies the name of the region. Argument type: STRING

Appendix A: Commands

sdegeo:create-pyramid

Examples

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-pyramid (position 0 0 0) 20 40 40 6 12
    "PolySilicon" "region_1")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-pyramid (position 0 0 0) 30 30 20 5 5
    "Gold" "region_2")
(sde:clear)
(sde:set-process-up-direction "-x")
(sdegeo:create-pyramid (position 0 0 0) 20 40 40 6 12
    "PolySilicon" "region_1")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-pyramid (position 0 0 0) 30 30 20 5 5
    "Gold" "region_2")
```

Appendix A: Commands

sdegeo:create-rectangle

sdegeo:create-rectangle

This Scheme extension adds a 2D rectangular region to the model. The rectangle is defined by specifying two opposite corner vertices.

If the generated rectangle overlaps with already existing regions, the default Boolean behavior determines the topology of the newly inserted region and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-rectangle v1 v2 material-name region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
v1 v2	Specifies the two opposite corner vertices of the rectangle. Argument type: POSITION
material-name	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Examples

```
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0))
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Silicon" "sir")
```

Appendix A: Commands

sdegeo:create-reg-polygon

sdegeo:create-reg-polygon

This Scheme extension adds a regular 2D polygonal region to the model. The regular polygon is defined by specifying the center position of the polygon, the radius, and the number of segments.

If the generated polygon overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted region and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-reg-polygon v1 rad  
[nsides] [angle] region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
v1	Specifies the center of the regular polygon. Argument type: POSITION
rad	Specifies the radius of the circle that defines the vertex points. Argument type: REAL
nsides	Optional. Specifies the number of sides of the polygonal region. Argument type: INTEGER
angle	Optional. Specifies the angle [degree], counterclockwise from the x-axis, which defines the 'rotation' of the regular polygon. Argument type: REAL
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Examples

```
(sdegeo:create-reg-polygon (position 0 0 0) 1 10 0 "Silicon"  
"region_1")
```

Appendix A: Commands

sdegeo:create-ruled-region

sdegeo:create-ruled-region

This Scheme extension adds a 2D ruled region to the model.

The ruled region is defined by specifying two existing edges. Edge orientation is not important, since the ruled region is always created in such a way that a simply connected region is created. The selected edges cannot intersect each other.

If the ruled region overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted region and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-ruled-region edge1 edge2 region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
edge1	Specifies one existing edge. Argument type: EDGE
edge2	Specifies a second existing edge. Argument type: EDGE
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Appendix A: Commands

sdegeo:create-sphere

sdegeo:create-sphere

This Scheme extension creates a sphere that is centered at a specified position.

Syntax

```
(sdegeo:create-sphere {center-position | {center-x center-y center-z}}
    radius region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
center-position	Specifies the center position of the sphere. Argument type: POSITION
center-x center-y center-z	Specify the center position of the sphere. Argument type: REAL
radius	Specifies the size of the sphere. Argument type: REAL
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Description

Two syntax formats are available for defining `center-position`:

- The first (original) syntax format defines the `center-position` by placing the xyz coordinates in a ‘position’ statement enclosed in parentheses (as shown in the example creating `sphere1`).
- The second syntax format defines the center position xyz coordinates without using the ‘position’ statement or the additional set of parentheses (as shown in the example defining `sphere2` and `sphere3`).

Otherwise, the two formats are identical.

If the generated body overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The `material` attribute is set to `region-material` and the `region` attribute is set to `region-name`.

Appendix A: Commands

sdegeo:create-spline-wire

Examples

```
(sde:clear)
(define spherel (sdegeo:create-sphere (position -4 -4 0) 1.5
                                         "Gold" ""))
(define sphere2 (sdegeo:create-sphere -30 0 0 15 "Silver" ""))
(define sphere3 (sdegeo:create-sphere 10 10 10 5 "Copper" ""))
```

sdegeo:create-spline-wire

This Scheme extension defines a spline wire body.

The vertex positions of the wire are specified in the argument list as a position list. The wire body does not have an attached material or region-name attribute, and is not written to the output boundary file. The wire body can be used, for example, during a sweep operation when the sweep along a wire option is used.

Syntax

```
(sdegeo:create-spline-wire vertex-list)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
vertex-list	Specifies a list of vertex positions. Argument type: POSITION LIST

Appendix A: Commands

sdegeo:create-torus

sdegeo:create-torus

This Scheme extension creates a solid torus of given radii centered at the origin.

Syntax

```
(sdegeo:create-torus {center-position | {center-x center-y center-z}}
                      major-radius minor-radius region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
center-position	Specifies the center location of the torus. Argument type: POSITION
center-x center-y center-z	Specify the center location of the torus. Argument type: REAL
major-radius	Specifies the distance from the center to the spine curve lying in this plane. It is specified around a circle having the minor axis and is swept to define the torus. Three shapes of tori can be specified (donut, apple, or lemon) depending on the relative magnitudes of the major and minor radii: <ul style="list-style-type: none">• If major-radius is greater than minor-radius, the torus is a donut.• If major-radius is positive but smaller than minor-radius, the torus is an apple.• If major-radius is negative, the torus is a lemon. Argument type: REAL
minor-radius	Specifies the minor radius. Argument type: REAL
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Description

The torus is defined in the xy plane of the active coordinate system and is oriented using the normal gvector of the active coordinate system.

Two syntax formats are available for the center position argument:

- The first (original) syntax format defines center-position by placing it in a 'position' statement enclosed in parentheses.

Appendix A: Commands

sdegeo:create-torus

- The second format defines the xyz coordinates of the center position without using the 'position' statement or the additional set of parentheses.

Otherwise, the two formats are identical.

If the generated body overlaps already existing regions, the default Boolean behavior determines the topology of the newly inserted body and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Examples

```
(sde:clear)
(sdegeo:create-torus (position -10 -5 -10) 7 3 "Gold" "")
(sdegeo:create-torus 10 15 20 10 5 "Gold" "")
```

Appendix A: Commands

sdegeo:create-triangle

sdegeo:create-triangle

This Scheme extension adds a 2D triangular region to the model. The triangle is defined by specifying three vertices.

If the generated triangle overlaps with existing regions, the default Boolean behavior determines the topology of the newly inserted region and the overlapping regions. The material attribute is set to `region-material` and the region attribute is set to `region-name`.

Syntax

```
(sdegeo:create-triangle v1 v2 v3 region-material region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
v1 v2 v3	Specify the three vertices that define the triangle. Argument type: POSITION
region-material	Specifies the material of the region. Argument type: DATEXMAT
region-name	Specifies the name of the region. Argument type: STRING

Examples

```
(sdegeo:create-triangle (position 0 0 0) (position 1 0 0)
                         (position 0.5 1 0) "Silicon" "sir")
```

Appendix A: Commands

sdegeo:curve-intersect

sdegeo:curve-intersect

This Scheme extension returns the intersection positions between the specified curves.

Syntax

```
(sdegeo:curve-intersect edge1 edge2)
```

Returns

POSITION LIST

Arguments

Argument	Description
edge1	Specifies one existing edge. Argument type: EDGE
edge2	Specifies a second existing edge. Argument type: EDGE

Examples

```
(define e1 (edge:linear (position 0 0 0) (position 10 10 0)))
(define e2 (edge:linear (position 0 10 0) (position 10 0 0)))
(sdegeo:curve-intersect e1 e2)
; (#[position 5 5 0])
(define e3 (edge:circular (position 5 5 0) 3))
(sdegeo:curve-intersect e1 e3)
; (#[position 7.12132034355964 7.12132034355964 0]
#[position 2.87867965644036 2.87867965644036 0])
```

sdegeo:define-3d-contact-by-polygon

This Scheme extension defines a 3D face contact by a polygon, that is, it imprints a general polygon wire to an existing body. The imprinted wire splits the faces of the body.

This Scheme extension can be used to define a polygonal contact area. The imprint functions split faces to surface patches. After the face split, the newly created face segments to be marked as contacts must be identified. During the assignment of contacts, these face ID numbers can be used explicitly.

Syntax

```
(sdegeo:define-3d-contact-by-polygon pos-list dir-vector contact-name)
```

Returns

None

Arguments

Argument	Description
pos-list	Specifies a list of the vertices of the imprinted polygon. The polygon must be closed to imprint it, so the first position must be repeated as the last position. Argument type: POSITION LIST
dir-vector	Specifies the direction in which the polygon is projected for the imprint step. Argument type: GVECTOR
contact-name	Specifies the name of the contact set. Argument type: STRING

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10) "Silicon"
    "region_1")
(sdegeo:define-contact-set "demo" 4 (color:rgb 1 0 0) "##")
(sdegeo:set-current-contact-set "demo")
(sdegeo:define-3d-contact-by-polygon (list (position 1 1 10)
    (position 5 1 10) (position 5 5 10) (position 3 2 10)
    (position 1 5 10) (position 1 1 10))
    (gvector 0 0 -1) "demo")
```

Appendix A: Commands

sdegeo:define-contact-set

sdegeo:define-contact-set

This Scheme extension defines a contact set name and initializes the supporting data.

For edge contacts (2D), the `edgeThickness` and `color` arguments are used. For face contacts (3D), the `color` and `facePattern` arguments are used. (The argument that is not used is ignored if it is specified in the argument list.)

You can use the **Contact Sets** button of the GUI to display the corresponding Contact Set dialog box, which can be used to define the arguments interactively. (Choose **Device > Contacts > Contact Sets**.)

Note:

The optional arguments can be given in any order.

Syntax

```
(sdegeo:define-contact-set name  
    [edgeThickness color] | [color facePattern])
```

Returns

Unspecified

Arguments

Argument	DescriptionArgument Type
name	Specifies the name of a contact set. If the contact set name is already defined, then the existing contact set is replaced by the new definition. Argument type: STRING
edgeThickness	Optional. Specifies the thickness of the edge. The default is 4.0. Argument type: INTEGER
color	Optional. Specifies an RGB color object, for example, <code>(color:rgb 0 1 0)</code> . The <code>color:rgb</code> command takes three real-number arguments, each in the range $[0,1]$. The default is <code>(color:rgb 1.0 0.0 0.0)</code> . Argument type: COLOR
facePattern	Optional. Specifies the pattern to be used for the face. Options are "solid" "##" " " "==" "//" ":" "<><>" "[] []". The default is "##". Argument type: STRING

Examples

```
(sdegeo:define-contact-set "Drain")  
(sdegeo:define-contact-set "Source" 4.0 (color:rgb 1.0 0.0 0.0) "##")
```

Appendix A: Commands

sdegeo:define-coord-sys

sdegeo:define-coord-sys

This Scheme extension creates a new coordinate system in the active work plane.

If the active work plane is called `awp`, then the newly created coordinate system is called `awp_cs-name`. The origin of the new coordinate system is based at `x-pos` and `y-pos`, relative to the origin of the active work plane.

The coordinate system is rotated by `angle`. Counterclockwise rotation is positive (looking from the positive z-axis of the active work plane).

Syntax

```
(sdegeo:define-coord-sys cs-name x-pos y-pos angle)
```

Returns

BOOLEAN

Arguments

Argument	Description
cs-name	Specifies the name of the coordinate system. Argument type: STRING
x-pos	Argument type: REAL
y-pos	Argument type: REAL
angle	Specifies the angle (in degrees) by which to rotate the coordinate system. Argument type: REAL

Appendix A: Commands

sdegeo:define-work-plane

sdegeo:define-work-plane

This Scheme extension creates a new work plane.

The x-axis points towards *x-axis* and the y-axis points towards *y-axis*. The z-axis is defined by the right-hand rule. The origin must be considered when you define the x-axis and y-axis.

Syntax

```
(sdegeo:define-work-plane wp-name origin x-axis y-axis)
```

Returns

BOOLEAN

Arguments

Argument	Description
wp-name	Specifies the name of the work plane. Argument type: STRING
origin	Specifies where to place the origin of the work plane. Argument type: POSITION
x-axis	Specifies the direction of the x-axis. Argument type: POSITION
y-axis	Specifies the direction of the y-axis. Argument type: POSITION

Examples

Define a new work plane, *wp1*, parallel to the global xy plane, located at *z* = 1:

```
(sdegeo:define-work-plane "wp1" (position 0 0 1) (position 1 0 1)
                           (position 0 1 1))
```

Appendix A: Commands

sdegeo:del-short-edges

sdegeo:del-short-edges

This Scheme extension deletes linear edges if the edge length is shorter than the specified edge-length.

Ridges can be protected by an angular tolerance. If the vertex normal difference is larger than angular-tolerance at a given vertex location, then the vertex (and adjoining edges) are protected. In this case, even if the edge length is smaller than edge-length, the edge is not removed.

This Scheme extension is fast, very robust, and recommended to simplify 2D geometry.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:del-short-edges entity edge-length angular-tolerance)
```

Returns

None

Arguments

Argument	Description
entity	Specifies a list of entities. Argument type: ENTITY LIST
edge-length	Specifies the edge length that determines whether linear edges are deleted. Argument type: REAL
angular-tolerance	Specifies the angular tolerance. Argument type: REAL

Appendix A: Commands

sdegeo:delete-collinear-edges

sdegeo:delete-collinear-edges

This Scheme extension removes collinear edges from the specified edge list by merging the neighboring collinear edges into one single-edge entity.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:delete-collinear-edges edge-list)
```

Returns

None

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST

sdegeo:delete-contact-boundary-edges

This Scheme extension deletes the current contact set attributes from all edges of the selected body or bodies. Alternatively, a body or body list can be given in the argument list. Only the active contacts are removed.

Syntax

```
(sdegeo:delete-contact-boundary-edges [body | body-list])
```

Returns

None

Arguments

Argument	Description
body body-list	Optional. Specifies a body or a list of bodies. Argument type: BODY BODY LIST

Appendix A: Commands

sdegeo:delete-contact-boundary-faces

sdegeo:delete-contact-boundary-faces

This Scheme extension removes a contact attribute from all faces of each specified body in `body-list`. If the argument list is empty and `body-list` is not defined, then the Scheme extension takes the body or bodies from (`sde:selected-entities`). The active contact set name is taken to specify the contacts. Alternatively, to the selected faces, a face list can be given explicitly in the argument list.

Syntax

```
(sdegeo:delete-contact-boundary-faces [body | body-list])
```

Returns

None

Arguments

Argument	Description	Argument Type
body body-list	Optional. Specifies a body or a list of bodies. Argument type: BODY BODY LIST	

Examples

```
(sde:clear)
(sdegeo:define-contact-set "xx" 4.0 (color:rgb 1 0 0) "##")
(sdegeo:set-current-contact-set "xx")
(define mycube (sdegeo:create-cuboid (position 0 0 0) (position 1 1 1)
                                         "PolySilicon" "region_1"))
(sdegeo:set-contact-boundary-faces mycube)
(sdegeo:delete-contact-boundary-faces mycube)
```

sdegeo:delete-contact-edges

This Scheme extension deletes the current contact set attributes from the selected edges. Alternatively, an edge or edge list can be given in the argument list. Only the active contacts are removed. (Choose **Device > Contacts > Unset Edges**.)

Syntax

```
(sdegeo:delete-contact-edges [edge | edge-list])
```

Returns

None

Arguments

Argument	Description
edge edge-list	Optional. Specifies an edge or a list of edges. Argument type: EDGE EDGE LIST

sdegeo:delete-contact-faces

This Scheme extension removes the contact attribute from the specified faces.

If the argument list is empty and face-list is not defined, then the Scheme extension takes the faces from (sde:selected-entities). The name of the active contact set is taken to specify the contacts. Alternatively, to the selected faces, a face list can be given explicitly in the argument list.

Syntax

```
(sdegeo:delete-contact-faces [face | face-list])
```

Returns

None

Arguments

Argument	Description
face face-list	Optional. Specifies a face or a list of faces. Argument type: FACE FACE LIST

Appendix A: Commands

sdegeo:delete-contact-set

sdegeo:delete-contact-set

This Scheme extension deletes the specified contact set and removes the corresponding contact attributes (either '2D contact' or '3D contact').

Syntax

```
(sdegeo:delete-contact-set) contact-name
```

Returns

None

Arguments

Argument	Description
contact-name	Specifies the name of a contact set. Argument type: STRING

sdegeo:delete-coord-sys

This Scheme extension deletes the specified coordinate system.

Syntax

```
(sdegeo:delete-coord-sys cs-name)
```

Returns

BOOLEAN

Arguments

Argument	Description
cs-name	Specifies the name of a coordinate system. Argument type: STRING

Appendix A: Commands

sdegeo:delete-edges

sdegeo:delete-edges

This Scheme extension removes the specified edges from the model. If there are neighboring regions, edges that are shared by more than one body must be selected from all bodies. Otherwise, model conformity is not preserved.

By default, a 90° angular tolerance is used to protect model features (that is, only those edges from the specified entity list that enclose an angle greater than 90° are removed from the edge list).

You can use the optional argument `angular-tolerance` to modify the default setting.

This Scheme extension appends the input edge list with all the matching neighbor edges to provide a conformal result (no gaps or overlaps are created by `sdegeo:delete-edges` between the originally matching neighbor bodies).

Note:

This Scheme extension applies to 2D models only. Only linear edges can be deleted.

Syntax

```
(sdegeo:delete-edges edge-list [angular-tolerance])
```

Returns

None

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST
angular-tolerance	Optional. Specifies the angular tolerance in degrees. The default is 90°. Argument type: REAL

sdegeo:delete-nearly-collinear-edges

This Scheme extension deletes nearly collinear edges from the specified edge list.

Nearly collinear edges are defined as follows: If the distance from the point between the two edges is closer than the specified distance (*dist*) to the edge that is defined by the other two noncommon endpoints of the two edges, then they are classified as nearly collinear. All nearly collinear edges are removed from the edge list.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:delete-nearly-collinear-edges edge-list dist)
```

Returns

None

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST
dist	Specifies a distance to the edge. Argument type: EDGE

Appendix A: Commands

sdegeo:delete-region

sdegeo:delete-region

This Scheme extension deletes the specified region or regions, including Ref/Eval window bodies. The body is specified by its entity number. The specified regions are permanently removed from the model.

Syntax

```
(sdegeo:delete-region bodies)
```

Returns

None

Arguments

Argument	Description
bodies	Specifies a region or a list of regions. Argument type: BODY BODY LIST

sdegeo:delete-short-edges

This Scheme extension removes edges from the specified list of edges. An edge is deleted from the model only if the length of the edge is shorter than the specified edge length, max-edge-length. If there are neighboring regions, edges that are shared by more than one body must be selected from all bodies. Otherwise, model conformity is not preserved.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:delete-short-edges edge-list max-edge-length)
```

Returns

None

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST
max-edge-length	Specifies the maximum edge length. Argument type: REAL

Appendix A: Commands

sdegeo:delete-vertices

sdegeo:delete-vertices

This Scheme extension removes the specified vertices from the model. If there are neighboring regions, vertices that are located at the same position and are shared by more than one body must be selected from all bodies. Otherwise, model conformity is not preserved.

Note:

This Scheme extension applies to 2D models only. Only vertices that are shared by linear edges can be deleted.

Syntax

```
(sdegeo:delete-vertices vertex-list)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: VERTEX_LIST

sdegeo:delete-work-plane

This Scheme extension deletes the specified work plane.

Note:

The global work plane 'base' is created automatically and cannot be deleted.

Syntax

```
(sdegeo:delete-work-plane wp-name)
```

Returns

BOOLEAN

Arguments

Argument	Description
wp-name	Specifies the name of a work plane. Argument type: STRING

Appendix A: Commands

sdegeo:distance

sdegeo:distance

This Scheme extension returns the distance between two vertices.

Syntax

```
(sdegeo:distance vertex-list)
```

Returns

REAL

Arguments

Argument	Description
vertex-list	Specifies two vertices. Argument type: VERTEX LIST

sdegeo:dnce

This Scheme extension removes nearly collinear edges from a 2D model. The specified angular tolerance determines which linear edges are removed.

If the edge normals differ less than the angular tolerance (in degrees) at a given vertex, then the two neighboring edges are deleted and they are replaced by a linear edge connecting the other two vertices of the two original edges that shared the given vertex.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:dnce angular-tolerance)
```

Returns

None

Arguments

Argument	Description
angular-tolerance	Specifies the angular tolerance. Argument type: REAL

Appendix A: Commands

sdegeo:extend-device

sdegeo:extend-device

This Scheme extension extends a device around its axis-aligned perimeter.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:extend-device {"right" extension} {"left" extension}
                      {"top" extension} {"bottom" extension})
```

Returns

BOOLEAN

Arguments

Argument	Description
"right" extension	Specifies to extend a device in the right direction by a specified extension distance. Argument type: REAL
"left" extension	Specifies to extend a device in the left direction by a specified extension distance. Argument type: REAL
"top" extension	Specifies to extend a device in the top direction by a specified extension distance. Argument type: REAL
"bottom" extension	Specifies to extend a device in the bottom direction by a specified extension distance. Argument type: REAL

Description

The keywords specify the direction of the extension. At least one keyword must be used in the argument list. More than one keyword can be used together with the actual extension distance.

If an extension is defined around a corner (for example, both "top" and "right" are specified), then the rounded corner is filled as well. The region names of the extended strips are derived from the original region names. The extended sides must be axis aligned.

The sdegeo:extend-device command takes the process *up* direction into consideration. If the process up direction is set to "-x":

- "left" is interpreted as "bottom".
- "right" is interpreted as "top".

Appendix A: Commands

sdegeo:extend-device

- "top" is interpreted as "left".
- "bottom" is interpreted as "right".

Examples

```
(sde:clear)
(degeo:create-rectangle (position 0 0 0) (position 1 0.2 0)
    "Silicon" "xx1")
(sdegeo:create-rectangle (position 0 0.2 0) (position 1 0.3 0)
    "Copper" "xx2")
(sdegeo:create-rectangle (position 0 0.3 0) (position 1 0.6 0)
    "Silver" "xx3")
(sdegeo:create-rectangle (position 0 0.6 0) (position 1 0.7 0)
    "PolySi" "xx4")
(sdegeo:extend-device "right" 0.3 "top" 0.2)
```

Appendix A: Commands

sdegeo:extrude

sdegeo:extrude

This Scheme extension extrudes a 2D device to a 3D model. It takes a list of 2D bodies (specified by `body-list`) and extrudes them in the positive z-direction by `extrusion-depth`. Instead of `body-list`, you can use the keyword `all`, in which case, the complete model is extruded.

This Scheme extension also applies to Ref/Eval windows. In the case of a submesh reference, information regarding the extruded Ref/Eval window before the extrusion and the `extrusion-depth` are stored in the command file, so that the meshing engine can reproduce doping information in the 3D extruded profile. For other doping profile types, the extruded Ref/Eval window is stored directly in the command file without extrusion information.

Syntax

```
(sdegeo:extrude body-list extrusion-depth [extrude-option])
```

Returns

None

Arguments

Argument	Description
body-list	Specifies a list of 2D bodies. Argument type: BODY LIST
extrusion-depth	Specifies the extrusion depth in the positive z-direction. Argument type: REAL
extrude-option	Optional. Specifies an extrusion option that is used to filter the input <code>body-list</code> . Possible values are "2D" "3D" "Mixed": <ul style="list-style-type: none">If set to "2D" (default), then only 2D entities (sheet bodies) from the input <code>body-list</code> are extruded.If set to "3D", then only the faces of 3D entities (solid bodies with a positive volume) from the input <code>body-list</code> are extruded.If set to "Mixed", then both 2D and 3D entities are extruded (the entire <code>body-list</code>).

Examples

```
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "region_1")
(sdegeo:create-rectangle (position 0 10 0) (position 10 10.1 0)
    "Oxide" "region_2")
(sdegeo:create-rectangle (position 4 10.1 0) (position 6 14 0)
    "PolySilicon" "region_3")
(sdegeo:extrude "all" 4)
```

Appendix A: Commands

sdegeo:face-find-interior-point

sdegeo:face-find-interior-point

This Scheme extension takes a face Scheme object and returns a position object for an interior point of that face.

Syntax

```
(sdegeo:face-find-interior-point position)
```

Returns

```
ENTITY (FACE)
```

Arguments

Argument	Description
position	Specifies a face Scheme object. Argument type: POSITION

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silicon" "xx")
(define myface (car (find-face-id (position 5 5 10))))
(sdegeo:face-find-interior-point myface)
;; #[ position 5 5 10 ]

(sde:clear)
(define mybody (sdegeo:create-polygon (list (position 0 0 0)
  (position 10 0 0) (position 10 5 0) (position 9 5 0)
  (position 9 1 0) (position 1 1 0) (position 1 5 0)
  (position 0 5 0) (position 0 0 0)) "Silicon" "xx"))
(define myface (car (entity:faces mybody)))
(sdegeo:face-find-interior-point myface)
;; #[ position 5 0.5 0 ]
```

Appendix A: Commands

sdegeo:fillet

sdegeo:fillet

This Scheme extension performs edge or vertex rounding operations on 3D bodies.

Syntax

```
(sdegeo:fillet edge-list | vertex-list  
    fillet-radius [adaptive-filletting])
```

Returns

None

Arguments

Argument	Description
edge-list vertex-list	Specifies either a list of edges or a list of vertices to be rounded. Argument type: EDGE LIST VERTEX LIST
fillet-radius	Specifies the rounding radius to be used for the operation. Argument type: REAL
adaptive-filletting	Optional. Specifies whether to perform adaptive filleting. If you set the argument to #t, then if the filleting operation fails using the original fillet-radius value, the operation is repeated with a sequence of filleting operations, using an adaptive approach, to set the fillet radius to smaller values until the operation succeeds. Argument type: BOOLEAN

Description

You can use the GUI to select the edges to be rounded or you can use other methods to find the edge entity IDs. For example, `find-edge-id` can be used to find the edge IDs.

Rounding is a complex operation and can fail for several reasons. A common problem is the use of an incorrect `fillet-radius`, which is usually too large. The `fillet-radius` must be selected so that the resulting model (after performing the filleting operation) is physically (topologically) correct. Another typical problem is the incorrect selection of the edges to be filleted. The edge or vertex list must contain all tangent continuous edges if an edge list is specified; otherwise, rounding fails. Rounding changes model topology; new faces (and edges) are created and old faces (edges) are removed from the model. When rounding is performed in several steps, the order of performing the `sdegeo:fillet` operation can be important. The rounding uses a constant-radius rounding for all edges in the edge or vertex list.

Appendix A: Commands

sdegeo:fillet

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silver" "region_1")
(define edge1 (find-edge-id (position 5 0 0)))
(sdegeo:fillet edge1 2)
(define face1 (find-face-id (position 5 5 10)))
(sdegeo:fillet (entity:edges face1) 1)
(define face1 (find-face-id (position 5 0 5)))
(sdegeo:fillet (entity:edges face1) 3)
```

Appendix A: Commands

sdegeo:fillet-2d

sdegeo:fillet-2d

This Scheme extension rounds the specified vertex or list of vertices.

When you use the GUI, this operation is applied to the selected entities. The (sde:selected-entities) Scheme extension returns the selected entity list. (Choose **Edit > Edit 2D > Fillet.**)

Note:

When the rounding should be performed on a vertex (or vertices) shared by neighboring regions, all vertices must be added to the argument list.

The fillet-radius must be selected so that the operation does not result in an invalid geometry. Small roundings can result in an excessive number of small elements during meshing.

Syntax

```
(sdegeo:fillet-2d vertex fillet-radius)
```

Returns

None

Arguments

Argument	Description
vertex	Specifies either a vertex or a list of vertices. Argument type: ACIS VERTEX ACIS VERTEX LIST
fillet-radius	Specifies the rounding radius to be used for the operation. Argument type: REAL

Examples

```
(define mybody (sdegeo:create-rectangle (position 0 0 0)
                                         (position 10 10 0) "Silicon" "region_1"))
(define myvertices (entity:vertices mybody))
(sdegeo:fillet-2d (list (list-ref myvertices 0)
                        (list-ref myvertices 1)) 3)
```

Appendix A: Commands

sdegeo:find-closest-edge

sdegeo:find-closest-edge

This Scheme extension goes through the edge list of all defined geometric bodies and returns the entity ID of the closest edge to the specified position.

Syntax

```
(sdegeo:find-closest-edge position)
```

Returns

```
(EDGE . REAL)
```

Arguments

Argument	Description
position	Argument type: POSITION

sdegeo:find-closest-face

This Scheme extension goes through the face list of all defined geometric bodies and returns the entity ID of the closest face to the specified position.

Syntax

```
(sdegeo:find-closest-face position)
```

Returns

```
(FACE . REAL)
```

Arguments

Argument	Description
position	Argument type: POSITION

Appendix A: Commands

sdegeo:find-closest-vertex

sdegeo:find-closest-vertex

This Scheme extension goes through the vertex list of all defined geometric bodies and returns the entity ID of the closest vertex to the specified position.

Syntax

```
(sdegeo:find-closest-vertex position)
```

Returns

```
(VERTEX.REAL)
```

Arguments

Argument	Description
position	Argument type: POSITION

Appendix A: Commands

sdegeo:find-touching-faces

sdegeo:find-touching-faces

This Scheme extension takes a list of faces and returns a list of pairs. The face pair list contains entries only for those faces from the input list that have neighbor faces. For these faces, the input face entity ID and the entity ID of the neighboring touching face are returned as a pair.

Syntax

```
(sdegeo:find-touching-faces face-list)
```

Returns

```
ENTITY (FACE PAIR) | (FACE PAIR LIST)
```

Arguments

Argument	Description
face-list	Specifies a face or list of faces. Argument type: FACE FACE LIST

Examples

```
(sde:clear)
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
    "Silicon" "x1"))
(define mb2 (sdegeo:create-cuboid (position 3 3 6) (position 7 7 12)
    "PolySilicon" "x2"))
(define nfl (sdegeo:find-touching-faces (car (find-face-id
    (position 5 3 7) mb2))))
;; In this case, (cdar nfl) returns the face on the Silicon body
;; that is neighbor to the PolySilicon (car (find-face-id (position 5
;; 3 7) mb2)) face (there is only one such face).
(define nfl (sdegeo:find-touching-faces (entity:faces mb2)))
;; Return a list that contains all neighbor Silicon faces (listing as
;; PAIRS with the corresponding PolySilicon face). There are 5 such
;; face pairs returned in the nfl list, since there are 5 neighbor
;; Silicon faces to the PolySilicon body.
```

Appendix A: Commands

sdegeo:find-touching-faces-global

sdegeo:find-touching-faces-global

This Scheme extension takes a list of faces and returns a face list. The returned list contains entries only for those faces from the input list that have neighbor faces. For these faces, the neighboring touching face IDs are returned.

Syntax

```
(sdegeo:find-touching-faces-global face-list)
```

Returns

ENTITY FACE LIST

Arguments

Argument	Description
face-list	Specifies a face or list of faces. Argument type: FACE FACE LIST

Examples

```
(sde:clear)
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
    "Silicon" "x1"))
(define mb2 (sdegeo:create-cuboid (position 3 3 6) (position 7 7 12)
    "PolySilicon" "x2"))
(define nfl (sdegeo:find-touching-faces-global (car (find-face-id
    (position 5 3 7) mb2))))
(define nfl (sdegeo:find-touching-faces-global (entity:faces mb2)))
```

sdegeo:get-active-work-plane

This Scheme extension returns the name of the active work plane.

Syntax

```
(sdegeo:get-active-work-plane)
```

Returns

STRING

Appendix A: Commands

sdegeo:get-auto-region-naming

sdegeo:get-auto-region-naming

This Scheme extension returns the status of the automatic region-naming option. It returns #t if the automatic region-naming option is switched on. It returns #f if it is switched off.

Syntax

```
(sdegeo:get-auto-region-naming)
```

Returns

BOOLEAN

sdegeo:get-contact-edgelist

This Scheme extension returns all the edges that have the specified contact set name attached as a 2d-contact attribute.

Syntax

```
(sdegeo:get-contact-edgelist contact-set-name)
```

Returns

EDGE LIST

Arguments

Argument	Description
contact-set-name	Specifies the name of a contact set. Argument type; STRING

Appendix A: Commands

sdegeo:get-contact-facelist

sdegeo:get-contact-facelist

This Scheme extension returns all faces that have the specified contact set name attached as a 3d-contact attribute.

Syntax

```
(sdegeo:get-contact-facelist contact-set-name)
```

Returns

FACE LIST

Arguments

Argument	Description
contact-set-name	Specifies the name of a contact set. Argument type; STRING

sdegeo:get-current-contact-set

This Scheme extension returns the name of the current (active) contact set.

Syntax

```
(sdegeo:get-current-contact-set)
```

Returns

STRING

Examples

```
(sdegeo:define-contact-set "Drain")
(sdegeo:define-contact-set "Source" 4.0 (color:rgb 1.0 0.0 0.0) "##")
(sdegeo:set-current-contact-set "Drain")
(sdegeo:get-current-contact-set)
```

Appendix A: Commands

sdegeo:get-default-boolean

sdegeo:get-default-boolean

This Scheme extension returns the default Boolean behavior as a string, which can be any of the following predefined values: "AB", "ABA", "BAB", "ABiA", "ABiB", and "XX".

Syntax

```
(sdegeo:get-default-boolean)
```

Returns

STRING

sdegeo:get-region-counter

This Scheme extension returns the valid region counter.

By default, region names are assigned automatically to the generated regions. The generated region names are `region_%N`, where `%N` is the region counter. The Scheme extension returns `%N`.

Syntax

```
(sdegeo:get-region-counter)
```

Returns

INTEGER

sdegeo:imprint-circular-wire

This Scheme extension imprints a circular wire to an existing body. The imprinted wire splits the faces of the existing bodies. This Scheme extension can be used to prescribe a circular contact area.

The imprint functions split existing faces to surface patches. After the face split, the newly created face patches to be marked as contacts must be identified. During the assignment of contacts, these face ID numbers can be used explicitly.

Syntax

```
(sdegeo:imprint-circular-wire center-position radius)
```

Returns

FACE_ID | FACE_ID_LIST of imprinted faces

Arguments

Argument	Description
center-position	Specifies the center of the circular wire body. Argument type: POSITION
radius	Specifies the radius of the circular patch. Argument type: REAL

Appendix A: Commands

sdegeo:imprint-contact

sdegeo:imprint-contact

This Scheme extension imprints all 3D contacts (face contacts) to the neighboring touching faces.

Syntax

```
(sdegeo:imprint-contact)
```

Returns

None

Examples

```
(sde:clear)
(define mb1 (sdegeo:create-cuboid (position 0 0 0) (position 10 10 4)
    "Silicon" "x1"))
(define mb2 (sdegeo:create-cuboid (position 3 3 4) (position 7 7 6)
    "PolySilicon" "x2"))
(sdegeo:define-contact-set "base" 4 (color:rgb 1 0 0) "##")
(sdegeo:set-current-contact-set "base")
(define siliconface (car (find-face-id (position 5 5 4) mb1)))
(sdegeo:set-contact-faces (list siliconface) "base")
;; Now the contact face is defined on the Silicon body (mb1) but the
;; contact is not yet defined on the touching face of the PolySilicon
;; body (mb2).
(sdegeo:imprint-contact)
;; The contact is transferred to the neighbor face of the PolySilicon
;; body as well.
```

sdegeo:imprint-polygonal-wire

This Scheme extension imprints a polygonal wire to an existing body. The imprinted wire splits the faces of the existing body. This Scheme extension can be used to prescribe a polygonal contact area. The imprint functions split existing faces to surface patches. After the face split, the newly created face patches to be marked as contacts must be identified. During the assignment of contacts, these face ID numbers can be used explicitly.

The argument `vertex-list` contains the vertices of the imprinted polygon. The polygon must be closed to imprint it, so the first position must be repeated as the last position.

Syntax

```
(sdegeo:imprint-polygonal-wire vertex-list)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies the vertices of the imprinted polygon. Argument type: POSITION LIST

sdegeo:imprint-rectangular-wire

This Scheme extension imprints a rectangular wire to an existing body. The imprinted wire splits the faces of the existing body. This Scheme extension can be used to prescribe a rectangular contact area. The imprint functions split existing faces to surface patches. After the face split, the newly created face patches to be marked as contacts must be identified. During the assignment of contacts, these face ID numbers can be used explicitly.

Syntax

```
(sdegeo:imprint-rectangular-wire position1 position2)
```

Returns

None

Arguments

Argument	Description
position1 position2	Specify the two opposite corners of the imprinted rectangular body. Argument type: POSITION

sdegeo:imprint-triangular-wire

This Scheme extension imprints a triangular wire to an existing body. The imprinted wire splits the faces of the existing body. This Scheme extension can be used to prescribe a triangular contact area. The imprint functions split existing faces to surface patches. After the face split, the newly created face patches to be marked as contacts must be identified. During the assignment of contacts, these face ID numbers can be used explicitly.

Syntax

```
(sdegeo:imprint-triangular-wire position1 position2 position3)
```

Returns

None

Arguments

Argument	Description
position1 position2 position3	Specify the three vertices of the imprinted triangular wire. Argument type: POSITION

sdegeo:insert-vertex

This Scheme extension inserts a vertex in a 2D model by projecting the specified insert position to the nearest edge.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:insert-vertex vertex-list)
```

Returns

None

Arguments

Argument	Description
vertex-list	Specifies a list of vertices. Argument type: VERTEX_LIST

Appendix A: Commands

sdegeo:max-edge-length

sdegeo:max-edge-length

This Scheme extension computes the length of each edge in `edge-list` and returns the length of the longest edge.

Syntax

```
(sdegeo:max-edge-length edge-list)
```

Returns

REAL

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "AB")
(sdegeo:create-rectangle (position 0 0 0) (position 2 2 0) "Silicon"
  "region_1")
(sdegeo:create-circle (position 1 2 0) 1 "Silicon" "region_2")
(define elist (entity:edges (get-body-list)))
(sdegeo:max-edge-length elist)
```

Appendix A: Commands

sdegeo:min-edge-length

sdegeo:min-edge-length

This Scheme extension computes the length of each edge in `edge-list` and returns the length of the shortest edge.

Syntax

```
(sdegeo:min-edge-length edge-list)
```

Returns

REAL

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "AB")
(sdegeo:create-rectangle (position 0 0 0) (position 2 2 0) "Silicon"
  "region_1")
(sdegeo:create-circle (position 1 2 0) 1 "Silicon" "region_2")
(define elist (entity:edges (get-body-list)))
(sdegeo:min-edge-length elist)
```

Appendix A: Commands

sdegeo:mirror-selected

sdegeo:mirror-selected

This Scheme extension reflects the specified entities.

The reflection data is specified through a transform, which is a transform:reflection-type transform object.

If keep-flag is #t and the mirrored body overlaps the original body, then the two bodies are merged in the case of geometric bodies. Doping and Ref/Eval bodies are merged even if the original and the mirrored bodies are disjoint.

Syntax

```
(sdegeo:mirror-selected entity-list transform keep-flag  
[reverse-DRS-flag])
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies a list of entities. Argument type: ENTITY LIST
transform	Argument type: transform:reflection
keep-flag	Argument type: BOOLEAN
reverse-DRS-flag	Optional. If this argument is set to #t, then doping and Ref/Eval wire bodies such as the direction of the doping baselines are reversed before being mirrored. So that after mirroring, the doping direction remains the same as the original. The default is #f. Argument type: BOOLEAN

Examples

```
(sde:clear)  
(define mb (sdegeo:create-rectangle (position 1 1 0) (position 2 2 0)  
"Silicon" "xx"))  
(define tr1 (transform:reflection (position 0 0 0) (gvector -1 0 0)))  
(sdegeo:mirror-selected (get-body-list) tr1 #t)
```

Appendix A: Commands

sdegeo:move-2d-regions

sdegeo:move-2d-regions

This Scheme extension moves the specified regions (identified by their entity IDs) to a new location.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:move-2d-regions body | body-list gvector)
```

Returns

BOOLEAN

Arguments

Argument	Description
body body-list	Specifies a region or a list of regions (identified by their entity IDs) to be moved. Argument type: BODY BODY LIST
gvector	Specifies the new position of the regions. Argument type: GVECTOR

Examples

```
(sde:clear)
(define mb1 (sdegeo:create-rectangle (position 0 0 0)
                                       (position 1 1 0) "Silver" "region_1"))
(sdegeo:move-2d-regions mb1 (gvector 1 1 0))
```

Appendix A: Commands

sdegeo:move-edge

sdegeo:move-edge

This Scheme extension moves the specified edge (identified by its entity ID) to a new location.

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:move-edge edge gvector)
```

Returns

None

Arguments

Argument	Description
edge	Specifies an edge to be moved. Argument type: EDGE
gvector	Specifies the new position of the edge. Argument type: GVECTOR

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Silver" "region_1")
(define top-edge (car (find-edge-id (position 0.5 1 0))))
(sdegeo:move-edge top-edge (gvector 0.2 0.2 0))
```

Appendix A: Commands

sdegeo:move-vertex

sdegeo:move-vertex

This Scheme extension moves the specified vertex or vertex list (identified by its entity IDs) to a new location.

Note:

This Scheme extension applies to 2D models only. Vertices of 2D geometric bodies, as well as vertices of 2D DRS bodies (Ref/Eval windows), can be used.

Syntax

```
(sdegeo:move-vertex ve pe)
```

Returns

None

Arguments

Argument	Description
ve	Specifies a vertex or a list of vertices to be moved. Argument type: VERTEX VERTEX LIST
pe	Specifies the new location of the vertex or vertices. The new location can be given by a position entity or by a gvector (if a single vertex is the input). If the input is a vertex list, then the new vertex positions can be specified by either a gvector or a position list. Argument type: POSITION GVECTOR POSITION LIST

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Silver" "region_1")
(define lower-left-vertex (car (find-vertex-id (position 0 0 0))))
(sdegeo:move-vertex lower-left-vertex (position -0.2 0.2 0))
```

sdegeo:point-entity-relationship

This Scheme extension determines the containment relationship of the specified position with respect to the specified body:

- If the body contains the specified position, then the Scheme extension returns `inside`.
- If the body lies on a face of the body, then it returns `boundary`.
- If the point is not contained, then it returns `outside`.

Syntax

```
(sdegeo:point-entity-relationship body lposition)
```

Returns

STRING

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY
lposition	Argument type: POSITION

Appendix A: Commands

sdegeo:polygonal-split

sdegeo:polygonal-split

This Scheme extension splits a 2D device along the specified path. The first and last positions in the argument list snap to the closest boundary edge. All device bodies are split along the specified path, and a new unique region name is assigned to each split part (original region name plus `_N`, where `N` is a counter, which counts the split parts for each region separately).

Syntax

```
(sdegeo:polygonal-split plist)
```

Returns

None

Arguments

Argument	Description
plist	Specifies a list of positions. Argument type: POSITION LIST

Examples

```
(sdegeo:create-circle (position 7 5 0) 3 "Copper" "r3")
(sdegeo:create-rectangle (position 0 0 0) (position 10 5 0) "Silicon"
    "r1")
(sdegeo:create-rectangle (position 3 5 0) (position 7 9 0) "PolySi"
    "r2")
(sdegeo:polygonal-split (list (position 2 4.5 0) (position 4 1 0)
    (position 9 1 0) (position 9 6 0) (position 9 6 0)))
```

Appendix A: Commands

sdegeo:prune-vertices

sdegeo:prune-vertices

This Scheme extension merges neighboring edges by pruning the vertices. It is fast, very robust, and recommended to simplify 2D geometry.

The Scheme extension sdegeo:dnce (delete nearly collinear edges) calls sdegeo:prune-vertices for all bodies. See [sdegeo:dnce on page 647](#).

Note:

This Scheme extension applies to 2D models only.

Syntax

```
(sdegeo:prune-vertices {body-list | edge-list | vertex-list}  
angular-tolerance)
```

Returns

None

Arguments

Argument	Description
body-list	Specifies a list of bodies. Argument type: BODY LIST
edge-list	Specifies a list of edges. Argument type: EDGE LIST
vertex-list	Specifies a list of vertices. Argument type: VERTEX LIST
angular-tolerance	Specifies the angular threshold for the operation. If the edge normals (at the common vertex position) are less than the specified angular-tolerance (in degrees), then the edges merge by removing the common vertex and replacing the two neighboring linear edges by a single edge. Argument type: REAL

Appendix A: Commands

sdegeo:ray-test

sdegeo:ray-test

This Scheme extension returns the positions where a ray intersects a solid as pairs. The first element of each pair is the entity hit by the ray, and the second element of the pair is the position where the ray intersects the solid. The pairs are sorted along the direction of the ray. If the ray intersects a single face more than once, then the Scheme extension returns the first intersection.

Syntax

```
(sdegeo:ray-test entity ray radius)
```

Returns

```
((ENTITY . POSITION) ...)
```

Arguments

Argument	Description
entity	Specifies an entity. It must be a solid body. Argument type: BODY
ray	Specifies a ray, which consists of a position and a direction. Argument type: RAY
radius	Specifies a radius. Argument type: REAL

Examples

```
; solid:ray-test
; Create a solid block.
(define block1 (solid:block (position 0 0 0) (position 40 40 40)))
; Determine where the ray intersects the solid block.
(sdegeo:ray-test block1 (ray (position 10 20 60) (gvector 0 0 -1)) 0.1)
; ((#[face 20 1] . #[position 10 20 40]) (#[face 19 1] .
; #[position 10 20 0]))

(sde:clear)
(define mysphere (sdegeo:create-sphere (position 5 5 10) 3 "Gold" "x2"))
(sdegeo:ray-test mysphere (ray (position 5 5 20) (gvector 0 0 -1)) 0.1)
; ((#[face 27 1] . #[position 5 5 13]))
```

Appendix A: Commands

sdegeo:reflect

sdegeo:reflect

This Scheme extension reflects the specified entities.

Note:

This Scheme extension is available but obsolete. Instead, use sdegeo:mirror-selected (see [sdegeo:mirror-selected on page 669](#)).

Syntax

```
(sdegeo:reflect entity-list base-position base-vector keep-flag)
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies a list of entities. Instead of an entity list, you can use the keyword "all", in which case, the operation is performed on <i>all</i> entities. Argument type: BODY BODY LIST
base-position	Specifies the base position that is used for the operation. Argument type: POSITION
base-vector	Specifies the direction of the reflection as a gvector. The reflect operation generates new bodies that inherit the material properties of the parent bodies. If base-vector points to the x-direction, then the region names for the new regions are inherited from the parent region names and .x is appended to the region names. If base-vector points to the y-direction, then .y is appended to the region names. Similarly, for the z-direction, .z is used. In all other cases, .tr is added to the region names. Argument type: GVECTOR
keep-flag	Specifies whether the parent elements are kept. If keep-flag is #t, then the parent elements are kept. If it is #f, then the parent elements are deleted. Argument type: BOOLEAN

Appendix A: Commands

sdegeo:rename-contact

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "region_1")
(sdegeo:create-rectangle (position 0 10 0) (position 10 10.1 0)
    "Oxide" "region_2")
(sdegeo:create-rectangle (position 4 10.1 0) (position 6 14 0)
    "PolySilicon" "region_3")
; (sdegeo:extrude "all" 4)
(sdegeo:reflect "all" (position 10 0 0) (gvector 1 0 0) #t)
```

sdegeo:rename-contact

This Scheme extension renames an existing contact.

Syntax

```
(sdegeo:rename-contact oldContName newContName)
```

Returns

None

Arguments

Argument	Description
oldContName	Specifies the name of a contact set. Argument type: STRING
newContName	Specifies the new name of a contact set. Argument type: STRING

Appendix A: Commands

sdegeo:revolve

sdegeo:revolve

This Scheme extension revolves the specified entity or entities with the specified `base-position` and `base-vector`. The revolve operation is always performed with respect to the active coordinate system.

When you use the GUI, the revolve operation is applied to the selected entities. The Scheme extension (`sde:selected-entities`) returns the selected entity list. If no entities are selected and the GUI is used, then the operation is performed for the complete model.

Caution:

When the revolve operation is applied to only some parts of a device, overlapping regions or gaps can be created. In this case, explicit Boolean operations must be performed on the model to avoid overlapping regions.

Syntax

```
(sdegeo:revolve entity-list base-position base-vector angle)
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Instead of an entity list, you can use the keyword "all", in which case, the entire device is revolved. Argument type: BODY BODY LIST
base-position	Specifies the base position about which the listed entities are revolved. Argument type: POSITION
base-vector	Specifies the axis of the revolve operation. Argument type: GVECTOR
angle	Specifies an angle. Argument type: REAL

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0) "Silicon"
    "region_1")
(sdegeo:create-rectangle (position 0 10 0) (position 10 10.1 0)
    "Oxide" "region_2")
(sdegeo:create-rectangle (position 4 10.1 0) (position 6 14 0)
    "PolySilicon" "region_3")
```

Appendix A: Commands

sdegeo:rotate-selected

```
(sdegeo:revolve "all" (position 0 0 0) (gvector 0 1 0) 90)
(sdegeo:reflect "all" (position 0 0 0) (gvector -1 0 0) #t)
```

sdegeo:rotate-selected

This Scheme extension rotates the specified entities.

The rotation data is specified through a transform, which is a `transform:rotation-type` transform object.

Syntax

```
(sdegeo:rotate-selected entity-list transform
  {[keep-flag]} |
  {[keep-flag] [overlap-rule]} |
  {[keep-flag] [repeat-number]} |
  {[keep-flag] [overlap-rule] [repeat-number]} )
```

Returns

BOOLEAN

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Argument type: BODY BODY LIST
transform	Specifies a transformation. Argument type: <code>transform:rotation</code>
keep-flag	Optional. If <code>keep-flag</code> is <code>#t</code> , then you can also specify <code>repeat-number</code> . If <code>keep-flag</code> is <code>#t</code> and the transformed entities overlap existing entities, then the default overlap removal rule is used to delete the overlaps, or another overlap removal rule can be specified using the <code>overlap-rule</code> argument. Argument type: BOOLEAN
overlap-rule	Optional. Specifies whether there is automatic Boolean overlap removal. The default is <code>#t</code> , in which case, overlaps are removed automatically. If <code>overlap-rule</code> is set to <code>#f</code> , then automatic overlap control is switched off. In addition, this argument can be any of the valid automatic Boolean overlap removal methods (such as "ABA" and "BAB"), in which case, the specified Boolean overlap removal rule is used when overlaps are deleted. Argument type: BOOLEAN or "ABA", "BAB"
repeat-number	Optional. Specifies another overlap removal rule. Argument type: INTEGER

Appendix A: Commands

sdegeo:rotate-selected

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(define mb (sdegeo:create-rectangle (position 10 0 0)
    (position 20 10 0) "Silicon" "xx"))
(define tr (transform:rotation (position 0 0 0) (gvector 0 0 1) 30))
(sdegeo:rotate-selected mb tr #t 6)
;; overlaps are not removed

(roll)
(sdegeo:rotate-selected mb tr #t 6 #f)
;; same as above, the overlaps are not removed

(roll)
(sdegeo:rotate-selected mb tr #t 6 #t)
;; overlaps are removed, using the current "ABA" rule

(roll)
(sdegeo:rotate-selected mb tr #t 6 "BAB")
;; overlaps are removed, using the specified "BAB" rule
```

Appendix A: Commands

sdegeo:scale

sdegeo:scale

This Scheme extension scales the specified entity or entities with the specified scaling factors. Scaling is defined by its xyz components. The z-component is optional.

When you use the GUI, scaling is applied to the selected entities. The Scheme extension (`(sde:selected-entities)`) returns the selected entity list. If no entities are selected and the GUI is used, then the operation is performed for the complete model.

The scale operation is always performed with respect to the active coordinate system, that is, the xyz values are applied in the active coordinate system, in the local xyz directions. (Choose **Edit > Transform > Scale.**)

Note:

When the operation is applied to only some parts of a device, overlapping regions or gaps can be created. In this case, explicit Boolean operations must be performed on the model to avoid overlapping regions.

Syntax

```
(sdegeo:scale entity-list scale-x scale-y [scale-z])
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Instead of an entity or an entity list, you can use the keyword "all", in which case, the entire device is scaled. Argument type: BODY BODY LIST
scale-x	Specifies the scaling factor for the x-component. Argument type: REAL
scale-y	Specifies the scaling factor for the y-component. Argument type: REAL
scale-z	Optional. Specifies the scaling factor for the z-component. Argument type: REAL

Appendix A: Commands

sdegeo:scale-selected

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0) "Silicon"
    "region_1")
(sdegeo:create-rectangle (position 4 8 0) (position 6 14 0)
    "PolySilicon" "region_2")
(sdegeo:scale "all" 2 1 1)
```

sdegeo:scale-selected

This Scheme extension scales the specified entities.

The scaling data is specified through a transform, which is a `transform:scaling-type` transform object.

Syntax

```
(sdegeo:scale-selected entity-list transform)
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Argument type: BODY BODY LIST
transform	Specifies a transformation. Argument type: <code>transform:scaling</code>

sdegeo:set-active-coord-sys

This Scheme extension sets the specified coordinate system as the active coordinate system. All subsequent geometry creation commands are applied to that coordinate system.

Syntax

```
(sdegeo:set-active-coord-sys cs-name)
```

Returns

BOOLEAN

Arguments

Argument	Description
cs-name	Specifies the name of a coordinate system. Argument type: STRING

sdegeo:set-active-work-plane

This Scheme extension sets the specified work plane as the active work plane. All subsequent geometry creation commands are applied to that work plane.

Syntax

```
(sdegeo:set-active-work-plane wp-name)
```

Returns

BOOLEAN

Arguments

Argument	Description
wp-name	Specifies the name of a work plane. Argument type: STRING

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0) "Silicon"
    "region_1")
(sdegeo:define-work-plane "wp1" (position 0 0 1) (position 1 0 1)
    (position 0 1 1))
(sdegeo:set-active-work-plane "wp1")
```

Appendix A: Commands

sdegeo:set-auto-region-naming

```
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "PolySilicon" "region_2")
; view the model, by rotating the view.
```

sdegeo:set-auto-region-naming

This Scheme extension switches on or off the automatic region-naming option.

Regions are named using `region_%d`, where `%d` is an increasing integer.

Syntax

```
(sdegeo:set-auto-region-naming on-off)
```

Returns

BOOLEAN

Arguments

Argument	Description
on-off	Specifies whether to switch on automatic region naming. Argument type: BOOLEAN

Appendix A: Commands

sdegeo:set-contact

sdegeo:set-contact

This Scheme extension assigns the active contact to the specified entities.

Syntax

```
(sdegeo:set-contact entity-list  
[contact-name ["remove"] ["imprint" BOOLEAN]])
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Argument type: ENTITY ENTITY LIST
contact-name	Optional. Specifies a contact name. Argument type: STRING
"remove"	Optional. If you specify this option with the contact name, then the specified entity or list of entity bodies are deleted, and the contacts are defined only for the imprinted edges or faces. Argument type: STRING
"imprint"	Optional. Specifies whether 3D contacts are imprinted automatically to the touching neighbor faces. The default is #t. Three-dimensional contacts are imprinted automatically to the touching neighbor faces, unless you specify "imprint" #f in the argument list. Argument type: BOOLEAN

Description

This Scheme extension attaches the active contact to the specified entity, or entities, if you do not specify the optional contact-name argument. If you specify contact-name, then it is activated (or created as a contact if it does not exist).

If the entity type is EDGE, then a 2D contact is defined. If the entity type is FACE (and the entity belongs to a 3D body), then a 3D contact is assigned. If the entity is a 2D body, then a 2D contact is defined on all edges of the entity. If the entity is a 3D body, a 3D contact is defined on all faces of the entity. (Choose **Device > Contacts > Contact**.)

Appendix A: Commands

sdegeo:set-contact-boundary-edges

This Scheme extension is more convenient and can replace the following Scheme extensions:

- sdegeo:set-contact-boundary-edges
- sdegeo:set-contact-boundary-faces
- sdegeo:set-contact-edges
- sdegeo:set-contact-faces

sdegeo:set-contact-boundary-edges

This Scheme extension attaches the name of the active contact set to all edges of a specified body. It defines all edges of a selected region or regions as contacts, using the name of the active contact set and the selected entities (`sde:selected-entities`).

Alternatively, a body list can be specified in the argument list. In this case, (`sde:selected-entities`) is ignored and the edge list is extracted from the specified body list. (Choose **Device > Contacts > Set Region Boundary Edges.**)

Syntax

```
(sdegeo:set-contact-boundary-edges body | body-list)
```

Returns

None

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY
body-list	Specifies a list of bodies. Argument type: BODY LIST

Appendix A: Commands

sdegeo:set-contact-boundary-faces

sdegeo:set-contact-boundary-faces

This Scheme extension attaches a contact attribute to each face of the specified body or bodies. If the argument list is empty, then the Scheme extension takes the bodies from (sde:selected-entities).

The name of the active contact set is taken to specify the contacts. Alternatively, to the selected faces, a face list can be given explicitly in the argument list.

Syntax

```
(sdegeo:set-contact-boundary-faces body | body-list)
```

Returns

None

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY
body-list	Specifies a list of bodies. Argument type: BODY LIST

Examples

```
(sde:clear)
(sdegeo:define-contact-set "xx" 4.0 (color:rgb 1 0 0) "##")
(sdegeo:set-current-contact-set "xx")
(define mycube (sdegeo:create-cuboid (position 0 0 0) (position 1 1 1)
                                         "PolySilicon" "region_1"))
(sdegeo:set-contact-boundary-faces mycube)
```

Appendix A: Commands

sdegeo:set-contact-edges

sdegeo:set-contact-edges

This Scheme extension marks all listed edges as contacts.

When you use the GUI to add contact attributes to the edges, the selected entities are used to initialize the edge list. The Scheme extension (`sde:selected-entities`) returns the edges that were previously selected (and that are highlighted) from the GUI as contacts. The contact set name is the active contact set name. (Choose **Device > Contacts > Set Edges**.)

Syntax

```
(sdegeo:set-contact-edges edge-list contact-set-name)
```

Returns

None

Arguments

Argument	Description
edge-list	Specifies a list of edges. Argument type: EDGE LIST
contact-set-name	Specifies the name of the contact set to which the contact edges will belong. The specified contact set must be predefined. Argument type: STRING

Appendix A: Commands

sdegeo:set-contact-faces

sdegeo:set-contact-faces

This Scheme extension attaches a contact attribute to faces. If the argument list is empty, then the Scheme extension takes the faces from (sde:selected-entities). The name of the active contact set is taken to specify the contacts. Alternatively, to the selected faces, a face list can be given explicitly in the argument list. (Choose **Device > Contacts > Set Faces.**)

Syntax

```
(sdegeo:set-contact-faces face-list)
```

Returns

None

Arguments

Argument	Description
face-list	Specifies a face or a list of faces. Argument type: FACE FACE LIST

sdegeo:set-contact-faces-by-polygon

This Scheme extension provides an alternative way to prescribe contacts. It can attach a contact attribute to only a part of an existing face (or list of faces).

The specified polygons are imprinted on the faces of the existing model. After the imprint operation, the original faces are split and the contact attribute is attached to the imprinted face segment.

Syntax

```
(sdegeo:set-contact-faces-by-polygon polygon-list normal-list  
contact-set-name)
```

Returns

None

Arguments

Argument	Description
polygon-list	Specifies a list of polygons. A polygon is a POSITION list. Argument type: POLYGON LIST
normal-list	Specifies a list of normal vectors. A normal vector must be specified for each face to mark the direction of the imprint. Argument type: GVECTOR LIST
contact-set-name	Specifies the name of the contact set. Argument type: STRING

Examples

```
; To define a polygonal contact region, use sde:define-contact  
; which is a list of polygons (a polygon is a list of 3D vertices).  
; Example:  
(sde:clear)  
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10) "Silicon"  
"r1")  
(sdegeo:define-contact-set "c1" 4 (color:rgb 1 0 0) "##")  
(sdegeo:define-contact-set "c2" 4 (color:rgb 1 0 0) "==")  
(sdegeo:define-contact-set "c3" 4 (color:rgb 1 0 0) "//")  
(sdegeo:set-current-contact-set "c1")  
(sdegeo:set-contact-faces-by-polygon (list (list (position 8 7 10)  
 (position 9 8 10) (position 8 9 10) (position 7 8 10)  
 (position 8 7 10)))  
 (list (gvector 0 0 -1)) "c1")  
(sdegeo:set-contact-faces-by-polygon (list (list (position 0 0 0)  
 (position 2 0 0) (position 2 3 0) (position 0 3 0)
```

Appendix A: Commands

sdegeo:set-current-contact-set

```
(position 0 0 0)))
(list (gvector 0 0 1)) "c2")
(sdegeo:set-contact-faces-by-polygon (list (list (position 0 5 2)
(position 0 8 5) (position 0 5 8) (position 0 2 5))
(list (position 10 5 2) (position 10 8 5) (position 10 5 8)
(position 10 2 5)))
(list (gvector 1 0 0) (gvector -1 0 0)) "c3")
(sdegeo:set-contact-faces-by-polygon (list (list (position 0 5 2)
(position 0 8 5) (position 0 5 8) (position 0 2 5))
(list (position 10 5 2)(position 10 8 5) (position 10 5 8)
(position 10 2 5)))
(list (gvector 1 0 0) (gvector -1 0 0)) "c3")
```

sdegeo:set-current-contact-set

This Scheme extension sets the name of the current (active) contact to the specified contact set. An error occurs if the specified contact set is not yet defined.

Syntax

```
(sdegeo:set-current-contact-set cs-name)
```

Returns

Name of the previously defined contact set

Arguments

Argument	Description
cs-name	Specifies the name of a contact set. Argument type: STRING

Examples

```
(sdegeo:define-contact-set "Drain")
(sdegeo:define-contact-set "Source" 4.0 (color:rgb 1.0 0.0 0.0) "##")
(sdegeo:set-current-contact-set "Drain")
```

Appendix A: Commands

sdegeo:set-default-boolean

sdegeo:set-default-boolean

This Scheme extension sets the default Boolean behavior.

Syntax

```
(sdegeo:set-default-boolean boolean-value)
```

Returns

STRING

Arguments

Argument	Description
boolean-value	<p>Specifies the default Boolean behavior. This argument can be any of the following predefined values (the default behavior determines how overlapping regions are treated):</p> <ul style="list-style-type: none">• When set to "AB", the newly created regions are merged (united) automatically with all existing overlapping regions. The merged regions inherit the DATEX material (and region name) from the new region.• "ABA" subtracts all overlapping regions from the existing regions.• "BAB" subtracts all existing regions from the newly created regions.• The "ABiA" behavior is similar to the "ABA" behavior, except that the overlaps are separate regions (with the DATEX material inherited from the new regions).• "ABiB" is similar to "ABiA", except that the overlap regions inherit the DATEX material from the existing regions.• "xx" allows the creation of overlapping regions. In this case, you must modify the model by explicitly deleting the overlapping parts, before generating the tessellated boundary output. <p>Argument type: STRING</p>

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "AB")
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "region_1")
(sdegeo:create-rectangle (position 3 8 0) (position 7 13 0)
    "PolySilicon" "region_2")
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position 20 0 0) (position 30 10 0)
    "Silicon" "region_3")
(sdegeo:create-rectangle (position 23 8 0) (position 27 13 0)
    "PolySilicon" "region_4")
(sdegeo:set-default-boolean "BAB")
(sdegeo:create-rectangle (position 40 0 0) (position 50 10 0)
    "Silicon" "region_5")
```

Appendix A: Commands

sdegeo:set-region-counter

```
(sdegeo:create-rectangle (position 43 8 0) (position 47 13 0)
    "PolySilicon" "region_6")
(sdegeo:set-default-boolean "ABiA")
(sdegeo:create-rectangle (position 0 20 0) (position 10 30 0)
    "Silicon" "region_7")
(sdegeo:create-rectangle (position 3 28 0) (position 7 33 0)
    "PolySilicon" "region_8")
(sdegeo:set-default-boolean "ABiB")
(sdegeo:create-rectangle (position 20 20 0) (position 30 30 0)
    "Silicon" "region_9")
(sdegeo:create-rectangle (position 23 28 0) (position 27 33 0)
    "PolySilicon" "region_10")
(sdegeo:set-default-boolean "XX")
(sdegeo:create-rectangle (position 40 20 0) (position 50 30 0)
    "Silicon" "region_11")
(sdegeo:create-rectangle (position 43 28 0) (position 47 33 0)
    "PolySilicon" "region_12")
```

sdegeo:set-region-counter

This Scheme extension sets the region counter explicitly.

When the automatic region-naming option is selected, the generated regions are numbered sequentially using the region counter. Regions are named automatically as Region_1, Region_2, and so on. When the region counter is set to N, the next generated region is named Region_N.

Syntax

```
(sdegeo:set-region-counter counter)
```

Returns

None

Arguments

Argument	Description
counter	Specifies the region counter. Argument type: INTEGER

Appendix A: Commands

sdegeo:set-region-counter-aut

sdegeo:set-region-counter-aut

This Scheme extension sets the region counter automatically.

The region attribute is checked for each existing regions and, if a `Region_N` attribute is found, the region counter attribute is set to the next available number $N+1$. When a boundary file is loaded, this Scheme extension is called automatically.

Syntax

```
(sdegeo:set-region-counter-aut)
```

Returns

None

Appendix A: Commands

sdegeo:skin-wires

sdegeo:skin-wires

This Scheme extension defines a 3D body such that the given wires or faces define the cross sections to be interpolated by the resulting 3D body.

At least two wire bodies or faces must be specified. The wires can be open or closed. The wires are copies, that is, the originals remain. The wires can share endpoints and do not have to be C1 continuous.

Syntax

```
(sdegeo:skin-wires body-list [path] [skin-options])
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
body-list	Specifies a list of wire bodies. Argument type: WIRE LIST FACE LIST
path	Optional. Defines a curve that is intersected with the plane of each profile. At each of the resulting intersection points on the curve, the tangent vector is calculated and is applied to the surface as a constraint at that profile. Argument type: WIRE
skin-options	Optional. Defines different skinning options. For a complete list of skinning options, see skin:options on page 830 . Argument type: SKIN OPTIONS

Examples

```
(sde:clear)
(define b1 (sdegeo:create-circle (position 0 0 0) 10 "Copper" "r1"))
(define b2 (sdegeo:create-ellipse (position 0 0 20)
                                  (position 20 0 20) 0.5 "Copper" "r2"))
(define f1 (car (entity:faces b1)))
(define f2 (car (entity:faces b2)))
(define bs (sdegeo:skin-wires (list f1 f2)))
(sde:add-material bs "Copper" "bs")
(entity:delete (list b1 b2))
```

Appendix A: Commands

sdegeo:skin-wires-guides

sdegeo:skin-wires-guides

This Scheme extension creates a sheet body that interpolates a series of wires or faces with a guide curve.

The given wires or faces define the cross sections to be interpolated by the resulting sheet body. The wire bodies are assumed to be simple and well-behaved. The wires can be open or closed. The wires are copies, that is, the originals remain. The wires can share endpoints and do not have to be C1 continuous.

Syntax

```
(sdegeo:skin-wires-guides body-list guides [skin-options])
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
body-list	Specifies a list of wire bodies. Argument type: WIRE LIST FACE LIST
guides	Specifies the curves that the skinning surface directly follows (in the skinning direction). The guides must intersect each wire profile within 1.0e-06 and must start and end on the first and last profile exactly. Any number of guides can be added, and they can fall directly on vertices or not. Guides must be C1 continuous and well-behaved (no looping). Argument type: EDGE LIST
skin-options	Optional. Defines different skinning options. For a complete list of skinning options, see skin:options on page 830 . Argument type: SKIN OPTIONS

Appendix A: Commands

sdegeo:skin-wires-normal

sdegeo:skin-wires-normal

This Scheme extension creates a sheet body that interpolates a series of wires or faces with take-off vectors normal to the plane of the wire body.

The given wires or faces define the cross sections to be interpolated by the resulting sheet body with the additional constraint of the surface take-off vectors leaving normal to the plane of each of the wire bodies.

There must be at least two wire bodies. The wire bodies are assumed to be simple, meaning only the first wire of each body is used for skinning. The wires can be open or closed. The wires are copies, that is, the originals remain. The wires can share endpoints and do not have to be C1 continuous.

Syntax

```
(sdegeo:skin-wires-normal body-list normal-type [skin-options])
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
body-list	Specifies a list of wire bodies. Argument type: WIRE LIST FACE LIST
normal-type	Specifies to which profiles the normal constraint must be applied. Values are: <ul style="list-style-type: none">• "first_normal"• "last_normal"• "ends_normal"• "all_normal" In the case of "first_normal", the constraint is applied only to the first profile. Other values follow similarly. Argument type: STRING
skin-options	Optional. Defines different skinning options. For a complete list of skinning options, see skin:options on page 830 . Argument type: SKIN OPTIONS

Appendix A: Commands

sdegeo:skin-wires-vectors

sdegeo:skin-wires-vectors

This Scheme extension creates a sheet body that interpolates a series of wires with take-off vectors as constraints.

The wires specified in the argument list define the cross sections to be interpolated by the resulting sheet body. There must be at least two wire bodies. Wire bodies are assumed to be simple, meaning only the first wire of each body is used for skinning. Wires can be open or closed. Wires are copies, that is, the originals remain. Wires can share endpoints and do not have to be C1 continuous.

The argument `magnitude-list` is a list of magnitudes. Skinning with vectors accepts as additional constraints a list of vectors and, optionally, a list of magnitudes to control the take-off directions with which the skin surface leaves the profiles. The number of supplied vectors must equal the number of profiles if the `closed` option is set to `FALSE`. If `closed` is set to `TRUE`, then one more vector can be given. It is used on the copy that is made of the first profile to create a closed body. If no additional vector is provided, then the first vector is used also on the first profile copy. In a similar way, magnitudes can be provided. A profile is interpolated without constraint if the vector supplied for it is the zero vector.

Syntax

```
(sdegeo:skin-wires-vectors body-list SPAvector-list  
  ([guide-list] | [magnitude-list]) [skin-options])
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
body-list	Specifies a list of wire bodies. Argument type: WIRE LIST FACE LIST
SPAvector-list	Specifies a list of vectors. Argument type: GVECTOR LIST
guide-list	Optional. Specifies a list of guides. Argument type: EDGE LIST
magnitude-list	Optional. Specifies a list of magnitudes. Argument type: REAL LIST
skin-options	Optional. Defines different skinning options. For a complete list of skinning options, see skin:options on page 830 . Argument type: SKIN OPTIONS

Appendix A: Commands

sdegeo:split-insert-device

sdegeo:split-insert-device

This Scheme extension splits a device at a specified position and inserts a part between the split bodies. The Scheme extension applies to both 2D and 3D devices. In addition, the Scheme extension supports the splitting of wire bodies.

Syntax

```
(sdegeo:split-insert-device splitpos splittdir splitlength splitmerge)
```

Returns

BOOLEAN

Arguments

Argument	Description
splitpos	Specifies the position of the split. Argument type: POSITION
splittdir	Specifies the direction of the split. Argument type: GVECTOR
splitlength	Specifies the length of the split. Argument type: REAL
splitmerge	Specifies whether to merge the split faces. If set to #t, then the split faces merge. Argument type: BOOLEAN

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
    "Silicon" "xx")
(sdegeo:split-insert-device (position 0.2 0 0) (gvector 1 0 0) 0.4 #t)
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon" "xx")
(sdegeo:split-insert-device (position 0.2 0 0) (gvector 1 0 0) 0.4 #t)
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 1 1 1) "Silicon" "xx")
(sdegeo:split-insert-device (position 0.5 0.5 0.5) (gvector 1 1 0)
    0.4 #t)
(sde:clear)
(sdegeo:create-sphere (position 0 0 0) 1 "Silicon" "xx")
(sdegeo:split-insert-device (position 0 0 0) (gvector 1 0 0) 0.4 #t)
```

Appendix A: Commands

sdegeo:sweep

sdegeo:sweep

This Scheme extension performs a sweep operation. It creates a sheet body or solid body from a profile and a path.

The amount to revolve is controlled by the sweep-angle option.

This Scheme extension also applies to Ref/Eval windows. In the case of a submesh reference, information regarding the sweep parameters and the pre-swept Ref/Eval window is stored in the command file so that the meshing engine can reproduce doping information in the 3D swept profile. For other doping profile types, the swept Ref/Eval window must be a valid 2D shape (for reference/baseline) or a 3D shape (for evaluation window) since only the final swept element is stored in the command file.

Syntax

```
(sdegeo:sweep profile path [sweep-options])
```

Returns

ENTITY

Arguments

Argument	Description
profile	Specifies a profile. This argument is a pointer to a face, which in turn, defines the sweep geometry and becomes the base of the solid or the edge of the surface. Argument type: FACE
path	This argument is a wire body along which the profile is swept, and path can be defined as one of the following: <ul style="list-style-type: none">• A path• A distance if profile is a planar face<ul style="list-style-type: none">Specifies the distance to sweep along the face normal. Argument type: REAL• A vector if rail_law is used (no twist)<ul style="list-style-type: none">Specifies the direction and distance to sweep. Argument type: GVECTOR• An axis (defined as position and vector) if rail_law is used (no twist)<ul style="list-style-type: none">Specifies a position and vector that defines the axis to revolve about. Argument type: POSITION . VECTOR <p>Argument type: WIRE EDGE</p>

Appendix A: Commands

sdegeo:sweep

Argument	Description
sweep-options	<p>Optional. Creates the data structure. If <code>sweep-options</code> is not specified, then the defaults are used as follows:</p> <ul style="list-style-type: none">• Create a solid.• Draft angle is 0.• Gap type is 2 (natural).• Twist angle is 0.• Rail law is minimum rotation. <p>Argument type: SWEEP OPTIONS</p>

Errors

An error is reported if the result of the sweep operation is unsuccessful.

Appendix A: Commands

sdegeo:sweep-corner

sdegeo:sweep-corner

This Scheme extension performs a corner sweep operation. It creates a sheet body or solid body from a profile and an axis (position, vector).

The generated solid body is similar to the generated solid body using `sdegeo:sweep` (when the same axis method is used), except that the swept body is extended to form a right corner.

Syntax

```
(sdegeo:sweep-corner profile axis)
```

Returns

ENTITY

Arguments

Argument	Description
profile	Specifies a profile. Argument type: FACE FACE LIST
axis	Specifies a position and vector that defines the axis to revolve about. The amount to revolve is 90° for the corner sweep operation. Argument type: POSITION . VECTOR

Errors

An error is reported if the result of the corner sweep operation is unsuccessful.

Examples

```
(sde:clear)
(sdegeo:create-rectangle (position 1 0 0) (position 2 1 0)
    "Silicon" "region_1")
(sdegeo:sweep-corner (find-face-id (position 1.5 1 0))
    (position 0 0 0) (gvector 0 1 0))
```

Appendix A: Commands

sdegeo:taper-faces

sdegeo:taper-faces

This Scheme extension tapers a list of faces by the supplied draft angle about an axis defined by the intersection between the plane of the face and a taper plane, which is defined by a point and normal given as arguments to the Scheme extension.

The direction of the normal defines the direction of the angle, that is, the resulting angled plane will slope in the direction of the draft plane normal. The taper plane need not intersect the face to be tapered and, even when it does, this intersection need not be an edge of the body (unlike in edge tapering).

Note:

Only planes, cones, ruled surfaces, and previously plane-tapered surfaces (provided the same taper plane is used) can be plane tapered.

Syntax

```
(sdegeo:taper-faces face-list point normal angle)
```

Returns

BODY

Arguments

Argument	Description
face-list	Specifies the faces of a body to be tapered. Argument type: FACE LIST
point	Specifies a position on the taper plane. Argument type: POSITION
normal	Specifies the normal of the taper plane at the point specified. Argument type: GVECTOR
angle	Specifies the rotation angle in degrees. Argument type: REAL

Appendix A: Commands

sdegeo:translate

sdegeo:translate

This Scheme extension translates the specified entity or entities with the specified distance.

When you use the GUI, the translate operation is applied to the selected entities. The Scheme extension (`sde:selected-entities`) returns the selected entity list. If no entities are selected and the GUI is used, then the operation is performed for the complete model. The translation is defined by its xyz components. (Choose **Edit > Transform > Translate**.)

Note:

This Scheme extension is still available, but it is obsolete. Instead, use `sdegeo:translate-selected` (see [sdegeo:translate-selected on page 706](#)).

The translate operation is always performed with respect to the active coordinate system, that is, the xyz values are applied in the active coordinate system, in the local xyz directions.

When the Scheme extension is applied only to some parts of a device, overlapping regions or gaps can be created. In this case, explicit Boolean operations must be performed on the model to avoid overlapping regions.

Syntax

```
(sdegeo:translate entity-list translate-x translate-y [translate-z])
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Alternatively, if you specify "all", then the entire device is translated. Argument type: BODY BODY LIST
translate-x	Specifies the translation for the x-component. Argument type: REAL
translate-y	Specifies the translation for the y-component. Argument type: REAL
translate-z	Optional. Specifies the translation for the z-component. Argument type: REAL

Appendix A: Commands

sdegeo:translate-selected

Examples

```
(sde:clear)
(sdegeo:set-default-boolean "ABA")
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "region_1")
(sdegeo:create-polygon (list (position 2 0 0) (position 3 2 0)
    (position 7 2 0) (position 8 0 0) (position 8 -3 0)
    (position 7 -4 0) (position 3 -4 0) (position 2 -3 0)
    (position 2 0 0)) "PolySilicon" "region_2")
(sdegeo:translate "all" 1 0)
(sdegeo:define-coord-sys "cs" 0 0 45)
(sdegeo:set-active-coord-sys "cs")
(sdegeo:translate "all" 1 0)
```

sdegeo:translate-selected

This Scheme extension translates the specified entities. The translate data is specified through a transform, which is a `transform:translation-type` transform object.

Syntax

```
(sdegeo:translate-selected entity-list transform keep-flag
    [repeat-number])
```

Returns

None

Arguments

Argument	Description
entity-list	Specifies an entity or a list of entities. Argument type: BODY BODY LIST
transform	Specifies a transformation. Argument type: <code>transform:translation</code>
keep-flag	Specifies whether the translated geometric body overlaps the original body. If set to <code>#t</code> and the translated geometric body overlaps the original body, then the overlap is removed using the default overlap handling rule. Argument type: BOOLEAN
repeat-number	Optional. Argument type: INTEGER

Appendix A: Commands

sdegeo:vsmooth

sdegeo:vsmooth

This Scheme extension is used for 2D boundary smoothing and can be called multiple times. It uses a least squares fit of a small set of consecutive data points (vertices) to a polynomial and takes the calculated central point of the fitted polynomial curve as the new smoothed data point (vertex). The fitted polynomial can be either quadratic or cubic.

The first argument of the function is the polynomial order (2 or 3). The second argument is the vertex list that should be smoothed. You can specify all the vertices of the model as input by using the (entity:vertices (get-body-list)) command.

The global Scheme variable `bsmooth-ac` specifies an angular criterion (in degrees) for shape preservation (preserving ridges). The default is 100.

Syntax

```
(sdegeo:vsmooth polynomial-degree vertex-list)
```

Returns

None

Arguments

Argument	Description
polynomial-degree	Specifies the polynomial order (2 or 3). Argument type: INTEGER
vertex-list	Specifies the list of vertices to be smoothed. Argument type: VERTEX LIST

sdeicwb:clear

This Scheme extension removes from memory all data related to IC WorkBench.

Syntax

```
(sdeicwb:clear)
```

Returns

None

Appendix A: Commands

sdeicwb:contact

sdeicwb:contact

This Scheme extension creates contacts for subsequent device simulations that are tied to a mask or a text label in the TCAD layout file.

Syntax

```
(sdeicwb:contact ("label.name" label-name | "layer.name" layer-name)
  "type" "box" "material" material
  ["adjacent.material" adjacent-material]
  ([ "boxheight" boxheight] | [ "hi" hi "lo" lo]) [ "name" name])

(sdeicwb:contact ("label.name" label-name | "layer.name" layer-name)
  "type" "point" "material" material
  ["height" height] [ "replace" #t | #f] [ "name" name])
```

Returns

Returns #t if successful. Otherwise, #f if the contact cannot be created.

Arguments

Argument	Description
label-name	Argument type: STRING
layer-name	Argument type: STRING
"type"	Specifies the type of contact: box-type contact or point-type contact. Options are "box" "point". Argument type: STRING
material	Argument type: STRING
adjacent-material	Optional. Argument type: STRING
boxheight	Optional. Argument type: REAL
hi, lo	Optional. Define the box extent, where the contact faces are assigned, in the x-direction or z-direction (depending on the up direction, which is set by the sde:set-process-up-direction Scheme extension). Argument type: REAL
height	Optional. Argument type: REAL
replace	Optional. Argument type: BOOLEAN

Appendix A: Commands

sdeicwb:contact

Argument	Description
name	Optional. The name of the contact defaults to the name of the layer or the text label, depending on whichever is specified. Use the <code>name</code> argument to overwrite the default name. Argument type: STRING

Description

This Scheme extension extracts the lateral placement of a contact from either the mask specified in the "layer.name" argument or the anchor point location (or locations) of the text label specified in the "label.name" argument.

Note:

Before you can refer to a mask in `sdeicwb:contact` with the "layer.name" argument, you must create it first, for example, with the `sdeicwb:generate-mask-by-layer-name` Scheme extension or the `sdepe:generate-mask` Scheme extension.

Invoke `sde:separate-lumps` before calling `sdeicwb:contact` to avoid that a point-type contact is assigned erroneously to other lumps in the requested material, which are not connected to the lump marked with the text label.

This Scheme extension supports both box-type and point-type contacts:

- A *box-type* contact ("type" "box") 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 polygons, while the vertical extent is taken from the `lo` and `hi` arguments. If the `lo` argument is not explicitly given, the vertical extent is determined automatically, based on the topmost interface with the material in the command call. The vertical extent of the box is controlled by `boxheight`.

The contact area is defined by the surface of the material specified with the `material` argument, which lies inside the mentioned box. The contact area can be further restricted to only the part of the surface that also touches the material specified with the "adjacent.material" argument.

If a text label name is given, the layer associated with that text label is used.

- A *point-type* contact ("type" "point") contains all the boundary elements of one region. The lateral position of the point is determined automatically as a point inside the polygon, while the vertical position is taken from the `height` argument. If the `height` argument is not explicitly given, then the vertical position is determined automatically, based on the interfaces with the material specified with the `material` argument. If "replace" #t is set, then the original region of a contact is removed.

If a text label name is given, then the anchor point of the text label is used.

Appendix A: Commands

sdeicwb:contact

Note:

It might be advantageous to create auxiliary layers or text labels in IC Validator WorkBench for the placement of contacts.

Examples

The location of contact faces can be defined using different argument combinations. The following examples illustrate the use of the `sdeicwb:contact` arguments.

Example 1

The `hi` and `lo` arguments define the box extent, where the contact faces are assigned, in the x-direction or z-direction (depending on the up direction, which can be set by the `sde:set-process-up-direction` Scheme extension). The following example creates a simple cube with a mask, which covers the complete domain (base):

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silicon" "xx")
(sde:addmasktomodel "m1" (list (list 0 0 10 10)))
```

If a mask covers only part of the specified material, then the contact is assigned only that part of the device, as follows:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 5 10 10)
  "Silicon" "xx1")
(sdegeo:create-cuboid (position 5 0 0) (position 10 10 10)
  "PolySilicon" "xx2")
(sde:addmasktomodel "m1" (list (list 0 0 10 10)))
```

If the contact should be assigned to the complete bottom and top faces, specify the `lo` and `hi` arguments such that the bottom and the top faces are between the `lo` – `hi` interval, as follows:

```
(sdeicwb:contact "layer.name" "m1" "type" "box" "material" "Silicon"
  "lo" -1 "hi" 11)
```

or:

```
(sdeicwb:contact "layer.name" "m1" "type" "box" "material" "Silicon"
  "lo" (- (position:z (car (bbox-exact (get-body-list)))) 0.1)
  "hi" (+ (position:z (cdr (bbox-exact (get-body-list)))) 0.1))
```

Appendix A: Commands

sdeicwb:contact

Note:

Both `lo` and `hi` must be below or above the bottom or top face z-coordinate.

Therefore, the following command does not work:

```
(sdeicwb:contact "layer.name" "m1" "type" "box" "material"
  "Silicon" "lo" (position:z (car (bbox-exact (get-body-list))))
  "hi" (position:z (cdr (bbox-exact (get-body-list)))))
```

Example 2

If you use the default up direction "`-x`", then you must reverse the `lo` and `hi` coordinates by specifying:

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silicon" "xx")
(sde:addmasktomodel "m1" (list (list 0 0 10 10)))
(sdeicwb:contact "layer.name" "m1" "type" "box" "material" "Silicon"
  "lo" 11 "hi" -1)
```

or:

```
(sdeicwb:contact "layer.name" "m1" "type" "box" "material" "Silicon"
  "lo" (+ (position:x (cdr (bbox-exact (get-body-list)))) 0.1)
  "hi" (- (position:x (car (bbox-exact (get-body-list)))) 0.1))
```

Example 3

Instead of using the `lo` and `hi` arguments, you can use `boxheight`. If neither `lo` nor `hi` is given, but `boxheight` is specified, then the top of the region is computed (`x`) and `lo` is set to `x - boxheight` and `hi` is set to `x + boxheight`:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silicon" "xx")
(sde:addmasktomodel "m1" (list (list 2 2 8 8)))
(sdeicwb:contact "layer.name" "m1" "type" "box" "material" "Silicon"
  "boxheight" 11)
```

Since `boxheight` is larger than the vertical extent of the device, the contacts are assigned to both the top and the bottom faces, which are covered by the mask.

Example 4

If `boxheight` is less than the height of the device, then the contact is assigned only to the top face as follows:

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silicon" "xx")
```

Appendix A: Commands

sdeicwb:contact

```
(sde:addmasktomodel "m1" (list (list 2 2 8 8)))
(sdeicwb:contact "layer.name" "m1" "type" "box" "material"
  "Silicon" "boxheight" 9)
```

Similarly for the default "-x" up direction:

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silicon" "xx")
(sde:addmasktomodel "m1" (list (list 2 2 8 8)))
(sdeicwb:contact "layer.name" "m1" "type" "box" "material" "Silicon"
  "boxheight" 11)
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
  "Silicon" "xx")
(sde:addmasktomodel "m1" (list (list 2 2 8 8)))
(sdeicwb:contact "layer.name" "m1" "type" "box" "material" "Silicon"
  "boxheight" 9)
```

Appendix A: Commands

sdeicwb:create-boxes-from-layer

sdeicwb:create-boxes-from-layer

This Scheme extension creates cuboids with a lateral extent given by the polygon bounding boxes from the layer in an IC WorkBench TCAD layout. The vertical extent is taken from the argument list.

This Scheme extension supports layout-driven structural operations, with a syntax independent of the coordinate system used in the currently active IC WorkBench simulation domain.

Use this Scheme extension, for example, to superimpose a dummy region onto a structure to turn its interfaces into contacts and delete it afterwards.

Syntax

```
(sdeicwb:create-boxes-from-layer lname top bot region-material  
region-name)
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
lname	Specifies the name of a layer. Argument type: STRING
top	Specifies the top coordinate of a box. Argument type: REAL
bot	Specifies the bottom coordinate of a box. Argument type: REAL
region-material	Specifies the name of a region material. Argument type: STRING
region-name	Specifies the name of a region. Argument type: STRING

Examples

```
(define PoTop (sdeicwb:get-region-top (find-material-id "PolySilicon")))  
(sdegeo:set-current-contact-set "gate")  
(sdegeo:set-default-boolean "ABA")  
(define GATEMETAL (sdeicwb:create-boxes-from-layer "ngate"  
         (sdeicwb:down PoTop 0.05) (sdeicwb:up PoTop 0.1) "Metal" "R.dummy"))  
(sdegeo:set-contact-boundary-faces GATEMETAL)  
(sdegeo:delete-region GATEMETAL)
```

sdeicwb:define-refinement-from-layer

This Scheme extension creates refinement windows with a lateral extent given by layers in an IC WorkBench TCAD layout. The vertical extent is taken from the argument list. In addition, it generates 3D refinement size settings with coordinate system-independent syntax.

Syntax

```
(sdeicwb:define-refinement-from-layer "lname" lname ["rname" rname]
                                         ["oversize" oversize] "top" top "bot" bot
                                         "dlrmin" dlrmin "dlrmax" dlrmax
                                         "dbtmin" dbtmin "dbtmax" dbtmax
                                         ["dbfmin" dbfmin] ["dbfmax" dbfmax]
                                         ["material" material] ["region" region] ["use-bbox" use-bbox])
```

Returns

None

Arguments

Argument	Description
lname	Specifies the name of a layer. Argument type: STRING
rname	Optional. Specifies an optional name of a refinement window. The default is the layer name. Argument type: STRING
oversize	Optional. Use this argument to extend the refinement window beyond the lateral extent of the actual segments or polygon bounding boxes. A nonzero value is subtracted from or added to the minimum and maximum segment or the polygon bounding box coordinates, respectively. Note: To use oversize for a non-rectangular mask while continuing to avoid unnecessary refinement, use IC Validator WorkBench to break, for example, an L-shaped polygon into two rectangular polygons such that the union of the polygon bounding boxes coincides with the shape itself. Argument type: REAL
top	Specifies the top coordinate of a refinement window. Argument type: REAL
bot	Specifies the bottom coordinate of a refinement window. Argument type: REAL
dlrmin	Specifies the minimum refinement in the left-right dimension. Argument type: REAL

Appendix A: Commands

sdeicwb:define-refinement-from-layer

Argument	Description
dlrmax	Specifies the maximum refinement in the left-right dimension. Argument type: REAL
dbtmin	Specifies the minimum refinement in the bottom-top dimension. Argument type: REAL
dbtmax	Specifies the maximum refinement in the bottom-top dimension. Argument type: REAL
dbfmin	Optional. Specifies the minimum refinement in the back-front dimension. Argument type: REAL
dbfmax	Optional. Specifies the maximum refinement in the back-front dimension. Argument type: REAL
material	Optional. Specifies a material or a list of materials. Argument type: DATEXMAT DATEXMAT LIST
region	Optional. Specifies a region or a list of regions. Argument type: STRING STRING LIST
use-bbox	Optional. This argument specifies, when set to #t, that when creating 3D refinement volumes from 2D polygons, the bounding box of the polygon must be used as the basis for a cuboid volume. If set to #f, then the polygon shape itself is used to create a 3D prism using the polygon as the top and bottom faces, with rectangular sides. Note: This argument is used only for 2D polygons in 3D structures. When "use-bbox" #f is set, oversize is ignored. The default is #t. Argument type: BOOLEAN

Description

This Scheme extension supports layout-driven mesh refinements, with a syntax independent of the coordinate system of the currently active IC WorkBench simulation domain.

The Scheme extension serves as an interface between the IC WorkBench TCAD layout and the Sentaurus Structure Editor commands `sdedr:define-refeval-window`, `sdedr:define-refinement-size`, and `sdedr:define-refinement-placement` by automatically obtaining the lateral dimension of the refinement box from the specified IC WorkBench layers, taking the vertical refinement box dimensions from the argument list.

Appendix A: Commands

sdeicwb:define-refinement-from-layer

For a 3D IC WorkBench simulation domain, a refinement window is created for each polygon found in the specified layers. The lateral extent of the respective refinement windows is given by the bounding box of the polygon. The name of the refinement window has the form `RPlace.<rname>_<counter>`. The name of the refinement size setting is `RSize.<rname>`.

You can use the "material" and "region" arguments to restrict the refinement to the specified bodies within the refinement box. The "material" argument is followed by either a DATEX material name or a list of materials. For example:

```
"material" "Silicon"  
"material" (list "Silicon" "PolySilicon" ...)
```

The "region" argument is followed by either a region name or a list of region names. For example:

```
"region" "region_1"  
"region" (list "region_1" "region_2" ...)
```

The Scheme extension also generates the appropriate calls to `sdedr:define-refinement-placement` to complete the refinement definition.

You can make explicit calls to `sdedr:define-refinement-function` to activate other meshing-specific options such as refinement on doping gradients or interface refinements.

Limitations

Layout-driven refinement is available only for the area under the given layer itself, *not* for the inverse of a layer. If refinement is needed in an area not covered by the layer, you must create the inverse of the layer as an auxiliary layer explicitly in IC Validator WorkBench.

Examples

```
(sdeicwb:define-refinement-from-layer "lname" "POLY" "rname" "Channel"  
  "oversize" 0.15 "top" (sdeicwb:up SiTop 0.03)  
  "bot" (sdeicwb:down SiTop 0.1)  
  "dlrmax" 0.05 "dlrmin" 0.025 "dbtmax" 0.02 "dbtmin" 0.01  
  "dbfmax" 0.05 "dbfmin" 0.025)  
(sdedr:define-refinement-function "RSize.Channel"  
  "DopingConcentration" "MaxTransDiff" 1)  
(sdedr:define-refinement-function "RSize.Channel" "MaxLenInt"  
  "Silicon" "Oxide" 0.0002 1.41)
```

Appendix A: Commands

sdeicwb:down

sdeicwb:down

This Scheme extension increases or decreases the vertical positions depending on the coordinate system convention. It supports layout-driven mesh generation and structural operations, with a syntax independent of the coordinate system of the currently active IC WorkBench simulation domain. See [sdeicwb:up on page 739](#).

For the UCS, the x-axis points downwards. For the DF–ISE coordinate system, the y-axis points backwards and the z-axis points upwards.

Syntax

```
(sdeicwb:down value increment)
```

Returns

For the UCS, sdeicwb:down returns value + increment.

For the DF–ISE coordinate system, it returns value – increment.

Arguments

Argument	Description
value	Argument type: REAL
increment	Argument type: REAL

Examples

```
(define SiTop (sdeicwb:get-region-top (find-material-id "Silicon")))
(define epsilon 0.1)
(define AboveSiInterface (sdeicwb:up SiTop epsilon))
(define BelowSiInterface (sdeicwb:down SiTop epsilon))
```

Appendix A: Commands

sdeicwb:gds2mac

sdeicwb:gds2mac

This Scheme extension converts a GDS layout file to a .mac file, which is then loaded automatically and is ready to be processed further by other sdeicwb commands.

Syntax

```
(sdeicwb:gds2mac "gds.file" gdsfile "cell" cellname
    "layer.names" lnames "layer.numbers" lnumbers
    "sim3d" sim3ddomain "scale" scale
    "domain.name" dname "mac.file" macfile
    ["recenter" #t | #f] ["stretches" stretches]
    ["loadmacfile" #t | #f])
```

Returns

BOOLEAN

Arguments

Argument	Description
gdsfile	Specifies the name of a GDS file. Argument type: STRING
cellname	Specifies the name of a GDS cell. The keyword "cell" can be the "?" wildcard character. In this case, the active (open) GDS cell is used for the .mac file conversion. Argument type: STRING
lnames	Specifies the names of the layers (see sdeicwb:mapreader on page 737 for how to initialize this variable easily). Note: If the lengths of the lnames and lnumbers lists differ, then a warning message is added to the log file, and the file conversion is not performed. Argument type: STRING LIST
lnumbers	Specifies the corresponding layer numbers of the named layers (see sdeicwb:mapreader on page 737 for how to initialize this variable easily). Note: If the lengths of the lnames and lnumbers lists differ, then a warning message is added to the log file, and the file conversion is not performed. Argument type: STRING LIST
sim3ddomain	Specifies a real list that defines the layout area to be converted. It uses a list of four real numbers. The domain coordinates are given in (unscaled) layout coordinates. Argument type: REAL LIST

Appendix A: Commands

sdeicwb:gds2mac

Argument	Description
scale	Specifies the mask scaling factor. Argument type: REAL
dname	Specifies the name of the domain. Argument type: STRING
macfile	Specifies the name of the output .mac file. Argument type: STRING
"recenter"	Optional. Specifies whether to recenter the converted .mac file. When sdeicwb:gds2mac loads the converted .mac file, recentering is not performed by default. To center the loaded mask, use "recenter" #t. Translation of the IC WorkBench file coordinates to the (0,0) origin is switched off when "recenter" #f is set. The default is #f. Argument type: BOOLEAN
"stretches"	Optional. Specifies a list of stretches, with each stretch having a name and being defined by a segment with two points. The segment must cross the bounding box of the 3D domain. Argument type: STRING
"loadmacfile"	Optional. Specifies whether to load the converted .mac file automatically. By default, the converted .mac file is loaded in to Sentaurus Structure Editor. Set "loadmacfile" #f to suppress the automatic loading of the converted file. The default is #t. Argument type: BOOLEAN

Examples

Example 1: Layer Names and Layer Numbers Read From External *.map File

Using sdeicwb:gds2mac requires the specification of the layer names and the layer numbers from the .gds file. You can obtain this information using the sdeicwb:mapreader Scheme extension, which returns the layer names and layer numbers in a list. The first element in this list contains the layer names and the second element in this list contains the layer numbers. In this example, all layers are converted:

```
(define map-content (sdeicwb:mapreader MAPFILE))  
(define LAYERNAMES (list-ref map-content 0))  
(define LAYERNUMBERS (list-ref map-content 1))
```

where MAPFILE is the name of the GDS .map file.

```
(define GDSFILE "mygdsfile.gds")  
(define CELLNAME "gdscellname")  
(define MAPFILE "mygdsfile.map")  
(define DOMAIN "SIM3D1")  
(define map-content (sdeicwb:mapreader MAPFILE))  
(define LAYERNAMES (list-ref map-content 0))  
(define LAYERNUMBERS (list-ref map-content 1))  
(sdeicwb:gds2mac "gds.file" GDSFILE "cell" CELLNAME "layer.names")
```

Appendix A: Commands

sdeicwb:gds2mac

```
LAYERNAMES "layer.names" LAYERNUMBERS "sim3d"
(list 115 115 985 485) "scale" 1.e-3 "mac.file" "xx")
```

Example 2: User-Defined Layer Names and Layer Numbers

```
(define GDSFILE "mygdsfile.gds")
(define CELLNAME "gdscellname")
(define DOMAIN "SIM3D1")
(define LAYERNAMES (list "POLY" "CONTACT" "METALL1" "METAL2" "RECESS"
    "source" "drain" "gate"))
(define LAYERNUMBERS (list "1:0" "2:0" "3:0" "4:0" "5:0" "9:1" "9:2"
    "9:3"))
(define STRETCH "{S1 = {850 50 850 550} S2 = {50 150 1000 150}}")
(sdeicwb:gds2mac "gds.file" GDSFILE "cell" CELLNAME
    "layer.names" LAYERNAMES "layer.numbers" LAYERNUMBERS
    "sim3d" (list 115 115 985 485) "scale" 1.e-3
    "mac.file" "xx" "stretches" STRETCH)
```

The .map file format is a two-column format:

```
<layername0> <layernumber0>
<layername1> <layernumber1>
<layername2> <layernumber2>
```

Appendix A: Commands

sdeicwb:generate-mask-by-layer-name

sdeicwb:generate-mask-by-layer-name

This Scheme extension creates a mask based on the given layer names referencing an IC WorkBench macro file previously loaded using the Scheme extension (sdeicwb:load-file) and based on the IC WorkBench domain previously selected with (sdeicwb:set-domain).

Note:

The IC WorkBench macro file must have been previously loaded and the IC WorkBench domain previously selected. The IC WorkBench domain must be three dimensional.

Syntax

```
(sdeicwb:generate-mask-by-layer-name maskname layernames  
[with-overlaps do-entity-check])
```

Returns

BODYID (the ENTITY ID of the generated mask)

() (Empty list) if the layer is defined in the IC WorkBench file, but the IC WorkBench mask does not contain any mask polygon definitions

#f if the IC WorkBench file does not contain the specified layer, or if the IC WorkBench domain is not three dimensional

Arguments

Argument	Description
maskname	Specifies the name of a mask. Argument type: STRING
layernames	Specifies the name of a layer or a list of layer names. Argument type: STRING STRING LIST
with-overlaps	Optional. If the IC WorkBench mask polygons overlap, for correct handling, you must set with-overlaps to #t. In this case, the overlaps are resolved by uniting the overlapping parts of the mask polygons. The default is #f. Argument type: BOOLEAN
do-entity-check	Optional. If do-entity-check is set to #t, then an additional entity check is performed. If the entity checker finds a problem, then an additional boundary repair step tries to resolve the problem. The default is #f. Argument type: BOOLEAN

Appendix A: Commands

sdeicwb:get-back

sdeicwb:get-back

This Scheme extension returns the *back* bounding-box coordinate of the current domains in Sentaurus Structure Editor coordinates with stretches already applied. The bounding box includes all stretches applied by sdeicwb:stretch.

When sdeicwb:set-domain has set the current domains, the coordinates of the domain bounding box can be obtained.

Syntax

(sdeicwb:get-back)

Returns

REAL

sdeicwb:get-dimension

This Scheme extension returns the dimension of the current domain. The current domain defines a layout area for simulation. The dimension value as an integer is returned according to the domain type as follows:

- Point: 1
- Gauge: 2
- Highlight: 3

Note:

Sentaurus Structure Editor supports only 3D simulation domains (highlights).

Syntax

(sdeicwb:get-dimension)

Returns

INTEGER

Appendix A: Commands

sdeicwb:get-domains

sdeicwb:get-domains

This Scheme extension returns a list of current domain names.

Current domains are set with `sdeicwb:set-domain`. The list of current domains is returned with `sdeicwb:get-domains`. If only one domain is the current domain, a list of length one is returned.

Syntax

```
(sdeicwb:get-domains)
```

Returns

STRING LIST

sdeicwb:get-front

This Scheme extension returns the *front* bounding-box coordinate of the current domains in Sentaurus Structure Editor coordinates with stretches already applied. The bounding box includes all stretches applied by `sdeicwb:stretch`.

When `sdeicwb:set-domain` has set the current domains, the coordinates of the domain bounding box can be obtained.

Syntax

```
(sdeicwb:get-front)
```

Returns

REAL

Appendix A: Commands

sdeicwb:get-global-bot

sdeicwb:get-global-bot

This Scheme extension returns the bottom coordinate of the global bounding box using a coordinate system-independent syntax.

This Scheme extension supports layout-driven mesh generation and structural operations, with a syntax independent of the coordinate system used.

Use this Scheme extension to obtain the bottommost coordinate for both coordinate systems. See [sdeicwb:get-global-top](#).

Syntax

```
(sdeicwb:get-global-bot)
```

Returns

For UCS, it returns the maximum x-coordinate. For DF–ISE coordinate system, it returns the minimum z-coordinate of the global bounding box.

Examples

```
(define GBot (sdeicwb:get-global-bot))
```

sdeicwb:get-global-top

This Scheme extension returns the top coordinate of the global bounding box using a coordinate system-independent syntax.

This Scheme extension supports layout-driven mesh generation and structural operations, with a syntax independent of the coordinate system used.

Use this Scheme extension to obtain the topmost coordinate for both coordinate systems. [sdeicwb:get-global-bot](#).

Syntax

```
(sdeicwb:get-global-top)
```

Returns

For UCS, it returns the minimum x-coordinate. For the DF–ISE coordinate system, it returns the maximum z-coordinate of the global bounding box.

Examples

```
(define GTop (sdeicwb:get-global-top))
```

Appendix A: Commands

sdeicwb:get-label

sdeicwb:get-label

This Scheme extension returns a list containing information (the layer name, the text label, and the label text coordinates) that belongs to the specified label name. The layer name and the text label are strings, and the label text coordinates are two real numbers.

Syntax

```
(sdeicwb:get-label label-name)
```

Returns

LIST

Arguments

Argument	Description
label-name	Specifies the name of a label. Argument type: STRING

sdeicwb:get-label-for-layer

This Scheme extension returns a list containing label information (the layer name, the text label, and the label text coordinates) that belongs to the specified layer. The layer name and the text label are strings, and the label text coordinates are two real numbers.

Syntax

```
(sdeicwb:get-label-for-layer layer-name)
```

Returns

LIST

Arguments

Argument	Description
label-name	Specifies the name of a layer. Argument type: STRING

Appendix A: Commands

sdeicwb:get-labels

sdeicwb:get-labels

This Scheme extension returns a list, containing all the label information lists. Each returned list item contains a full label information list (the layer name, the text label, and the label text coordinates).

Syntax

(sdeicwb:get-labels)

Returns

LIST

sdeicwb:get-layer-ids

This Scheme extension returns a list of layer IDs from the IC WorkBench mask file previously loaded by sdeicwb:load-file.

Syntax

(sdeicwb:get-layer-ids)

Returns

STRING LIST

sdeicwb:get-layer-names

This Scheme extension returns a list of layer names from the IC WorkBench mask file previously loaded by sdeicwb:load-file.

Syntax

(sdeicwb:get-layer-names)

Returns

STRING LIST

sdeicwb:get-layer-polygon-midpoints

This Scheme extension computes the midpoints of the segments or polygons in a layer. It locates objects created by layout-driven operations. For example, for a layout-driven contact assignment, you might want to select a backend metal region using an (auxiliary) layer and turn all its interfaces into a contact region.

Syntax

```
(sdeicwb:get-layer-polygon-midpoints layer-name height)
```

Returns

A list of positions given by the midpoints of the polygon bounding box and the height as the third coordinate

Arguments

Argument	Description
layer-name	Specifies the name of a layer. Argument type: STRING
height	Argument type: REAL

Examples

```
(sdegeo:set-current-contact-set "drain")
(define DRAINMETAL (find-body-id (car
  (sdeicwb:get-layer-polygon-midpoints
  "ndrain" (sdeicwb:up SiTop epsilon)))))
(sdegeo:set-contact-boundary-faces DRAINMETAL)
(sdegeo:delete-region DRAINMETAL)
```

Appendix A: Commands

sdeicwb:get-left

sdeicwb:get-left

This Scheme extension returns the *left* bounding-box coordinate of the current domains in Sentaurus Structure Editor coordinates with stretches already applied. The bounding box includes all stretches applied by sdeicwb:stretch.

When sdeicwb:set-domain has set the current domains, the coordinates of the domain bounding box can be obtained.

Syntax

```
(sdeicwb:get-left)
```

Returns

REAL

sdeicwb:get-polygon-bounding-boxes-by-layer-name

This Scheme extension returns the bounding boxes for polygons on a given layer and the given current domain as segments representing the diagonal of the bounding box for that polygon. The returned values are represented as:

```
(list (list a_1 b_1) (list a_2 b_2) ... (list a_n b_n))
```

where a_n and b_n are the corner POSITIONS of the bounding box of each polygon on that layer.

Note:

The current domain must first be set using the sdeicwb:set-domain Scheme extension.

Syntax

```
(sdeicwb:get-polygon-bounding-boxes-by-layer-name layer-name)
```

Returns

POSITION LIST

Arguments

Argument	Description
layer-name	Specifies the name of a layer. Argument type: STRING

Appendix A: Commands

sdeicwb:get-polygon-by-name

sdeicwb:get-polygon-by-name

This Scheme extension returns the polygon coordinates for a given polygon name.

Polygons are named as they are read in using sdeicwb:load-file.

A list of polygon names is available on a given layer using the Scheme extension

sdeicwb:get-polygon-names-by-layer-name.

Syntax

```
(sdeicwb:get-polygon-by-name polygon-name)
```

Returns

POSITION LIST

Arguments

Argument	Description
polygon-name	Specifies the name of a polygon. Argument type: STRING

sdeicwb:get-polygon-names-by-layer-name

This Scheme extension returns the names of polygons on the given layer.

Syntax

```
(sdeicwb:get-polygon-names-by-layer-name layer-name)
```

Returns

STRING LIST

Arguments

Argument	Description
layer-name	Specifies the name of a layer. Argument type: STRING

Appendix A: Commands

sdeicwb:get-region-bot

sdeicwb:get-region-bot

This Scheme extension returns the bottom coordinate of a given region using a coordinate system-independent syntax.

This Scheme extension supports layout-driven mesh generation and structural operations, with a syntax independent of the coordinate system used.

Use this Scheme extension to obtain the bottommost coordinate of the specified region for both coordinate systems. See [sdeicwb:get-region-top on page 731](#).

Syntax

```
(sdeicwb:get-region-bot body)
```

Returns

For the UCS, it returns the maximum x-coordinate. For the DF–ISE coordinate system, it returns the minimum z-coordinate of the region bounding box.

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY

Examples

```
(define SubstrateID (find-material-id "Silicon"))
(define SiBottom (sdeicwb:get-region-bot SubstrateID))
(define PolyID (find-material-id "PolySilicon"))
(define PoBottom (sdeicwb:get-region-bot PolyID))
```

Appendix A: Commands

sdeicwb:get-region-top

sdeicwb:get-region-top

This Scheme extension returns the top coordinate of a given region using a coordinate system-independent syntax.

This Scheme extension supports layout-driven mesh generation and structural operations, with a syntax independent of the coordinate system used.

Use this Scheme extension to obtain the topmost coordinate of the specified region for both coordinate systems. See [sdeicwb:get-region-bot on page 730](#).

Syntax

```
(sdeicwb:get-region-top body)
```

Returns

For the UCS, it returns the minimum x-coordinate. For the DF–ISE coordinate system, it returns the maximum z-coordinate of the region bounding box.

Arguments

Argument	Description
body	Specifies a body. Argument type: BODY

Examples

```
(define SubstrateID (find-material-id "Silicon"))
(define SiTop (sdeicwb:get-region-top SubstrateID))

(define PolyID (find-material-id "PolySilicon"))
(define PoTop (sdeicwb:get-region-top PolyID))
```

Appendix A: Commands

sdeicwb:get-right

sdeicwb:get-right

This Scheme extension returns the *right* bounding-box coordinate of the current domains in Sentaurus Structure Editor coordinates with stretches already applied.

When sdeicwb:set-domain has set the current domains, the coordinates of the domain bounding box can be obtained.

Calling the Scheme extension sdeicwb:get-right returns the *right* coordinate. The bounding box includes all stretches applied by sdeicwb:stretch.

Syntax

(sdeicwb:get-right)

Returns

REAL

sdeicwb:get-xmax

This Scheme extension returns the ‘xmax’ bounding-box coordinate of the current domains in IC WorkBench coordinates with stretches already applied.

When sdeicwb:set-domain has set the current domains, the ‘xmax’ coordinate of the domain bounding box is returned in IC WorkBench coordinates by calling sdeicwb:get-xmax. The bounding box includes all stretches applied by sdeicwb:stretch.

Syntax

(sdeicwb:get-xmax)

Returns

REAL

Appendix A: Commands

sdeicwb:get-xmin

sdeicwb:get-xmin

This Scheme extension returns the ‘xmin’ bounding-box coordinate of the current domains in IC WorkBench coordinates with stretches already applied.

When sdeicwb:set-domain has set the current domains, the ‘xmin’ coordinate of the domain bounding box is returned in IC WorkBench coordinates by calling sdeicwb:get-xmin. The bounding box includes all stretches applied by sdeicwb:stretch.

Syntax

(sdeicwb:get-xmin)

Returns

REAL

sdeicwb:get-ymax

This Scheme extension returns the ‘ymax’ bounding-box coordinate of the current domains in IC WorkBench coordinates with stretches already applied.

When sdeicwb:set-domain has set the current domains, the ‘ymax’ coordinate of the domain bounding box is returned in IC WorkBench coordinates by calling sdeicwb:get-ymax. The bounding box includes all stretches applied by sdeicwb:stretch.

Syntax

(sdeicwb:get-ymax)

Returns

REAL

Appendix A: Commands

sdeicwb:get-ymin

sdeicwb:get-ymin

This Scheme extension returns the ‘ymin’ bounding-box coordinate of the current domains in IC WorkBench coordinates with stretches already applied.

When sdeicwb:set-domain has set the current domains, the ‘ymin’ coordinate of the domain bounding box is returned in IC WorkBench coordinates by calling sdeicwb:get-ymin. The bounding box includes all stretches applied by sdeicwb:stretch.

Syntax

(sdeicwb:get-ymin)

Returns

REAL

sdeicwb:get-zmax

This Scheme extension returns the ‘zmax’ bounding-box coordinate of the current domains in IC WorkBench coordinates with stretches already applied.

When sdeicwb:set-domain has set the current domains, the ‘zmax’ coordinate of the domain bounding box is returned in IC WorkBench coordinates by calling sdeicwb:get-zmax. The bounding box includes all stretches applied by sdeicwb:stretch.

Syntax

(sdeicwb:get-zmax)

Returns

REAL

Appendix A: Commands

sdeicwb:get-zmin

sdeicwb:get-zmin

This Scheme extension returns the ‘zmin’ bounding-box coordinate of the current domains in IC WorkBench coordinates with stretches already applied.

When sdeicwb:set-domain has set the current domains, the ‘zmin’ coordinate of the domain bounding box is returned in IC WorkBench coordinates by calling sdeicwb:get-zmin. The bounding box includes all stretches applied by sdeicwb:stretch.

Syntax

(sdeicwb:get-zmin)

Returns

REAL

Appendix A: Commands

sdeicwb:load-file

sdeicwb:load-file

This Scheme extension loads an IC WorkBench macro file.

The IC WorkBench macro file must be read in using `sdeicwb:load-file` as a prerequisite to other `sdeicwb` Scheme extensions that act on the domains and masks defined in that file.

All IC WorkBench coordinates in the macro file are multiplied by the optional `scale` argument as they are read.

Caution:

The IC WorkBench macro file must be readable.

If the `.mac` file is already loaded, it is not reloaded by `sdeicwb:load-file`. If you used `sdeicwb:gds2mac` to convert and load a layout file, different scaling and centering specified in the `sdeicwb:load-file` Scheme extension have no effect.

Syntax

```
(sdeicwb:load-file filename [(scale | center?) | (scale center?)]))
```

Returns

Returns `#t` if successful. Otherwise, `#f` if the file cannot be read.

Arguments

Argument	Description
filename	Specifies a macro file to load. Argument type: STRING
scale	Optional. Specifies a scaling factor. Argument type: REAL
center?	Optional. Specifies whether to center the loaded mask. Set <code>center?</code> to <code>#t</code> to center the loaded mask. If it is set to <code>#t</code> , then the mask vertex coordinates are modified by the value of the vertex coordinate <i>minus</i> the value of the highlight coordinate. Argument type: BOOLEAN

Appendix A: Commands

sdeicwb:mapreader

sdeicwb:mapreader

This Scheme extension parses a .map file, and returns the layer names and the layer numbers. The first returned list is the layer name list, and the second is the layer number list.

Syntax

```
(sdeicwb:mapreader filename)
```

Returns

List of two STRING lists

Arguments

Argument	Description
filename	Specifies the name of a file. Argument type: STRING

Description

After sdeicwb:mapreader is called, you can call sdeicwb:gds2mac with the layernames and layernumbers arguments, so that when the map file is available, there is no need to add the layernames and numbers manually to the sdeicwb:gds2mac call (see [sdeicwb:gds2mac on page 718](#)).

Map File Format

The .map file has a two-column format (<layername> <space or tab> <layernumber>):

```
<layername0> <layernumber0>
<layername1> <layernumber1>
<layername2> <layernumber2>
```

If the layer .map file is exported from a third-party layout editor, then you might have to edit the file manually. The exported .map file might contain some other information and might be in a different format than the required two-column format.

Examples

```
(define map-content (sdeicwb:mapreader "layer.map"))
(define layernames (list-ref map-content 0))
(define layernumbers (list-ref map-content 1))
```

Then, the sdeicwb:gds2mac command can be called with the layernames and layernumbers arguments.

Appendix A: Commands

sdeicwb:set-domain

sdeicwb:set-domain

This Scheme extension sets the current domain to the given domain name or list of domain names.

Setting the current domain is a prerequisite for other sdeicwb Scheme extensions that implicitly depend on the current domain being defined.

Syntax

```
(sdeicwb:set-domain domain)
```

Returns

None

Arguments

Argument	Description
domain	Specifies the name of a domain or a list of domain names. Argument type: STRING STRING LIST

sdeicwb:stretch

This Scheme extension applies a given stretch by a given amount to the current domains. The order of applied stretches is important because the location of other stretches can change given the application of one stretch.

Syntax

```
(sdeicwb:stretch stretch-name stretch-amount)
```

Returns

None

Arguments

Argument	Description
stretch-name	Specifies the stretch. Argument type: STRING
stretch-amount	Specifies the amount of stretch. Argument type: REAL

Appendix A: Commands

sdeicwb:up

sdeicwb:up

This Scheme extension increases or decreases the vertical positions depending on the coordinate system convention. It supports layout-driven mesh generation and structural operations, with a syntax independent of the coordinate system of the currently active IC WorkBench simulation domain. See [sdeicwb:down on page 717](#).

For the UCS, the x-axis points downwards. For the DF–ISE coordinate system, the y-axis points backwards and the z-axis points upwards.

Syntax

```
(sdeicwb:up value increment)
```

Returns

For the UCS, it returns value – increment. For the DF–ISE coordinate system, it returns value + increment.

Arguments

Argument	Description
value	Argument type: REAL
increment	Argument type: REAL

Examples

```
(define SiTop (sdeicwb:get-region-top (find-material-id "Silicon")))
(define epsilon 0.1)
(define AboveSiInterface (sdeicwb:up SiTop epsilon))
(define BelowSiInterface (sdeicwb:down SiTop epsilon))
```

Appendix A: Commands

sdeio:read-dfise-mask

sdeio:read-dfise-mask

This Scheme extension loads the specified layout file (.lyt) into the modeler and initializes the data structure that defines the mask layouts inside Procem.

During process emulation, the initialized mask layouts can be used in pattern operations. Mask polarities are not fixed; each mask can be used either as a 'light' or 'dark' field mask.

Syntax

```
(sdeio:read-dfise-mask filename)
```

Returns

BOOLEAN (#t for success; #f for failure)

Arguments

Argument	Description
filename	Specifies the name of the mask layout file, with the file extension .lyt. It can also contain a path. Argument type: STRING

Appendix A: Commands

sdeio:read-tdr

sdeio:read-tdr

This Scheme extension extracts the boundary and loads a TDR boundary file into Sentaurus Structure Editor.

Syntax

```
(sdeio:read-tdr file-name ["skip-doping"])
```

Returns

None

Arguments

Argument	Description
file-name	Specifies the name of a TDR boundary file. Argument type: STRING
"skip-doping"	Optional. Specify this option to omit doping information contained in the TDR file. Argument type: STRING

Description

This Scheme extension performs the following operations:

1. It takes a TDR mesh file and extracts the boundary, unless the TDR file already contains the boundary as geometry (which is typical of TDR files saved by Sentaurus Process and Sentaurus Interconnect).
2. It loads the boundary.
3. It creates a submesh definition and placement using the doping information contained in the TDR file, unless you have specified "skip-doping".

Appendix A: Commands

sdeio:read-tdr-bnd

sdeio:read-tdr-bnd

This Scheme extension loads a TDR boundary file into Sentaurus Structure Editor.

Each TDR region element is converted to an ACIS body. For 2D models, contact attributes are attached to the corresponding edges of the ACIS model and to the corresponding faces for 3D models. Two-dimensional regions are converted to nonuniform sheet bodies, while 3D regions are converted to manifold solids. The material properties and region names are attached as attributes to the ACIS bodies.

Syntax

(sdeio:read-tdr-bnd file-name)

Returns

None

Arguments

Argument	Description
file-name	Specifies the name of a TDR boundary file. Argument type: STRING

Appendix A: Commands

sdeio:save-1d-tdr-bnd

sdeio:save-1d-tdr-bnd

This Scheme extension creates a 1D TDR boundary output file extracted from a simple 2D geometry along the *up* coordinate axis defined by (`sde:set-process-up-direction`). See [sde:set-process-up-direction on page 501](#).

The 1D geometry is extracted from the entities listed in `body-list` in the *up* direction along the corresponding axis (at $x=0$ in the vertical direction, or $y=0$ in the horizontal direction). The 2D geometry should be a simple 2D representation of the required 1D geometry, extruded in the corresponding direction.

The Scheme extension `sde:set-process-up-direction` defines the *up* direction – vertical versus horizontal – and takes either "`-x`" (default) or "`+z`". The default value corresponds to the up direction being along the x -axis. The value "`+z`" corresponds to the up direction being along the z -axis.

The output file is extracted from the entities listed in `body-list`. The 2D geometry is tessellated first, using the same process as in two dimensions. These parameters are described in [sdeio:save-tdr-bnd on page 745](#).

Syntax

```
(sdeio:save-1d-tdr-bnd body-list file-name
  [ "aspect ratio" aspect-ratio]
  [ "aut-tolerance-control" aut-tolerance-control]
  [ "faceter" faceter]
  [ "max edge length" max-edge-length]
  [ "normal tolerance" normal-tolerance]
  [ "surface tolerance" surface-tolerance]
  [ "vertex-precision" vertex-precision]
  [ writeinterface saveprecision])
```

Returns

Returns #t if the boundary is saved successfully; otherwise, returns #f

Arguments

Argument	Description
body-list	Specifies a body list. Alternatively, you can specify "all". Argument type: BODY LIST
file-name	Specifies the name of a TDR boundary file. Argument type: STRING
aspect-ratio	Optional. Specifies the aspect ratio. The default is 0. Argument type: REAL

Appendix A: Commands

sdeio:save-1d-tdr-bnd

Argument	Description
aut-tolerance-control	Optional. Specifies whether to use automatic tolerance control. The default is #t. Argument type: BOOLEAN
faceter	Optional. Specifies the faceter to use. Options are "v1" "v2". The default is "v1". Argument type: STRING
max-edge-length	Optional. Specifies the maximum edge length. The default is 0. Argument type: REAL
normal-tolerance	Optional. Specifies the normal tolerance. The default is 15. Argument type: REAL
surface-tolerance	Optional. Specifies the surface tolerance. The default is 0. Argument type: REAL
vertex-precision	Optional. Specifies the vertex precision in the range [8-16]. The default is bndprecision. Argument type: INTEGER
writeinterface	Optional. Specifies whether to write the interface. Argument type: BOOLEAN
saveprecision	Optional. Specifies the precision with which to save the boundary. Argument type: INTEGER

Examples

```
; Create three regions in 2D bordering the y-axis
(sdegeo:create-rectangle (position 0 0 0) (position 1 5 0)
    "Silicon" "body")
(sdegeo:create-rectangle (position 0 -0.1 0) (position 1 0 0)
    "Oxide" "gateox")
(sdegeo:create-rectangle (position 0 -1 0) (position 1 -0.1 0)
    "PolySilicon" "gatepoly")

; Add contacts to top and bottom of the structure
(sdegeo:define-contact-set "gate" 4 (color:rgb 1 0 0) "##")
(sdegeo:define-contact-set "substrate" 4 (color:rgb 1 0 0) "##")
(sdegeo:set-current-contact-set "substrate")
(sdegeo:set-contact-edges (list (car (find-edge-id
    (position 0.5 5 0)))) "substrate")
(sdegeo:set-current-contact-set "gate")
(sdegeo:set-contact-edges (list (car (find-edge-id
    (position 0.5 -1 0)))) "gate")

; Save 1D extraction to boundary file in TDR format
(sdeio:save-1d-tdr-bnd (get-body-list) "1D.tdr")
```

Appendix A: Commands

sdeio:save-tdr-bnd

sdeio:save-tdr-bnd

This Scheme extension generates an output TDR boundary file.

Note:

The order of arguments in the argument list is arbitrary. However, the first two arguments must be body-list file-name. Each keyword must be followed by a value.

Syntax

```
(sdeio:save-tdr-bnd body-list file-name
  ["aspect ratio" aspect-ratio]
  ["aut tolerance control" aut-tolerance-control]
  ["faceter" faceter] ["grid mode" grid-mode]
  ["max edge length" max-edge-length]
  ["normal tolerance" normal-tolerance]
  ["separate lumps" separate-lumps]
  ["surface tolerance" surface-tolerance]
  ["triangulate" triangulate]
  ["unique region names" unique-region-names]
  ["vertex precision" vertex-precision])
```

Returns

(BOOLEAN, faceter)

Arguments

Argument	Description
body-list	Specifies a body list. Alternatively, you can specify "all", in which case, all geometry regions are saved in the TDR boundary file. Argument type: ENTITY LIST
file-name	Specifies the name of a TDR boundary file. Argument type: STRING
aspect-ratio	Optional. Sets the aspect ratio of the triangles in the surface tessellation. The default is 0, which means no restriction. Argument type: REAL
aut-tolerance-control	Optional. Specifies whether to use automatic tolerance control. The default is aut-adjust-refinement-tolerance. The aut-adjust-refinement-tolerance global Scheme variable is set to #f by default. Argument type: BOOLEAN

Appendix A: Commands

sdeio:save-tdr-bnd

Argument	Description
faceter	<p>Optional. Specifies the faceter to use. Options are "v1" "v2", which are different algorithms that can be used for the tessellation. The default is "v1". Argument type: STRING</p> <p>The "v2" faceter satisfies the refinement criteria (surface and normal tolerances) more precisely than the "v1" faceter; therefore, it creates denser meshes and more elements in the tessellated boundary file.</p> <p>The "v2" faceter creates a better tessellation for spline faces than the "v1" faceter. However, if layering is applied in Sentaurus Mesh, then using the "v1" faceter in Sentaurus Structure Editor is recommended to avoid a possible overrefinement in the bulk by the layering operation.</p>
grid-mode	<p>Optional. Specifies the grid mode. Options are "AF_GRID_ONE_DIR" "AF_GRID_TO_EDGES".</p> <p>The default "grid mode" "AF_GRID_ONE_DIR" is used to avoid poles for spherical faces.</p> <p>Argument type: STRING</p>
max-edge-length	Optional. Specifies the maximum edge length. The default is 0, which deactivates this control. Argument type: REAL
normal-tolerance	Optional. Specifies the normal tolerance. The default is 15. Argument type: REAL
separate-lumps	Optional. Specifies whether to separate lumps. The default is #t. Argument type: BOOLEAN
surface-tolerance	Optional. Specifies the surface tolerance. The default is 0, which deactivates this control. Argument type: REAL
triangulate	<p>Optional. Specifies whether to triangulate. The default is #t.</p> <p>Note: This argument works only for 3D models with planar faces and linear edges.</p> <p>Argument type: BOOLEAN</p>
unique-region-names	Optional. The default is #t. Argument type: BOOLEAN
vertex-precision	Optional. Specifies an integer ([8–16]) indicating how many decimal digits the vertex coordinate list in the tessellated boundary file should contain. The default is bndprecision (14). Argument type: REAL

Appendix A: Commands

sdeio:save-tdr-bnd

Description

Depending on model dimensionality, this Scheme extension generates a 2D or 3D TDR boundary file. Model dimensionality is determined by the solids in the `body-list` argument. Mixed dimensionality is not supported, so all entities in `body-list` must be either 2D or 3D solid entities.

By default in three dimensions, the model boundary is triangulated and the triangulated boundary (satisfying all the specified refinement criteria) is saved to the TDR boundary file. If you set "`triangulate`" `#f` and only planar faces are present with linear edges, the TDR boundary contains only the face polygons; no triangulation is performed. In this case, all other controls are ignored.

By default, before the TDR boundary file is written, the structure is triangulated, that is, polygonal faces are split into multiple triangles. For very complex geometries, triangulation can take a considerable amount of time, but the triangulation can be switched off by setting "`triangulate`" `#f`. Consequently, face polygons are written to the TDR boundary file as polygons.

However, you can use "`triangulate`" `#f` only if the model is polyhedral. (Only planar faces and planar edges are supported.)

One restriction for "`triangulate`" `#f` is that polyhedron faces cannot contain internal holes. (Only one edge loop per face is allowed.)

If you set "`aut tolerance control`" `#f` (default), the faceter uses the user-specified tolerance controls. If you set "`aut tolerance control`" `#t`, the value of "`surface tolerance`" changes if the model contains nonplanar faces and small edges, with the edge length smaller than twice the surface tolerance. In this case, the surface tolerance refinement control is adjusted automatically to half of the smallest edge length. This automatic adjustment avoids having intersecting boundaries in the tessellated boundary file. (This can occur when very thin curved regions are present in the model and the surface tolerance is greater than the thickness of the region.)

The global Scheme variable `aut-adjust-refinement-tolerance` can also be used to control the automatic readjustment of the surface tolerance. The default value of this variable is `#f`.

The (`set! aut-adjust-refinement-tolerance #t`) command switches on automatic setting of the surface tolerance.

When the tessellated boundary file is saved, the refinement control values (surface, normal tolerance, and so on) are satisfied by the surface triangulation.

The surface tessellation controls "`surface tolerance`", "`aspect ratio`", "`normal tolerance`", and "`max edge length`" affect the quality of the surface triangulation. Smaller values result in a denser surface triangulation for curved faces. For planar faces, "`surface tolerance`" and "`normal tolerance`" do not change the triangulation.

Appendix A: Commands

sdepe:add-substrate

sdepe:add-substrate

This Scheme extension adds an initial substrate layer to the device. The domain boundary must be set before this function is called.

See [sdepe:pattern on page 766](#).

Syntax

```
(sdepe:add-substrate "material" material "thickness" thickness  
  ["base" base] ["region" region-name])
```

Returns

ENTITY (BODY)

Arguments

Argument	Description
material	Identifies the DATEX material of the generated body. Argument type: DATEXMAT
thickness	Specifies the thickness of the substrate layer. Argument type: REAL
base	Optional. Assigns a value for the bottom z-coordinate of the substrate layer for process-up-direction "-x" or the top x-coordinate of the substrate layer for process-up-direction "+z". If you do not specify "base", then the substrate layer is placed such that the bottom is placed at z=0 for process-up-direction "-x" or at x=0 for process-up-direction "+z". Argument type: REAL
region-name	Optional. Assigns an explicit region name to the generated substrate. Argument type: STRING

Appendix A: Commands

sdepe:clean

sdepe:clean

This Scheme extension cleans the model before a process emulation operation. It regularizes all the bodies and fixes the imprints. It can be used to simplify or clean the Procem model, especially if the input model is generated by other tools and a tessellated boundary is imported to Sentaurus Structure Editor first, as the initial structure.

Syntax

(sdepe:clean)

Returns

None

Appendix A: Commands

sdepe:define-pe-domain

sdepe:define-pe-domain

This Scheme extension defines a base domain for the process emulation. It creates the simulation domain for subsequent Procem operations.

Syntax

```
(sdepe:define-pe-domain {{x0 y0 x1 y1} | {polygon}})
```

Returns

None

Arguments

Argument	Description
x0 y0 x1 y1	Specify four real numbers to define the simulation domain. If the argument list contains four real numbers, then a rectangle is defined, using the four numbers as the xy coordinates as the two opposite corners of the simulation domain. Argument type: REAL
polygon	Specifies a list of real pairs to define the simulation domain. If you define a list of real pairs, then these pairs define the vertex points of the simulation domain in the xy plane for process-up-direction "+z" and in the yz plane for process-up-direction "-x". If a polygon is used to define the simulation domain, then the only restriction is that the polygon must be a simply connected convex polygon. The domain is defined in the xy plane at z=0 for process-up-direction "+z", and the domain is defined in the yz plane at x=0 for process-up-direction "-x". Argument type: REAL LIST

Description

The argument list can be either four real numbers or a list of real pairs.

This Scheme extension creates a wire body, representing the simulation domain. This wire body is called `domainboundary`, and this global variable can be accessed directly from Scheme. The `domainboundary` is an artificial object that is needed for other Procem commands to identify the simulation domain.

The `domainboundary` is not written separately to the output boundary file. If, after some Procem operations, you want to change the simulation domain, `domainboundary` must be deleted using the command (`entity:delete domainboundary`). This might be necessary, for example, if the model was reflected after some Procem operations, which changed the bounding box of the model.

Appendix A: Commands

sdepe:define-pe-domain

If a 3D model is already defined, Procem operations can be applied directly, without the need to define the domain boundary explicitly. In this case, the domain boundary is computed automatically from the existing model when the first Procem operation is used. The domain boundary is computed as the xy cross section of the model at the minimal z-position.

Note:

The process emulation commands operate on the entire device and cannot be restricted to operate only on a certain part (defined by a rectangular domain). However, if you want to modify the process emulation domain, use the `sdegeo:chop-domain` command to remove the unwanted part of the model. After the unwanted part is removed, the previously computed domain boundary must be deleted using the `(entity:delete domainboundary)` command.

Examples

```
; to define a polygonal simulation domain use:  
(sdepe:define-pe-domain (list 0 0 10 0 15 2.5 10 5 0 5))  
; or equivalently  
(sdepe:define-pe-domain 0 0 10 0 15 2.5 10 5 0 5)  
; to define a rectangular simulation domain use:  
(sdepe:define-pe-domain (list 0 0 10 5))  
; or equivalently  
(sdepe:define-pe-domain 0 0 10 5)
```

Appendix A: Commands

sdepe:depo

sdepe:depo

This Scheme extension performs a deposition step (see [Deposition on page 193](#)). For details about the arguments, see [Table 29 on page 206](#).

Note:

The order of arguments in the argument list is not relevant, but the correct data values must always follow the keywords. Certain options might be contradictory, for example, the "lop_x" algorithm cannot be used for anisotropic deposition, and the "PT" algorithm cannot be used with rounding. In addition, certain arguments must be specified together. The Scheme extension sdepe:depo performs a preprocessing consistency check of the arguments. If a keyword does not match, then an error message is displayed and the function execution is terminated.

Syntax

```
(sdepe:depo "material" material "thickness" thickness
  [ "region" region-name] [ "type" depo-type] [ "algorithm" depo-alg]
  ( [ "radius" radius] [ "vexity" vexity-type]
    [ "advanced-rounding" advrounding] | [ "rounding" rounding] )
  ( [ "chamfer" cutback-distance] [ "vexity" vexity-type] )
  ( [ "taper-angle" taper-angle] [ "taper-direction" taper-direction] )
  ( [ "initial-thickness" initial-thickness]
    [ "selective-material" selective-material]
    [ "selective-taper" selective-taper] )
  ( [ "ray-vector" ray-vector] [ "BC" BC] [ "ext-dist" ext-dist]
    [ "shadowing" shadowing] )
  [ "steps" nsteps] [ "adaptive" adaptive] [ PT keywords and values])
```

Returns

ENTITY of the generated deposit layer

Arguments

Argument	Description
material	Argument type: DATEXMAT
thickness	Argument type: REAL
region-name	Optional. Argument type: STRING
depo-type	Optional. Specifies the deposition type. Options are "iso" "aniso". Argument type: STRING

Appendix A: Commands

sdepe:depo

Argument	Description
depo-alg	Optional. Specifies the deposition algorithm to use. Options are "lopx" "lop-move" "sweep" "PT". Argument type: STRING
radius	Optional. Specifies the radius. Argument type: REAL
vexity-type	Optional. Specifies the vexity type. Options are "convex" "concave" "all". Argument type: VEXITY TYPE
advrounding	Optional. Argument type: BOOLEAN
rounding	Optional. Argument type: BOOLEAN
chamfer	Optional. Argument type: REAL
taper-angle	Optional. Argument type: REAL
taper-direction	Optional. Specifies the taper direction as a list as follows: <ul style="list-style-type: none"> • (list "x" "-x" "y" "-y") (or any combination of these directions) for process-up-direction "+z" • (list "y" "-y" "z" "-z") (or any combination of these directions) for process-up-direction "-x"
initial-thickness	Optional. Specifies the initial thickness, between (0-1). Argument type: REAL
selective-material	Optional. Argument type: DATEXMAT
selective-taper	Optional. Argument type: BOOLEAN
ray-vector	Optional. Argument type: GVECTOR
BC	Optional. Specifies the boundary conditions. Options are "none" "Reflect" "Periodic" "ext-dist". Argument type: STRING
ext-dist	Optional. Argument type: REAL
shadowing	Optional. Argument type: BOOLEAN
nsteps	Optional. Specifies the number of steps, which must be equal to or greater than 1. Argument type: INTEGER
adaptive	Optional. Argument type: BOOLEAN
PT keywords and values	Optional. See Table 26 on page 188 .

sdepe:doping-constant-placement

This Scheme extension defines and applies a constant doping to a given region.

In the case of constant doping, a separate `implant` command is not necessary, as is the case with Gaussian doping profiles.

Syntax

```
(sdepe:doping-constant-placement name species concentration  
    region-name)
```

Returns

None

Arguments

Argument	Description
name	Argument type: STRING
species	Argument type: STRING
concentration	Argument type: REAL
region-name	Argument type: STRING

Appendix A: Commands

sdepe:etch-material

sdepe:etch-material

This Scheme extension performs an etching step operation.

The etching step is very similar to the deposition step, except that the deposition is performed on a complementary body. The deposition is performed on the complementary body and, as a last step, this deposited layer is removed from the original device (only using the bodies with the specified material attribute). All regions with a different material attribute that are originally on top of the material being etched will protect the etch material underneath.

For detailed descriptions of the available keywords and effects, see [sdepe:depo on page 752](#).

Note:

The deposition offset is specified using the "thickness" attribute, and the etch offset is specified using the "depth" attribute. This is the only difference between the argument names for these two Scheme extensions (`sdepe:depo` and `sdepe:etch-material`).

Syntax

```
(sdepe:etch-material "material" material "depth" depth
+ all deposition arguments for sdepe:depo
["overetch" overetch] ["ebl" ebl] [PT keywords and values])
```

Returns

None

Arguments

Argument	Description
material	Argument type: DATEXMAT
depth	Argument type: REAL
overetch	Optional. Specifies an offset by which to move the vertical etch faces. Argument type: REAL
ebl	Optional. Determines during etching whether only the initially exposed bodies (with the given etching material) are affected (if "ebl" "top" is specified) or whether initially unexposed layers of the specified materials are affected (if "ebl" "all" is specified). The default is "ebl" "all". Argument type: STRING

Appendix A: Commands

sdepe:extend-masks

Argument	Description
PT keywords and values	Optional. See Table 26 on page 188 .
+ all deposition arguments for sdepe:depo	See sdepe:depo on page 752 .

sdepe:extend-masks

This Scheme extension extends all masks about the boundary, which is specified by pminx, pminy, pmaxx, and pmaxy. All existing masks are affected.

Syntax

```
(sdepe:extend-masks pminx pminy pmaxx pmaxy extension-distance)
```

Returns

BOOLEAN

Arguments

Argument	Description
pminx pminy pmaxx pmaxy	Specify the boundary. Argument type: REAL
extension-distance	Specifies the extension distance. Argument type: REAL

Examples

```
(sde:clear)  
(sdepe:generate-mask "M1" (list (list 0 0 10 5)))  
(sdepe:extend-masks 0 0 10 5 2)
```

```
(sde:clear)  
(sdepe:generate-mask "M1" (list (list 0 0 10 5)))  
(sdepe:extend-masks 0 0 5 2 2)  
(sdepe:extend-masks 6 3 10 5 2)
```

Appendix A: Commands

sdepe:fill-device

sdepe:fill-device

This Scheme extension fills the device with the specified material up to the specified height.

Syntax

```
(sdepe:fill-device "material" material  
                   ["height" height] ["region" region-name])
```

Returns

ENTITY of the generated fill region

Arguments

Argument	Description
material	Specifies the DATEX material with which to fill the device. Argument type: DATEXMAT
height	Optional. Specifies the height. If this argument is not specified, then the Scheme extension fills up the device to the top position. Argument type: REAL
region-name	Optional. Specifies a name for the filled region. Argument type: STRING

Appendix A: Commands

sdepe:generate-domainboundary

sdepe:generate-domainboundary

This Scheme extension extracts the simulation boundary from an existing model.

When Procem is started from an empty model, one of the first commands should be (sdepe:define-pe-domain) to define the domain boundary. All subsequent functions work on that domain.

If you want to perform additional process steps on a model, this step is not needed. It is sufficient to call (sdepe:generate-domainboundary), which extracts the domainboundary wire from the model. (The same restrictions apply here as for (sdepe:define-pe-domain).) The extracted boundary must be convex and can contain only one lump. The Scheme extension initializes the Scheme variable domainboundary (wire body).

Syntax

```
(sdepe:generate-domainboundary)
```

Returns

None

Examples

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:set-default-boolean "AB")
(sdegeo:create-cylinder (position 0 0 0) (position 0 0 1) 5
    "Silicon" "region_1")
(sdegeo:create-cuboid (position 0 -5 0) (position 15 5 1)
    "Silicon" "region_2")
(sdepe:generate-domainboundary)
(sdepe:depo "PolySilicon" 0.5)
```

sdepe:generate-empty-mask

This Scheme extension creates a new empty mask.

Empty masks behave the same way in Procem operations as *regular* masks, except no geometry is attached to the empty masks. The name of the empty mask is registered, and query functions can be used to find it (`get-empty-mask-list`, `exists-empty-mask-name`).

When mask polarity "light" is used, an empty mask behaves like a regular mask that is placed completely outside of the simulation domain.

When mask polarity "dark" is used, an empty mask behaves like a regular mask that covers the complete simulation domain.

Syntax

```
(sdepe:generate-empty-mask maskname)
```

Returns

BOOLEAN

Arguments

Argument	Description
maskname	Specifies the name of a mask. Argument type: STRING

Examples

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 4)
    "Silicon" "base")
(sdepe:generate-empty-mask "M1")
(sdepe:pattern "mask" "M1" "thickness" 1 "material" "Resist")
(sdepe:pattern "mask" "M1" "thickness" 1 "material" "Resist"
    "polarity" "dark")
```

Appendix A: Commands

sdepe:generate-mask

sdepe:generate-mask

This Scheme extension creates a mask for further process operations in Procem.

Syntax

```
(sdepe:generate-mask mask-name polygon-list)
```

Returns

None

Arguments

Argument	Description
mask-name	Specifies the name of the generated mask. Argument type: STRING
polygon-list	Specifies the vertex points for the generated mask. Argument type: POLYGON LIST

Description

The defined masks can be used in further patterning operations. (The used masks are not deleted.) Each mask is defined as a polygon list. When the polygon list is converted to a mask, first each polygon is converted to a 2D sheet body. Then, a containment check is performed, and the partially overlapping converted sheet bodies are united. If a sheet body is completely contained inside another body, it will be subtracted from that body and an internal void will form. In this way, multiply connected mask sheet bodies can be created.

A single mask polygon list cannot contain intersecting edges, and the polygon vertex list must define a manifold sheet body for each polygon. (No cutlines or dangling edges are allowed.) If the vertex list is not closed, the converter will close it automatically. Zero length edges are not allowed in the polygon vertex list (that is, there should be at least a 1.0e-06 distance between the vertices). Different masks can have intersecting polygonal boundaries. If a mask contains rectangular polygons, it is sufficient to specify only two opposite corners of such polygons. Each polygon is defined as an xy coordinate list.

The generated mask will be a 2D sheet body, and a ‘mask name’ `mask-name` attribute will be attached to the mask body. If a mask with the specified name already exists, subsequent operations will use the newly created mask.

Appendix A: Commands

sdepe:generate-mask

Examples

```
(sde:clear)
(sdepe:generate-mask "DEMO-MASK-1" (list (list 2 2 5 2 5 3 2 3)
                                         (list 5 6 9 6 7 9)))
(sdepe:generate-mask "DEMO-MASK-2" (list (list 14 7 13.73 8 13 8.73
                                             12 9 11 8.73 10.27 8 10 7 10.27 6 11 5.27 12 5 13 5.23 13.73 6)))
(sdepe:generate-mask "DEMO-MASK-3" (list (list 11 3 10.73 4 10 4.73 9
                                             5 8 4.73 7.27 4 7 3 7.27 2 8 1.27 9 1 10 1.23 10.73 2)))

(sde:clear)
(sdepe:generate-mask "MASK1" (list (list 1 1 3 3) (list 7 7 9 9)))
(sdepe:generate-mask "MASK2" (list (list 7 1 9 3 3 9 1 7)))
(sdepe:generate-mask "MASK3" (list (list 2 1 3 2 2 3 1 2)
                                   (list 8 1 9 2 8 3 7 2) (list 8 7 9 8 8 9 8)
                                   (list 2 7 3 8 2 9 1 8)))
(sdepe:generate-mask "MASK4" (list (list 3 3 7 7)))
(sdepe:generate-mask "MASK5" (list (list 3 0 8 2) (list 8 3 10 7)
                                   (list 3 8 7 10) (list 0 3 2 7)))
```

Appendix A: Commands

sdepe:icon_layer

sdepe:icon_layer

This Scheme extension creates interconnect structures. It can be used like other sdepe Scheme extensions.

The arguments must be added as keyword-value pairs. The "mask" and "thickness" arguments must be specified.

If the region names are not given, then different layers will have the same region names ("ic-region" and "env-region"). Before the generation of boundary output, these regions can be united or renamed to have unique region names.

Syntax

```
(sdepe:icon_layer "mask" mname "thickness" lth
  ["polarity" mpolarity] ["taper-angle" tang] ["ic-material" icmat]
  ["env-material" envmat] ["ic-region" icr] ["env-region" envr]
  ["base-coord" bc] ["taper-position" tpos])
```

Returns

None

Arguments

Argument	Description
mname	Specifies the name of the mask. The default is "". Argument type: STRING
lth	Specifies the layer thickness. The default is 0. Argument type: REAL
mpolarity	Optional. Specifies the mask polarity. Options are "light" "dark". The default is "light". Argument type: STRING
tang	Optional. Specifies the taper angle. The default is 0. Argument type: REAL
icmat	Optional. Specifies the interconnect material. The default is "Metal". Argument type: DATEXMAT
envmat	Optional. Specifies the material used for the fill. The default is "Gas". Argument type: DATEXMAT
icr	Optional. Specifies the name of the interconnect region. The default is "ic-region". Argument type: STRING
envr	Optional. Specifies the region name of the filled part. The default is "env-region". Argument type: STRING

Appendix A: Commands

sdepe:icon_layer

Argument	Description
bc	<p>Optional. Specifies the base coordinate of the created layer in the z-direction for process-up-direction "+z" and in the x-direction for process-up-direction "-x".</p> <p>The default is the previous maximal z-coordinate for process-up-direction "+z" and the minimal x-coordinate for process-up-direction "-x".</p> <p>Argument type: REAL</p>
tpos	<p>Optional. Specifies the taper position. Options are "top" "bottom". The default is "bottom".</p> <p>If "taper-position" "bottom" is used, then the mask is tapered from the bottom position of the interconnect layer.</p> <p>If "taper-position" "top" is used, then the initial mask is extruded first and, then the sidewalls are tapered from the top position of the interconnect layer.</p> <p>Argument type: STRING</p>

Examples

[Example: Generating an Interconnect Structure on page 212](#)

Appendix A: Commands

sdepe:implant

sdepe:implant

This Scheme extension creates a doping distribution using analytic functions to emulate an implantation operation that can include diffusion.

Syntax

```
(sdepe:implant doping-name
  ["flat"] ["abs-height" abs-height | "delta-height" delta-height]
  ["pmat" pmatlist] ["direction" direction])
```

Returns

None

Arguments

Argument	Description
doping-name	Specifies the name of the doping profile. Argument type: STRING
"flat"	Optional. By specifying "flat", the contour of the exposed surface is projected onto the xy plane, and this planar surface is used as the reference for the placement of the doping profile doping-name. Argument type: STRING
abs-height	Optional. Specifies the absolute height. Argument type: REAL
delta-height	Optional. Specifies the delta height. Argument type: REAL
pmatlist	Optional. Specifies either a DATEX material or a list of DATEX materials that will behave like resist. The specified materials also protect the top exposed surface from doping implantation. Argument type: DATEXMAT DATEXMAT LIST
direction	Optional. Determines the direction of the profile with respect to the baseline computed by sdepe:implant. Options are "Positive" "Negative" "Both". The default is "Both". Argument type: STRING

Appendix A: Commands

sdepe:implant

Description

This Scheme extension applies the doping profile `doping-name` to the top surface of a device. Areas of 'resist' and device features covered by resist are protected from doping implantation.

The doping profile must be previously defined using one of the `sdedr` doping definition commands:

- `sdedr:define-1d-external-profile`
- `sdedr:define-analytical-profile`
- `sdedr:define-constant-profile`
- `sdedr:define-erf-profile`
- `sdedr:define-gaussian-profile`

Sentaurus Structure Editor analyzes the geometry of the surface of the structure to find resist layers blocking the interface with the gas. By default, the doping profile `doping-name` is placed at all surface faces exposed to the gas.

By specifying the "`flat`" option, the contour of the exposed surface is projected onto the `xy` plane, and this planar surface is used as the reference for the placement of the doping profile `doping-name`. By default, this plane is positioned along the `z`-axis at the uppermost nonresist device feature. This `z`-coordinate can be adjusted by using the option `delta-height` (in this case, the maximum `z`-coordinate is offset by `delta-height`) or can be set to an absolute value (fixed value) using the option `abs-height`.

Note:

By default, the `sdepe:implant` command places the doping profile at all surfaces exposed to gas, including the vertical sidewalls. If implantation should be performed only on the inclined (nonvertical) faces, then you must add the "`flat`" option to the `sdepe:implant` argument list.

A refinement window defined by using the `sdepe:implant` command, which is used for the doping, extends to the domain boundary by default. You can change this behavior by setting the global Scheme variable `extend-implant-window` to `#t`. Its default value is `#f`, but you can use the `(set! extend-implant-window #t)` Scheme command to change it. If you set `extend-implant-window #t`, then the refinement window extends from the domain boundary by $3 \times Length$, where the `Length` value is taken from the doping profile definition. If the `Factor` parameter was used in the doping definition, then the extension distance will be $3 \times Factor$.

Appendix A: Commands

sdepe:pattern

sdepe:pattern

This Scheme extension generates a patterned layer. The "mask" keyword identifies the mask to be used for the pattern operation. The mask must be already defined.

Since the pattern operation is analogous to a deposition step, for detailed explanations of the used algorithms and types, see [sdepe:depo on page 752](#).

Syntax

```
(sdepe:pattern "mask" mask-name "polarity" polarity
  ["type" depo-type] ["algorithm" depo-alg]
  "material" material "thickness" thickness
  ["steps" nsteps] ["region" region] [PT keywords and values])
```

Returns

ENTITY of the generated pattern layer

Arguments

Argument	Description
mask-name	Specifies the name of an existing mask to be used for the pattern operation. Argument type: STRING
polarity	Specifies the mask polarity. Options are "light" "dark". Argument type: STRING
depo-type	Optional. Specifies the deposition type. Options are "aniso" "iso". Argument type: STRING
depo-alg	Optional. Specifies the deposition algorithm to use. Options are "lopx" "lop-move" "sweep" "PT". Argument type: STRING
material	Specifies the name of a material. Argument type: DATEXMAT
thickness	Specifies the thickness. Argument type: REAL
nsteps	Optional. Specifies the number of steps, which is greater than or equal to 1. Argument type: INTEGER
region	Optional. Specifies the region name of the patterned body. Argument type: STRING
PT keywords and values	Optional. See Table 26 on page 188 .

Appendix A: Commands

sdepe:photo

sdepe:photo

This Scheme extension generates a photo (flat pattern) layer. The "mask" keyword identifies the mask to be used for the operation. The mask must be already defined.

The photo operation is similar to a combined sequence of a fill, a pattern, and an etch step. The used algorithms for the sdepe:photo operation are selected automatically to provide more robustness. Users cannot change these algorithms.

Note:

The difference between sdepe:photo and sdepe:pattern is that the sdepe:photo operation creates a flat top.

Syntax

```
(sdepe:photo "mask" mask-name "polarity" polarity ["type" depo-type]
  "material" material { "thickness" thickness | "height" height }
  ["steps" nsteps] ["region" region])
```

Returns

ENTITY of the generated pattern layer

Arguments

Argument	Description
mask-name	Specifies the name of an existing mask to be used for the operation. Argument type: STRING
polarity	Specifies the mask polarity. Options are "light" "dark". Argument type: STRING
depo-type	Optional. Specifies the deposition type. Options are "aniso" "iso". Argument type: STRING
material	Specifies the name of a material. Argument type: DATEXMAT
thickness	Specifies the thickness. Argument type: REAL
height	Specifies the height. Argument type: REAL
nsteps	Optional. Specifies the number of steps, which is greater than or equal to 1. Argument type: INTEGER
region	Optional. Specifies the region name of the patterned body. Argument type: STRING

Appendix A: Commands

sdepe:polish-device

sdepe:polish-device

This Scheme extension removes the top part of the device.

Syntax

```
(sdepe:polish-device { "thickness" thickness | "height" height }
                      [ "material" material])
```

Returns

None

Arguments

Argument	Description
thickness	If you specify "thickness", then the distance is measured from the top device position, and all materials from the top are removed. Argument type: REAL
height	If you specify "height", then all material above the specified height is removed. Argument type: REAL
material	Optional. If you specify "material", then only the top exposed bodies with the specified material attribute are polished. (This behavior is similar to a selective polish operation.) Argument type: DATEXMAT

Appendix A: Commands

sdepe:remove

sdepe:remove

This Scheme extension removes either a specified material or specified region.

Note:

This Scheme extension removes only exposed materials or regions.

Syntax

```
(sdepe:remove "material" material-name | "region" region-name)
```

Returns

None

Arguments

Argument	Description
material-name	Specifies a material to remove. Argument type: DATEXMAT
region-name	Specifies a region to remove. Argument type: STRING

Appendix A: Commands

sdepe:trim-masks

sdepe:trim-masks

This Scheme extension trims masks so that they do not extend outside the domain boundary. When `sdepe:define-pe-domain` is used to define the simulation domain, the global variable `domainboundary` is initialized. It is set to the wire body that represents the domain boundary. After `domainboundary` is set, (`sdepe:trim-masks domainboundary`) is used, which trims all masks to the specified domain.

Note:

If the domain boundary is not set explicitly and the `sdepe` commands are used on an existing body, then the domain boundary is extracted automatically and the `domainboundary` variable is set as well.

Syntax

`(sdepe:trim-masks domainboundary)`

Returns

None

Arguments

Argument	Description
<code>domainboundary</code>	Specifies a domain boundary. Argument type: WIRE BODY

Appendix A: Commands

sdesnmesh:axisaligned

sdesnmesh:axisaligned

This Scheme extension sets the global axis-aligned parameters used in the `AxisAligned` section of the mesh command file (see [AxisAligned Section on page 277](#)).

Note:

In earlier versions, you could specify `xCuts`, `yCuts`, and `zCuts` using a `REAL` or `REAL LIST` value. For example:

```
(sdesnmesh:axisaligned "xCuts" 1 "yCuts" (list 0.1 0.2 0.3))
```

This syntax has been extended to support new features in Sentaurus Mesh. For backward compatibility, the earlier syntax (allowing a `REAL` or `REAL LIST` value for `xCuts`, `yCuts`, and `zCuts`) is still supported. However, starting from Version T-2022.03, `xCuts`, `yCuts`, and `zCuts` can accept a `STRING` value as well. With a string syntax, these parameters can be specified as:

```
(sdesnmesh:axisaligned "xCuts" "1" "yCuts" "0.1 0.2 0.3")
```

Both example commands generate the same `AxisAligned` section in the Sentaurus Mesh command file. Using a `STRING` value for `xCuts`, `yCuts`, and `zCuts` also allows you to specify a series of cutlines using the more compact notation that Sentaurus Mesh supports. For example, the following command:

```
(sdesnmesh:axisaligned "xCuts" "1" "yCuts" "0.1 0.2 0.3"  
"zCuts" "(0 0.001) (1 0.002) 2")
```

causes Sentaurus Structure Editor to generate the following section in the Sentaurus Mesh command file:

```
AxisAligned {  
    xCuts=(1)  
    yCuts=(0.1 0.2 0.3)  
    zCuts=((0 0.001) (1 0.002) 2)  
}
```

This syntax for `zCuts` generates a series of mesh lines as described in the *Sentaurus™ Mesh User Guide*, `AxisAligned` Section (see the description of the `xCuts`, `yCuts`, and `zCuts` parameters).

Syntax

```
(sdesnmesh:axisaligned  
    ["binaryTreeSplitBox" binaryTreeSplitBox]  
    ["binaryTreeSplitFactorX" binaryTreeSplitFactorX]  
    ["binaryTreeSplitFactorY" binaryTreeSplitFactorY]  
    ["binaryTreeSplitFactorZ" binaryTreeSplitFactorZ]  
    ["convexTriangulation" convexTriangulation]  
    ["fitInterfaces" fitInterfaces] ["hintBoxSize" hintBoxSize]  
    ["imprintAccuracy" imprintAccuracy]  
    ["imprintCoplanarFacesOnly" imprintCoplanarFacesOnly]
```

Appendix A: Commands

sdesnmesh:axisaligned

```
[ "imprintCoplanarityAngle" imprintCoplanarityAngle]
[ "imprintCoplanarityDistance" imprintCoplanarityDistance]
[ "latticeCellSize" latticeCellSize]
[ "latticeDimensions" latticeDimensions]
[ "maxAngle" maxAngle] [ "maxAspectRatio" maxAspectRatio]
[ "maxBoundaryCutRatio" maxBoundaryCutRatio]
[ "maxNeighborRatio" maxNeighborRatio]
[ "overscan" overscan] [ "overscanResolution" overscanResolution]
[ "skipSameMaterialInterfaces" skipSameMaterialInterfaces]
[ "smoothing" smoothing] [ "spacingMethod" spacingMethod]
[ "splitDisconnectedRegions" splitDisconnectedRegions]
[ "virtualSpacing" virtualSpacing]
[ "xCuts" xCuts] [ "yCuts" yCuts] [ "zCuts" zCuts])
```

Returns

None

Arguments

Argument	Description
binaryTreeSplitBox	Optional. For 2D structures, this argument requires four numbers: (list REAL REAL REAL REAL). For 3D structures, this argument requires six numbers: (list REAL REAL REAL REAL REAL) Argument type: LIST
binaryTreeSplitFactorX	Optional. Argument type: REAL
binaryTreeSplitFactorY	Optional. Argument type: REAL
binaryTreeSplitFactorZ	Optional. Argument type: REAL
convexTriangulation	Optional. Argument type: BOOLEAN
fitInterfaces	Optional. Argument type: BOOLEAN
hintBoxSize	Optional. Argument type: REAL
imprintAccuracy	Optional. Argument type: REAL
imprintCoplanarFacesOnly	Optional. Argument type: BOOLEAN
imprintCoplanarityAngle	Optional. Argument type: REAL
imprintCoplanarityDistance	Optional. Argument type: REAL

Appendix A: Commands

sdesnmesh:axisaligned

Argument	Description
latticeCellSize	Optional. For 2D structures, this argument requires two numbers: (list REAL REAL). For 3D structures, this argument requires three numbers: (list REAL REAL REAL) Argument type: LIST
latticeDimensions	Optional. For 2D structures, this argument requires two numbers: (list REAL REAL). For 3D structures, this argument requires three numbers: (list REAL REAL REAL) Argument type: LIST
maxAngle	Optional. Argument type: REAL
maxAspectRatio	Optional. Argument type: REAL
maxBoundaryCutRatio	Optional. Argument type: REAL
maxNeighborRatio	Optional. Argument type: REAL
overscan	Optional. Argument type: BOOLEAN
overscanResolution	Optional. Argument type: REAL
skipSameMaterialInterfaces	Optional. Argument type: BOOLEAN
smoothing	Optional. Argument type: BOOLEAN
spacingMethod	Optional. Options are "even" "none" "regular" "smooth". Argument type: STRING
splitDisconnectedRegions	Optional. Argument type: BOOLEAN
virtualSpacing	Optional. Argument type: BOOLEAN
xCuts	Optional. Argument type: REAL LIST STRING
yCuts	Optional. Argument type: REAL LIST STRING
zCuts	Optional. Argument type: REAL LIST STRING

Appendix A: Commands

sdesnmesh:boundary

sdesnmesh:boundary

This Scheme extension sets the parameters of the Boundary section of the mesh command file (see [Boundary Section on page 281](#)).

Syntax

```
(sdesnmesh:boundary "Decimation"
    ["apply" applydecimation] ["accuracy" accuracydecimation]
    ["coplanarityAngle" coplanarityAngle]
    ["minDihedralAngle" minDihedralAngledecimation]
    ["shortEdge" shortEdge])

(sdesnmesh:boundary "DelPSC"
    ["apply" applyDelPSC] ["accuracy" accuracyDelPSC]
    ["ridgeAngle" ridgeAngle] ["ridgeSampling" ridgeSampling]
    ["skipFeatureSize" skipFeatureSize])

(sdesnmesh:boundary "DualContouring"
    ["apply" applydualContouring] ["decimation" decimation]
    ["minAngle" minAngle]
    ["minDihedralAngle" minDihedralAngledualContouring]
    ["resolution" resolution])

sdesnmesh:boundary "RegionMismatch"
    ["allow" allowregionMismatch] ["minVolume" minVolume])

(sdesnmesh:boundary "Save" "filename" filename)
```

Returns

None

Arguments

Argument	Description
accuracydecimation	Optional. Argument type: REAL
accuracyDelPSC	Optional. Argument type: REAL
allowregionMismatch	Optional. Argument type: BOOLEAN
applydecimation	Optional. Argument type: BOOLEAN
applyDelPSC	Optional. Argument type: BOOLEAN
applydualContouring	Optional. Argument type: BOOLEAN

Appendix A: Commands

sdesnmesh:boundary-clear

Argument	Description
coplanarityAngle	Optional. Argument type: REAL
decimation	Optional. Argument type: BOOLEAN
filename	Specifies the name of a file. Argument type: STRING
minAngle	Optional. Argument type: REAL
minDihedralAngledecimation	Optional. Argument type: REAL
minDihedralAngledualContouring	Optional. Argument type: REAL
minVolume	Optional. Argument type: REAL
resolution	Optional. Argument type: REAL
ridgeAngle	Optional. Argument type: REAL
ridgeSampling	Optional. Argument type: REAL
shortEdge	Optional. Argument type: REAL
skipFeatureSize	Optional. Argument type: REAL

sdesnmesh:boundary-clear

This Scheme extension deletes all boundary definition-related data that was previously defined by the sdesnmesh:boundary Scheme extension.

This means that after (sdesnmesh:boundary-clear) is called the Boundary section will not be added to the Sentaurus Mesh command file.

Syntax

(sdesnmesh:boundary-clear)

Returns

BOOLEAN

Appendix A: Commands

sdesnmesh:delaunizer

sdesnmesh:delaunizer

This Scheme extension sets the global parameters of the `Delaunizer` section of the mesh command file (see [Delaunizer Section on page 279](#)). These are the parameters for the delaunizer module.

Syntax

```
(sdesnmesh:delaunizer
  "coplanarityAngle" coplanarityAngle
  "coplanarityDistance" coplanarityDistance
  "delaunayTolerance" delaunayTolerance
  "edgeProximity" edgeProximity "faceProximity" faceProximity
  "maxAngle" maxAngle "maxConnectivity" maxConnectivity
  "maxNeighborRatio" maxNeighborRatio "maxPoints" maxPoints
  "maxSolidAngle" maxSolidAngle "maxTetQuality" maxTetQuality
  "minAngle" minAngle
  "minDihedralAngleAllowed" minDihedralAngleAllowed
  "minEdgeLength" minEdgeLength
  "minEdgeLengthAllowed" minEdgeLengthAllowed
  "sliverAngle" sliverAngle "sliverDistance" sliverDistance
  "sliverRemovalAlgorithm" sliverRemovalAlgorithm
  "storeDelaunayWeight" storeDelaunayWeight
  "type" type)
```

Returns

None

Arguments

Argument	Description
coplanarityAngle	Argument type: REAL
coplanarityDistance	Argument type: REAL
delaunayTolerance	Argument type: REAL
edgeProximity	Argument type: REAL
faceProximity	Argument type: REAL
maxAngle	Argument type: REAL
maxConnectivity	Argument type: REAL
maxNeighborRatio	Argument type: REAL

Appendix A: Commands

sdesnmesh:delaunizer

Argument	Description
maxPoints	Argument type: REAL
maxSolidAngle	Argument type: REAL
maxTetQuality	Argument type: REAL
minAngle	Argument type: REAL
minDihedralAngleAllowed	Argument type: REAL
minEdgeLength	Argument type: REAL
minEdgeLengthAllowed	Argument type: REAL
sliverAngle	Argument type: REAL
sliverDistance	Argument type: REAL
sliverRemovalAlgorithm	Argument type: INTEGER
storeDelaunayWeight	Argument type: BOOLEAN
type	Options are "boxmethod" "conforming" "constrained". Argument type: STRING

sdesnmesh:delaunizer-tolerance

This Scheme extension sets the tolerance parameters for the delaunizer in the Sentaurus Mesh command file (see [Delaunizer Section on page 279](#)).

Syntax

```
(sdesnmesh:delaunizer-tolerance tolerance location location-type  
window)
```

Returns

None

Arguments

Argument	Description
tolerance	Specifies the tolerance. Argument type: REAL
location	Specifies the location. Options are "boundary" "surface". Argument type: STRING
location-type	Specifies the location type. Options are: "region" ["region1" ["region2"]] "material" ["material1" ["material2"]] "global"
window	Specifies the dimensions of a window by using the format "window" x0 y0 z0 x1 y1 z1.

Appendix A: Commands

sdesnmesh:interpolate

sdesnmesh:interpolate

This Scheme extension sets the parameters of the `Interpolate` section of the mesh command file (see [IOControls Section and Interpolate Section on page 276](#)).

Syntax

```
(sdesnmesh:interpolate
  [ "interpolateElements" interpolateElements]
  [ "interpolateInterfaces" interpolateInterfaces]
  [ "keepTotalConcentration" keepTotalConcentration]
  [ "lateralDiffusion" lateralDiffusion])
```

Returns

BOOLEAN

Arguments

Argument	Description
interpolateElements	Optional. The default is #f. Argument type: BOOLEAN
interpolateInterfaces	Optional. The default is #f. Argument type: BOOLEAN
keepTotalConcentration	Optional. The default is #f. Argument type: BOOLEAN
lateralDiffusion	Optional. The default is #f. Argument type: BOOLEAN

Appendix A: Commands

sdesnmesh:iocontrols

sdesnmesh:iocontrols

This Scheme extension sets the parameters of the `IOControls` section of the mesh command file (see [IOControls Section and Interpolate Section on page 276](#)).

Syntax

```
(sdesnmesh:iocontrols
  [ "inputFile" inputFile ] [ "outputFile" outputFile ]
  [ "EnableSections" EnableSections ] [ "EnableEMW" EnableEMW ]
  [ "EnableIsotropic" EnableIsotropic ]
  [ "EnableOffset" EnableOffset ] [ "EnableTensor" EnableTensor ]
  [ "EnableTools" EnableTools ]
  [ "numThreads" numThreads ]
  [ "saveInterfaceRegions" saveInterfaceRegions ]
  [ "unstructuredTensorMesh" unstructuredTensorMesh ]
  [ "useDFISEcoordinates" useDFISE | "useUCScoordinates" useUCS ]
  [ "verbosity" verbosity ] )
```

Returns

BOOLEAN

Arguments

Argument	Description
inputFile	Optional. Argument type: STRING
outputFile	Optional. Argument type: STRING
EnableSections	Optional. Argument type: BOOLEAN
EnableEMW	Optional. Argument type: BOOLEAN
EnableIsotropic	Optional. Argument type: BOOLEAN
EnableOffset	Optional. Argument type: BOOLEAN
EnableTensor	Optional. Argument type: BOOLEAN
EnableTools	Optional. Argument type: BOOLEAN
numThreads	Optional. Argument type: INTEGER
saveInterfaceRegions	Optional. The default is #t. Argument type: BOOLEAN
unstructuredTensorMesh	Optional. The default is #f. Argument type: BOOLEAN

Appendix A: Commands

sdesnmesh:offsetting

Argument	Description
useDFISE	Optional. The default is #f. Argument type: BOOLEAN
useUCS	Optional. The default is #t. Argument type: BOOLEAN
verbosity	Optional. Argument type: INTEGER

sdesnmesh:offsetting

This Scheme extension sets the global parameters for offsetting (see [Offsetting Refinements on page 252](#)). These parameters are added to the `Offsetting` section of the mesh command file.

Syntax

```
(sdesnmesh:offsetting ["includeExterior" includeExterior]
                      ["normalSmoothing" normalSmoothing])
```

Returns

BOOLEAN

Arguments

Argument	Description
includeExterior	Optional. Argument type: BOOLEAN
normalSmoothing	Optional. Argument type: BOOLEAN

Examples

```
(sdesnmesh:offsetting "includeExterior" #t "normalSmoothing" #t)
```

Appendix A: Commands

sdesnmesh:qualityreport

sdesnmesh:qualityreport

This Scheme extension sets the parameters of the QualityReport section of the Sentaurus Mesh command file (see [QualityReport Section on page 284](#)).

See *Sentaurus™ Mesh User Guide*, QualityReport Section, for detailed descriptions of arguments.

Syntax

```
(sdesnmesh:qualityreport qrtype qrelements
  [ "limitMaxConnectivity" limitMaxConnectivity ]
  [ "limitMaxNonDelaunay" limitMaxNonDelaunay ]
  [ "limitMinAngle" limitMinAngle ]
  [ "limitMinEdgeLength" limitMinEdgeLength ]
  [ "limitMinVolume" limitMinVolume ])
```

Returns

BOOLEAN

Arguments

Argument	Description
qrtype	Specifies the type of quality report. Options are "Global" "Material" "Region". The QualityReport section can contain one global block, and multiple material and region blocks defined by qrtype as follows: <ul style="list-style-type: none">"Global" defines a global block. In this case, the qrelements argument is not used."Material" defines a material block. In this case, qrelements is a list of DATEX materials."Region" defines a region block. In this case, qrelements is a list of region names. Argument type: STRING
qrelements	Argument type: STRING LIST
limitMaxConnectivity	Optional. Argument type: INTEGER
limitMaxNonDelaunay	Optional. Argument type: REAL
limitMinAngle	Optional. Argument type: REAL
limitMinEdgeLength	Optional. Argument type: REAL
limitMinVolume	Optional. Argument type: REAL

Appendix A: Commands

sdesnmesh:qualityreport

Examples

```
(sde:clear)
(sdesnmesh:qualityreport "Global" "limitMaxConnectivity" 100)

(sdesnmesh:qualityreport "Material" (list "Silicon" "PolySilicon")
 "limitMaxNonDelaunay" 5 "limitMinEdgeLength" 0.01)

(sdesnmesh:qualityreport "Region" (list "region1")
 "limitMaxNonDelaunay" 10 "limitMinAngle" 60
 "limitMinEdgeLength" 0.001 "limitMinVolume" 0.0001)
```

These Scheme extensions generate the following QualityReport section in the Sentaurus Mesh command file:

```
QualityReport {
    Global {
        limitMaxConnectivity = 100
    }
    Material = { "Silicon" "PolySilicon" } {
        limitMaxNonDelaunay = 5
        limitMinEdgeLength = 0.01
    }
    Region = { "region1" } {
        limitMaxNonDelaunay = 10
        limitMinAngle = 60
        limitMinEdgeLength = 0.001
        limitMinVolume = 0.0001
    }
}
```

Appendix A: Commands

sdesnmesh:tensor

sdesnmesh:tensor

This Scheme extension replaces the `Tensor` section of the mesh command file with the given string. It allows direct specification of the `Tensor` section without syntax-checking (see [Tensor Section on page 281](#)).

Note:

Double quotation marks within the string must be escaped with the backslash. For example:

```
\\"example string\\"
```

Syntax

```
(sdesnmesh:tensor value)
```

Returns

BOOLEAN

Arguments

Argument	Description
value	Specifies content that will replace the <code>Tensor</code> section of the mesh command file. Argument type: STRING

Appendix A: Commands

sdesnmesh:tools

sdesnmesh:tools

This Scheme extension replaces the `Tools` section of the mesh command file with the given string. It allows direct manipulation of the `Tools` section without syntax-checking (see [Tools Section on page 284](#)).

Note:

Double quotation marks within the string must be escaped with the backslash. For example:

```
\\"example string\\"
```

Syntax

```
(sdesnmesh:tools value)
```

Returns

BOOLEAN

Arguments

Argument	Description
value	Specifies content that will replace the <code>Tools</code> section of the mesh command file. Argument type: STRING

Appendix A: Commands

sdesp:begin

sdesp:begin

This Scheme extension begins the Sentaurus Process and Sentaurus Structure Editor boundary-merging mode.

This Scheme extension is called before beginning the merging of .sat file boundaries. It must be called before the sdesp:define-step command can be used. For that reason, it is necessary to begin the Sentaurus Process and Sentaurus Structure Editor mode, which is terminated with sdesp:finalize.

Note:

This Scheme extension applies to 3D models only.

Syntax

```
(sdesp:begin)
```

Returns

None

Examples

```
; Create Sentaurus Process Tcl file using sdesp commands  
; Start Sentaurus Process/Sentaurus Structure Editor mode  
(sdesp:begin)  
  
; Load .sat files, each time defining a step name to be reproduced  
; in Sentaurus Process  
(sdesp:define-step "step1" "_step1.sat")  
(sdesp:define-step "step2" "_step2.sat")  
(sdesp:define-step "step3" "_step3.sat")  
(sdesp:define-step "step4" "_step4.sat")  
  
; Conclude Sentaurus Process/Sentaurus Structure Editor mode,  
; write Tcl file  
(sdesp:finalize "sprocess_sde.tcl")
```

Appendix A: Commands

sdesp:define-step

sdesp:define-step

This Scheme extension merges a boundary (in a .sat file) and defines a structure step to be reproduced in Sentaurus Process. It loads a .sat file and merges with the currently loaded geometry and associates `step-name` with the current step. See [sdesp:begin on page 786](#).

Syntax

```
(sdesp:define-step step-name file-name)
```

Returns

None

Arguments

Argument	Description
step-name	Specifies the name of a step. Argument type: STRING
file-name	Specifies the .sat file to load. Argument type: STRING

sdesp:finalize

This Scheme extension completes the merging of boundaries in the Sentaurus Process and Sentaurus Structure Editor mode. It writes a Tcl file defining the `recreate_step` Tcl procedure called from within Sentaurus Process to recreate model geometry by redefining region materials. See [sdesp:begin on page 786](#).

Syntax

```
(sdesp:finalize Tcl-file-name)
```

Returns

None

Arguments

Argument	Description
Tcl-file-name	Specifies the name of a Tcl file. Argument type: STRING

Appendix A: Commands

sdesp:restore-state

sdesp:restore-state

This Scheme extension restores the internal state of Sentaurus Process and Sentaurus Structure Editor to permit the restarting of the boundary-merging process.

During boundary merging by Sentaurus Process and Sentaurus Structure Editor, you might want to save the state and restore it later (perhaps in a different Sentaurus Structure Editor session) before continuing with the merging leading to its finalization.

To restore: Most of the internal bookkeeping is stored in the .sat file that is reloaded. However, the sequence of steps that have been defined so far must be restored using this Scheme extension. The list of all steps in the correct order is given as an argument list to this extension.

See [sdesp:begin on page 786](#).

Syntax

```
(sdesp:restore-state step-name-list)
```

Returns

None

Arguments

Argument	Description
step-name-list	Specifies a list of steps in the correct order. Argument type: STRING LIST

Appendix A: Commands

sdestr:capitalize

sdestr:capitalize

This Scheme extension changes the first character of the input string to uppercase if it is a lowercase alphabetic character. It returns either the capitalized string or the unchanged string.

Note:

The input parameter and the returned value of the sdestr:capitalize Scheme extension are the same as for the Python `capitalize` method (see <https://www.javatpoint.com/python-string-capitalize-method>).

Syntax

```
(sdestr:capitalize input-string)
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:capitalize "try this.")
> "Try this."
(sdestr:capitalize "javatpoint")
> "Javatpoint"
(sdestr:capitalize "Javatpoint")
> "Javatpoint"
(sdestr:capitalize "#javatpoint")
> "#javatpoint"
(sdestr:capitalize "1-javatpoint")
> "1-javatpoint"
```

Appendix A: Commands

sdestr:casetfold

sdestr:casetfold

This Scheme extension returns a lowercase copy of the string.

Note:

The input parameter and the returned value of the sdestr:casetfold Scheme extension are the same as for the Python `casifold` method (see <https://www.javatpoint.com/python-string-casifold-method>).

Syntax

```
(sdestr:casetfold input-string)
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:casetfold "JAVATPOINT")
> "javatpoint"

(sdestr:casetfold "JAVATPOINT - $"
> "javatpoint ? ss"

(sdestr:casetfold "JaVaTpOiNt"
> "javatpoint"
```

Appendix A: Commands

sdestr:center

sdestr:center

This Scheme extension takes an input string and a string length.

Note:

The input parameter and the returned value of the sdestr:center Scheme extension are the same as for the Python `center` method (see <https://www.javatpoint.com/python-string-center-method>).

Syntax

```
(sdestr:center input-string str-length [fill-char])
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
str-length	Specifies the string length. Argument type: INTEGER
fill-char	Optional. Specifies a character with which to fill the left and right padding of the string. If <code>fill-char</code> is not specified, then the default character is the hash character (#). Argument type: STRING

Examples

```
(sdestr:center "Hello World!" 30)
> "#####Hello World!#####"

(sdestr:center "Hello World!" 30 #\;)
> ";Hello World!;"
```

Appendix A: Commands

sdestr:count

sdestr:count

This Scheme extension returns the number of occurrences of a specified substring in the input string, in the specified range.

Note:

The input parameter and the returned value of the sdestr:count Scheme extension are the same as for the Python `count` method (see <https://www.javatpoint.com/python-string-count-method>).

Syntax

```
(sdestr:count input-string substring [start-index end-index])
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
substring	Specifies the substring. Argument type: STRING
start-index	Optional. Specifies the start index of the range. Argument type: INTEGER
end-index	Optional. Specifies the end index of the range. Argument type: INTEGER

Examples

```
(sdestr:count "Hello Javatpoint" "t")
> 2

(sdestr:count "ab bc ca de ed ad da ab bc ca" "a")
> 6

(sdestr:count "ab bc ca de ed ad da ab bc ca" "a" 3)
> 5

(sdestr:count "ab bc ca de ed ad da ab bc ca" "a" 3 8)
> 1

(sdestr:count "ab bc ca de ed ad da ab bc ca 12 23 35 62" "2")
> 3
```

Appendix A: Commands

sdestr:endswith

sdestr:endswith

This Scheme extension returns true if the input string terminates with the specified suffix between the optional start index and end index. Otherwise, it returns false.

Note:

The input parameter and the returned value of the sdestr:endswith Scheme extension are the same as for the Python `endswith` method (see <https://www.javatpoint.com/python-string-endswith-method>).

Syntax

```
(sdestr:endswith input-string suffix  
[start-index = 0 end-index]) = (length-input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
suffix	Specifies the suffix. Argument type: STRING
start-index	Optional. Specifies the start index. If it is not specified, then <code>start-index=0</code> . Argument type: INTEGER
end-index	Optional. Specifies the end index. If it is not specified, then <code>end-index</code> is the length of the input string. Argument type: INTEGER

Examples

```
(sdestr:endswith "Hello this is javatpoint." ".")  
> #t

(sdestr:endswith "Hello this is javatpoint." "is")  
> #f

(sdestr:endswith "Hello this is javatpoint." "is" 10)  
> #f

(sdestr:endswith "Hello this is javatpoint." "is" 0 13)  
> #t
```

Appendix A: Commands

sdestr:expandtabs

sdestr:expandtabs

This Scheme extension replaces all tab characters (\t) in a string with the number of specified spaces.

Note:

The input parameter and the returned value of the sdestr:expandtabs Scheme extension are the same as for the Python expandtabs method (see <https://www.javatpoint.com/python-string-expandtabs-method>).

Syntax

```
(sdestr:expandtabs input-string [tabsize])
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
tabsize	Optional. Specifies the number of spaces that replaces a tab (\t) character (default is 8). This value can be overridden as required. Argument type: INTEGER

Examples

```
(sdestr:expandtabs "Welcome \t to \t the \t Javatpoint.")  
> "Welcome           to           the           Javatpoint."  
  
(sdestr:expandtabs "Welcome \t to \t the \t Javatpoint." 1)  
> "Welcome   to   the   Javatpoint."  
  
(sdestr:expandtabs "Welcome \t to \t the \t Javatpoint." 2)  
> "Welcome     to     the     Javatpoint."  
  
(sdestr:expandtabs "Welcome \t to \t the \t Javatpoint." 3)  
> "Welcome       to       the       Javatpoint."
```

Appendix A: Commands

sdestr:find

sdestr:find

This Scheme extension returns the index value of the string, where the substring is found between the start index and the end index.

Note:

The input parameter and the returned value of the sdestr:find Scheme extension are the same as for the Python `find` method (see <https://www.javatpoint.com/python-string-find-method>).

Syntax

```
(sdestr:find input-string substring start-index end-index)
```

Returns

If found, it returns the index of the substring. Otherwise, it returns -1.

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
substring	Specifies the substring. Argument type: STRING
start-index	Specifies the start index. Argument type: INTEGER
end-index	Specifies the end index. Argument type: INTEGER

Examples

```
(sdestr:find "Welcome to the Javatpoint." "the")
> 11

(sdestr:find "Welcome to the Javatpoint." "is")
> -1

(sdestr:find "Welcome to the Javatpoint." "t")
> 8

(sdestr:find "Welcome to the Javatpoint." "t" 25)
> -1

(sdestr:find "Welcome to the Javatpoint." "t" 20 25)
> 24
```

Appendix A: Commands

sdestr:format

sdestr:format

This Scheme extension performs format operations on the input string.

Braces {} are used in the input string, to be replaced with the specified value or values.
Braces can contain either an index argument or a positional argument.

The Scheme extension returns the formatted input string, using the passed value or values.

Note:

The input parameter and the returned value of the sdestr:format Scheme extension are the same as for the Python `format` method (see <https://www.javatpoint.com/python-string-format-method>).

Syntax

```
(sdestr:format input-string values)
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
values	Specifies values. Argument type: STRING or STRING STRING ...

Examples

The following example formats the input string using positional delimiters:

```
(define str "Java")
(define str2 "C#")
(sdestr:format "{} and {} both are programming languages" str str2)
> "Java and C# both are programming languages"
```

In the following example, the delimiters (braces) use numeric indices to replace and format the input string:

```
(define str "Java")
(define str2 "C#")
(sdestr:format "{1} and {0} both are programming languages" str str2)
> "C# and Java both are programming languages"
```

Appendix A: Commands

sdestr:format

The following examples format the input string in different number systems:

```
(define val 10)
(sdestr:format "decimal: {0:d}" val)    ;; # display decimal result
> "decimal: 10"

(sdestr:format "hex: {0:x}" val)         ;; # display hexadecimal result
> "hex: a"

(sdestr:format "octal: {0:o}" val)        ;; # display octal result
> "octal: 12"

(sdestr:format "binary: {0:b}" val)        ;; # display binary result
> "binary: 1010"
```

The following examples format the value as a floating-point value or as a percentile:

```
(define val 100000000)
(sdestr:format "decimal: {:,}" val)      ;; # formatting float value
> "decimal: 100,000,000"

(sdestr:format "decimal: {:.2%}" (/ 56 9)) ;; # formatting percentile value
> "decimal: 622.22%"
```

Appendix A: Commands

sdestr:index

sdestr:index

This Scheme extension returns the index value of the input string, where the substring is found between the start index and the end index.

It throws an exception if the string is not found. It works in the same way as sdestr:find.

Note:

The input parameter and the returned value of the sdestr:index Scheme extension are the same as for the Python `index` method (see <https://www.javatpoint.com/python-string-index-method>).

Syntax

```
(sdestr:index input-string substring start-index end-index)
```

Returns

INTEGER

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
substring	Specifies the substring. Argument type: STRING
start-index	Specifies the start index. Argument type: INTEGER
end-index	Specifies the end index. Argument type: INTEGER

Examples

```
(sdestr:index "Welcome to the Javatpoint." "at")
> 18
```

```
(sdestr:index "Welcome to the Javatpoint." "ate")
> *** Error sde:error: ValueError: substring not found
```

```
(sdestr:index "Welcome to the Javatpoint." "p" 19 21)
> 20
```

Appendix A: Commands

sdestr:isalnum

sdestr:isalnum

This Scheme extension returns true if the characters in the string are alphanumeric, that is, alphabetic characters or numerals, and there is at least one character. Otherwise, this Scheme extension returns false.

Note:

The input parameter and the returned value of the sdestr:isalnum Scheme extension are the same as for the Python `isalnum` method (see <https://www.javatpoint.com/python-string-isalnum-method>).

Syntax

```
(sdestr:isalnum input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:isalnum "Welcome")
> #t

(sdestr:isalnum "Welcome ")
> #f

(sdestr:isalnum "Welcome123")
> #t

(sdestr:isalnum "Welcome 123")
> #f

(sdestr:isalnum "123456")
> #t
```

Appendix A: Commands

sdestr:isalpha

sdestr:isalpha

This Scheme extension returns true if the input string contains only alphanumeric characters, and there is at least one character. Otherwise, it returns false.

Note:

The input parameter and the returned value of the sdestr:isalpha Scheme extension are the same as for the Python `isalpha` method (see <https://www.javatpoint.com/python-string-isalpha-method>).

Syntax

```
(sdestr:isalnum input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:isalpha "Javatpoint")
> #t

(sdestr:isalpha "Welcome to the Javatpoint")
> #f

(sdestr:isalpha "Javatpoint12345")
> #f
```

Appendix A: Commands

sdestr:isdecimal

sdestr:isdecimal

This Scheme extension returns true if the input string contains only decimal digits, and there is at least one digit. Otherwise, it returns false.

Note:

The input parameter and the returned value of the sdestr:isdecimal Scheme extension are the same as for the Python `isdecimal` method (see <https://www.javatpoint.com/python-string-isdecimal-method>).

Syntax

```
(sdestr:isdecimal input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:isdecimal "Javatpoint")
> #f

(sdestr:isdecimal "123")
> #t

(sdestr:isdecimal "2.50")
> #t

(sdestr:isdecimal "@#$")
> #f
```

Appendix A: Commands

sdestr:isdigit

sdestr:isdigit

This Scheme extension returns true if the input string contains only decimal digits, and there is at least one digit. Otherwise, it returns false.

Note:

The input parameter and the returned value of the sdestr:isdigit Scheme extension are the same as for the Python `isdigit` method (see <https://www.javatpoint.com/python-string-isdigit-method>).

Syntax

```
(sdestr:isdigit input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:isdigit "12345")
> #t

(sdestr:isdigit "120-2569-854")
> #f

(sdestr:isdigit "123!@#$")
> #f
```

Appendix A: Commands

sdestr:islower

sdestr:islower

This Scheme extension returns true if the input string contains only lowercase alphabetic characters, among other non-alphabetic characters. Otherwise, it returns false.

Note:

The input parameter and the returned value of the sdestr:islower Scheme extension are the same as for the Python `islower` method (see <https://www.javatpoint.com/python-string-islower-method>).

Syntax

```
(sdestr:islower input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:islower "javatpoint")
> #t

(sdestr:islower "Welcome To JavaTpoint")
> #f

(sdestr:islower "hi, my contact is 9856*****")
> #t
```

Appendix A: Commands

sdestr:isnumeric

sdestr:isnumeric

This Scheme extension returns true if the input string contains only numeric characters. Otherwise, it returns false. Numeric characters include digits (0, 1, 2, ..., 9) and all characters that have the Unicode numeric value property (+, -, e).

Note:

The input parameter and the returned value of the `sdestr:isnumeric` Scheme extension are the same as for the Python `isnumeric` method (see <https://www.javatpoint.com/python-string-isnumeric-method>).

Syntax

```
(sdestr:isnumeric input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:isnumeric "12345")
> #t

(sdestr:isnumeric "java12345")
> #f

(sdestr:isnumeric "123-4525-00")
> #f

(sdestr:isnumeric "-123.45e2")
> #t

(sdestr:isnumeric "-123.45E2")
> #f
```

Appendix A: Commands

sdestr:isspace

sdestr:isspace

This Scheme extension returns true if the input string contains only whitespace characters. Otherwise, it returns false.

Here, *whitespace characters* includes space, new lines, and tabs. Such characters are defined in the Unicode Character Database as **Other** or **Separator** and include:

- " ": Space
- \t: Horizontal tab
- \n: New line
- \v: Vertical tab
- \f: Feed
- \r: Return

Note:

The input parameter and the returned value of the sdestr:isspace Scheme extension are the same as for the Python `isspace` method (see <https://www.javatpoint.com/python-string-isspace-method>).

Syntax

```
(sdestr:isspace input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:isspace " ")
> #t
(sdestr:isspace "ab cd ef")
> #f
(sdestr:isspace "ab cd ef \n")
> #f
(sdestr:isspace "\t \r \n")
> #t
```

Appendix A: Commands

sdestr:istitle

sdestr:istitle

This Scheme extension returns true if the input string is titled correctly. Otherwise, it returns false.

A title is considered correct if an uppercase alphabetic character is the first character in the input string.

Note:

The input parameter and the returned value of the sdestr:istitle Scheme extension are the same as for the Python `istitle` method (see <https://www.javatpoint.com/python-string-istitle-method>).

Syntax

```
(sdestr:istitle input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:istitle "Welcome To Javatpoint")
> #t

(sdestr:istitle "hello javatpoint")
> #f

(sdestr:istitle "ab cd ef")
> #f

(sdestr:istitle "Ab Cd Ef")
> #t

(sdestr:istitle "1b 2d 3f")
> #f
```

Appendix A: Commands

sdestr:isupper

sdestr:isupper

This Scheme extension returns true if the input string has only uppercase alphabetic characters, if there are alphabetic characters. It returns false if any alphabetic characters are lowercase.

Note:

The input parameter and the returned value of the sdestr:isupper Scheme extension are the same as for the Python `isupper` method (see <https://www.javatpoint.com/python-string-isupper-method>).

Syntax

```
(sdestr:isupper input-string)
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:isupper "WELCOME TO JAVATPOINT. ")
> #t

(sdestr:isupper "WELCOME To JAVATPOINT. ")
> #f

(sdestr:isupper "Learn Java Here. ")
> #f

(sdestr:isupper "123 @#$ -JAVA. ")
> #t
```

Appendix A: Commands

sdestr:join

sdestr:join

This Scheme extension merges the string representation of the given string list with the specified pattern.

It returns a new string that is the concatenation of the strings in the list with the specified pattern string.

Note:

The input parameter and the returned value of the sdestr:join Scheme extension are the same as for the Python `join` method (see <https://www.javatpoint.com/python-string-join-method>).

Syntax

```
(sdestr:join input-string-list pattern)
```

Returns

STRING

Arguments

Argument	Description
input-string-list	Specifies a list of input strings. Argument type: STRING LIST
pattern	Specifies a pattern. Argument type: STRING

Examples

```
(sdestr:join (list "1" "2" "3" "4") ":" )
> "1:2:3:4"

(sdestr:join (list "J" "a" "v" "a" "t" "p" "o" "i" "n" "t") "")
> "Javatpoint"

(sdestr:join (list "Java" "C#" "Python") "->")
> "Java->C#->Python"

(sdestr:join (list "key1" "key2") "&")
> "key1&key2"
```

Appendix A: Commands

sdestr:length

sdestr:length

This Scheme extension returns the number of characters in the input string.

Note:

The input parameter and the returned value of the sdestr:length Scheme extension are the same as for the Python `length` method (see <https://www.javatpoint.com/java-string-length>).

Syntax

(sdestr:length input-string)

Returns

INTEGER

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:length "javatpoint")
> 10
```

Appendix A: Commands

sdestr:ljust

sdestr:ljust

This Scheme extension returns the input string aligned to the left and, depending on the specified string length, it fills any remaining space with the specified character.

Note:

The input parameter and the returned value of the sdestr:ljust Scheme extension are the same as for the Python `ljust` method (see <https://www.javatpoint.com/python-string-ljust-method>).

Syntax

```
(sdestr:ljust input-string str-length [fill-char])
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
str-length	Specifies the string length. Argument type: INTEGER
fill-char	Optional. Specifies a character with which to fill the left padding of the string. If <code>fill-char</code> is not specified, then the default fill character is " ". Argument type: STRING

Examples

```
(sdestr:ljust "Javatpoint" 20)
> "Javatpoint "

(sdestr:ljust "Javatpoint" 15 "$")
> "Javatpoint$$$$"

(sdestr:ljust "Javatpoint" 5)
> "Javatpoint"

(sdestr:ljust "Javatpoint" 5 "$")
> "Javatpoint"
```

Appendix A: Commands

sdestr:lower

sdestr:lower

This Scheme extension converts any uppercase alphabetic characters in the input string to lowercase. It returns the converted input string, depending on the specified string length, it fills any remaining space with the specified character.

Note:

The input parameter and the returned value of the `sdestr:lower` Scheme extension are the same as for the Python `lower` method (see <https://www.javatpoint.com/python-string-lower-method>).

Syntax

```
(sdestr:lower input-string)
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:lower "Javatpoint")
> "javatpoint"
```

```
(sdestr:lower "Welcome To JAVAtpoint, WWW.JAVATPOINT.COM")
> "welcome to javatpoint, www.javatpoint.com"
```

Appendix A: Commands

sdestr:rstrip

sdestr:rstrip

This Scheme extension removes all leading space or characters from the input string.

Note:

The input parameter and the returned value of the sdestr:rstrip Scheme extension are the same as for the Python `lstrip` method (see <https://www.javatpoint.com/python-string-lstrip-method>).

Syntax

```
(sdestr:rstrip input-string [remove-string])
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
remove-string	Optional. Specifies particular characters to remove if they are leading characters in the input string. If you do not specify <code>remove-string</code> , then the Scheme extension removes all the leading space from the string. Argument type: STRING

Examples

```
(sdestr:rstrip " Javatpoint ")
> "Javatpoint "

(sdestr:rstrip "$$$-$Javatpoint-$$$$" "$")
> "-Javatpoint-$$$$"

(sdestr:rstrip "http://www.javatpoint.com" "http://www.")
> "javatpoint.com"
```

Appendix A: Commands

sdestr:partition

sdestr:partition

This Scheme extension splits the input string at the first occurrence of the specified separator. It returns the part of the input string before the separator string, the separator itself, and the part after it. If the separator is not found, then this Scheme extension returns the separator and two empty strings.

Note:

The input parameter and the returned value of the sdestr:partition Scheme extension are the same as for the Python `partition` method (see <https://www.javatpoint.com/python-string-partition-method>).

Syntax

```
(sdestr:partition input-string separator)
```

Returns

List with three members

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
separator	Specifies a separator. Argument type: STRING

Examples

```
(sdestr:partition "Java is a programming language" "is")
> ("Java" "is" " a programming language")

(sdestr:partition "Java is a programming language" "Java")
> (" " "Java" " is a programming language")

(sdestr:partition "Java is a programming language" "language")
> ("Java is a programming" "language" "")

(sdestr:partition "Java is a programming language" "av")
> ("J" "av" "a is a programming language")

(sdestr:partition "Java is a programming language" "not")
> ("Java is a programming language" "" "")
```

Appendix A: Commands

sdestr:replace

sdestr:replace

This Scheme extension replaces a part of the input string with a new string. It returns the input string with the replaced string.

Note:

The input parameter and the returned value of the `sdestr:replace` Scheme extension are the same as for the Python `replace` method (see <https://www.javatpoint.com/python-string-replace-method>).

Syntax

```
(sdestr:replace input-string old-string new-string [ncount])
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
old-string	Specifies the old string. Argument type: STRING
new-string	Specifies the new string. Argument type: STRING
ncount	Optional. If you specify this argument, then the Scheme extension replaces one string with a new string for the specified number of occurrences of the old string. If <code>ncount</code> is not specified, then all occurrences of the old string are replaced. Argument type: INTEGER

Examples

```
(sdestr:replace "Java is a programming language" "Java" "C")  
> "C is a programming language"
```

```
(sdestr:replace "Java C C# Java Php Python Java" "Java" "Fennel")  
> "Fennel C C# Fennel Php Python Fennel"
```

```
(sdestr:replace "Java C C# Java Php Python Java" "Java" "Fennel" 1)  
> "Fennel C C# Java Php Python Java"
```

Appendix A: Commands

sdestr:rfind

sdestr:rfind

This Scheme extension is similar to sdestr:find, but it traverses the input string in a backward direction.

It returns the index value of the input string, where the substring is found between the start index and the end index.

Note:

The input parameter and the returned value of the sdestr:rfind Scheme extension are the same as for the Python rfind method (see <https://www.javatpoint.com/python-string-rfind-method>).

Syntax

```
(sdestr:rfind input-string substring start-index end-index)
```

Returns

If found, it returns the index of the substring; otherwise, -1.

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
substring	Specifies the substring. Argument type: STRING
start-index	Specifies the start index. Argument type: INTEGER
end-index	Specifies the end index. Argument type: INTEGER

Examples

```
(sdestr:rfind "Learn Java from Javatpoint" "Java")
> 16

(sdestr:rfind "It is technical tutorial" "t")
> 18

(sdestr:rfind "It is technical tutorial" "t" 5)
> 18

(sdestr:rfind "It is technical tutorial" "t" 5 10)
> 6
```

Appendix A: Commands

sdestr:rindex

sdestr:rindex

This Scheme extension is the same as `sdestr:index`, but it traverses the input string in a backward direction.

It returns the index value of the input string where the substring is found between the start index and the end index.

It throws an exception if the substring is not found. It works in the same way as `sdestr:find`.

Note:

The input parameter and the returned value of the `sdestr:rindex` Scheme extension are the same as for the Python `rindex` method (see <https://www.javatpoint.com/python-string-rindex-method>).

Syntax

```
(sdestr:rindex input-string substring start-index end-index)
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
substring	Specifies the substring. Argument type: STRING
start-index	Specifies the start index. Argument type: INTEGER
end-index	Specifies the end index. Argument type: INTEGER

Examples

```
(sdestr:rindex "It is technical tutorial" "t")
> 18
(sdestr:rindex "It is technical tutorial" "t" 5 15)
> 6
(sdestr:rindex "Hello C Language" "t")
> "ValueError: substring not found"
(sdestr:rindex "Hello C Language" "t" 5 15)
> "ValueError: substring not found"
```

Appendix A: Commands

sdestr:rjust

sdestr:rjust

This Scheme extension returns the input string aligned to the right and, depending on the specified string length, it fills any remaining space with the specified character.

Note:

The input parameter and the returned value of the sdestr:rjust Scheme extension are the same as for the Python `rjust` method (see <https://www.javatpoint.com/python-string-rjust-method>).

Syntax

```
(sdestr:rjust input-string str-length [fill-char])
```

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
str-length	Specifies the string length. Argument type: INTEGER
fill-char	Optional. Specifies a character with which to fill the right padding of the string. Argument type: STRING

Examples

```
(sdestr:rjust "Javatpoint" 20)
> " Javatpoint"

(sdestr:rjust "Javatpoint" 15 "$")
> "$$$$$Javatpoint"

(sdestr:rjust "Javatpoint" 5)
> "Javatpoint"

(sdestr:rjust "Javatpoint" 5 "$")
> "Javatpoint"
```

Appendix A: Commands

sdestr:rpartition

sdestr:rpartition

This Scheme extension splits the input string at the last occurrence of the separator string. It returns the part of the input string before the separator string, the separator itself, and the part after it. If the separator is not found, then the Scheme extension returns the separator and two empty strings.

Note:

The input parameter and the returned value of the `sdestr:rpartition` Scheme extension are the same as for the Python `rpartition` method (see <https://www.javatpoint.com/python-string-rpartition-method>).

Syntax

```
(sdestr:rpartition input-string separator)
```

Returns

List with three members

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
separator	Specifies the separator. Argument type: STRING

Examples

```
(sdestr:rpartition "Java is a programming language" "is")
> ("Java ", "is", " a programming language")

(sdestr:rpartition "Java is a programming language" "Java")
> ("", "Java", " is a programming language")

(sdestr:rpartition "Java is a programming language" "language")
> ("Java is a programming ", "language", "")

(sdestr:rpartition "Java is a programming language" "av")
> ("J", "av", "a is a programming language")
```

Appendix A: Commands

sdestr:rsplit

sdestr:rsplit

This Scheme extension separates the input string and returns a list. It splits the string from the right, using the separator string as a delimiter.

This Scheme extension works the same way as `sdestr:split` except that splitting the input string occurs from the right.

Note:

The input parameter and the returned value of the `sdestr:rsplit` Scheme extension are the same as for the Python `rsplit` method (see <https://www.javatpoint.com/python-string-rsplit-method>).

Syntax

```
(sdestr:rsplit input-string [separator] [ncount])
```

Returns

STRING LIST

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
separator	Optional. Specifies the separator. If the separator is not specified, then any space string is a separator. Argument type: STRING
ncount	Optional. If you specify this argument, then the Scheme extension replaces one string with a new string for the specified number of occurrences of the old string. If ncount is not specified, then all occurrences of the old string are replaced. Argument type: INTEGER

Examples

```
(sdestr:rsplit "Java is a programming language")
> ("Java" "is" "a" "programming" "language")
(sdestr:rsplit "Java is a programming language" "Java")
> (" " " is a programming language")
(sdestr:rsplit "Java is a programming language" "a")
> ("J" "v" " is " " progr" "mming l" "ngu" "ge")
(sdestr:rsplit "Java is a programming language" "a" 1)
> ("J" "va is a programming language")
(sdestr:rsplit "Java is a programming language" "a" 3)
> ("J" "v" " is " " programming language")
```

Appendix A: Commands

sdestr:rstrip

sdestr:rstrip

This Scheme extension is the same as `sdestr:split`, but it processes the input string in a backward direction.

Note:

The input parameter and the returned value of the `sdestr:rstrip` Scheme extension are the same as for the Python `rstrip` method (see <https://www.javatpoint.com/python-string-rstrip-method>).

Syntax

```
(sdestr:rstrip input-string [separator])
```

Returns

List of words in string

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
separator	Optional. Specifies the separator. If the separator is not specified, then the string splits according to space. Argument type: STRING

Examples

```
(sdestr:rstrip "Java and C# ")
> "Java and C#"

(sdestr:rstrip "Java and C#" "#")
> "Java and C"

(sdestr:rstrip "Java and C#" "#")
> "Java and C"
```

Appendix A: Commands

sdestr:split

sdestr:split

This Scheme extension splits the input string into a list, based on the delimiter specified as the separator string.

The input string is split according to the space if the delimiter is not specified. It returns a list of substrings concatenated with the delimiter.

Note:

The input parameter and the returned value of the sdestr:split Scheme extension are the same as for the Python `split` method (see <https://www.javatpoint.com/python-string-split-method>).

Syntax

```
(sdestr:split input-string [separator] [maxsplit])
```

Returns

STRING LIST

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
separator	Optional. Specifies the separator. If the separator is not specified, then the string splits according to space. Argument type: STRING
maxsplit	Optional. Argument type: INTEGER

Examples

```
(sdestr:split "Java is a programming language")
> ("Java" "is" "a" "programming" "language")
(sdestr:split "Java is a programming language" "Java")
> (" " " is a programming language")
(sdestr:split "Java is a programming language" "a")
> ("J" "v" " is " " progr" "mming l" "ngu" "ge")
(sdestr:split "Java is a programming language" "a" 1)
> ("J" "va is a programming language")
(sdestr:split "Java is a programming language" "a" 3)
> ("J" "v" " is " " programming language")
```

Appendix A: Commands

sdestr:splitlines

sdestr:splitlines

This Scheme extension splits the input string at line boundaries. It returns a list of strings at each line with new lines removed.

Elements that can break lines are:

- New line (\n)
- Return (\r)
- Return + line feed (\r\n)
- Line tabulation (\v or \x0b)
- Form feed (\f or \x0c)
- File separator (\x1c)
- Group separator (\x1d)
- Record Separator (\x1e)
- Next line (C1 control code) (\x85)
- Line separator (\u2028)
- Paragraph separator (\u2029)

Note:

The input parameter and the returned value of the sdestr:splitlines Scheme extension are the same as for the Python splitlines method (see <https://www.javatpoint.com/python-string-splitlines-method>).

Syntax

(sdestr:splitlines input-string)

Returns

STRING LIST

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Appendix A: Commands

sdestr:splitlines

Examples

```
(sdestr:splitlines "Java is a programming language")
> ("Java is a programming language")

(sdestr:splitlines "Java \n is a programming \r language")
> ("Java " " is a programming " " language")

(sdestr:splitlines "Java \n is a programming \r language")
> ("Java \n" " is a programming \r" " language")
(sdestr:splitlines "Java \n is a programming \r language for \r\n
    software development")
> ("Java " " is a programming " " language for " "
    software development")
```

Appendix A: Commands

sdestr:startswith

sdestr:startswith

This Scheme extension returns true if the input string starts with the specified prefix. Otherwise, it returns false.

Optionally, you can specify a start index from where search starts and an end index where the search stops.

Note:

The input parameter and the returned value of the sdestr:startswith Scheme extension are the same as for the Python `startswith` method (see <https://www.javatpoint.com/python-string-startswith-method>).

Syntax

```
(sdestr:startswith input-string prefix [start-index] [end-index])
```

Returns

BOOLEAN

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
prefix	Specifies the prefix. Argument type: STRING
start-index	Optional. Specifies the start index from where the search starts. Argument type: INTEGER
end-index	Optional. Specifies the end index from where the search stops. Argument type: INTEGER

Examples

```
(sdestr:startswith "Hello Javatpoint" "Hello")
> #t
(sdestr:startswith "Hello Javatpoint" "Java")
> #f
(sdestr:startswith "Hello Javatpoint" "Java" 6)
> #t
(sdestr:startswith "Hello Javatpoint" "Java" 6 10)
> #t
(sdestr:startswith "Hello Javatpoint" "Java" 6 12)
> #f
```

Appendix A: Commands

sdestr:swapcase

sdestr:swapcase

This Scheme extension converts uppercase alphabetic characters to lowercase and lowercase alphabetic characters to uppercase in the input string.

Note:

The input parameter and the returned value of the sdestr:swapcase Scheme extension are the same as for the Python swapcase method (see <https://www.javatpoint.com/python-string-swapcase-method>).

Syntax

(sdestr:swapcase input-string)

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:swapcase "HELLO JAVATPOINT")
> "hello javatpoint"
```

```
(sdestr:swapcase "hello javatpoint")
> "HELLO JAVATPOINT"
```

```
(sdestr:swapcase "Hello JavaTpoint")
> "hELLO jAVAtPOINT"
```

Appendix A: Commands

sdestr:upper

sdestr:upper

This Scheme extension converts any alphabetic characters in the input string to uppercase.

Note:

The input parameter and the returned value of the sdestr:upper Scheme extension are the same as for the Python `upper` method (see <https://www.javatpoint.com/python-string-upper-method>).

Syntax

(sdestr:upper input-string)

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING

Examples

```
(sdestr:upper "Hello javatpoint")
> "HELLO JAVATPOINT"
```

Appendix A: Commands

sdestr:zfill

sdestr:zfill

This Scheme extension adds zeros (0) to the left of the input string to a specified length.

It returns the input string padded with zeros, up to the specified length. However, if the specified length is less than the number of characters in the input string, then the Scheme extension returns the input string.

If the input string starts with a plus (+) or minus (-) sign, then the Scheme extension returns the input string padded with zeros, up to the specified length, with the plus or minus sign as the first character.

Note:

The input parameter and the returned value of the sdestr:zfill Scheme extension are the same as for the Python `zfill` method (see <https://www.javatpoint.com/python-string-zfill-method>).

Syntax

(sdestr:zfill input-string str-length)

Returns

STRING

Arguments

Argument	Description
input-string	Specifies an input string. Argument type: STRING
str-length	Specifies the length of the string. Argument type: INTEGER

Examples

```
(sdestr:zfill "Zfill Example" 20)
> "0000000Zfill Example"
(sdestr:zfill "Zfill Example" 5)
> "Zfill Example"
(sdestr:zfill "+100" 10)
> "+000000100"
(sdestr:zfill "-200" 10)
> "-000000200"
(sdestr:zfill "--Rohan--" 10)
> "-0-Rohan--"
```

Appendix A: Commands

set-interface-contact

set-interface-contact

This Scheme extension attaches a contact attribute to the common faces of the specified regions.

Note:

This Scheme extension applies to 3D models only.

Syntax

```
(set-interface-contact region1 region2 contact-name)
```

Returns

None

Arguments

Argument	Description
region1	Specifies a region name. Argument type: STRING
region2	Specifies a region name. Argument type: STRING
contact-name	Specifies the name of a contact. Argument type: STRING

Examples

```
(sde:clear)
(sdegeo:create-cuboid (position 0 0 0) (position 10 10 10)
    "Silicon" "mb1")
(sdegeo:create-sphere (position 10 10 0) 4 "PolySilicon" "mb2")
(sdegeo:define-contact-set "ifcontact" 4 (color:rgb 1 0 0) "##")
(set-interface-contact "mb1" "mb2" "ifcontact")
```

shell?

This Scheme extension determines whether a Scheme object is a shell.

It returns #t if the specified object is a shell; otherwise, it returns #f.

Syntax

(shell? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queries. Argument type: SCHEME OBJECT

Appendix A: Commands

skin:options

skin:options

This Scheme extension sets the options in the data structure to be used by skinning APIs.

Syntax

```
(skin:options {skin-option-name value} [skin-options])
```

Returns

sweep-options

Arguments

Argument	Description
skin-option-name	Specifies the name of a skinning option. See <i>Description</i> for a list of available options. Argument type: STRING
value	Specifies the value of a skinning option. Argument type: BOOLEAN
skin-options	Optional. Argument type: SKIN OPTIONS

Description

This Scheme extension sets the Scheme object `skin-options` that is used by the skinning and lofting Scheme extensions.

The following skinning options, for `skin-option-name` are available:

Skinning option	Description
align	(Boolean) Aligns the direction of the cross-section curves such that the normal of the first profile points towards the second profile. All other profiles are aligned to follow the first and second. If the sections are not oriented in the same direction, then the <code>align</code> option can be used to avoid producing a twisted, self-intersecting body. The default is <code>#t</code> .
allow_same_uv	(Boolean) Allows surfaces with the same <code>u</code> and <code>v</code> direction to be created. If the option is set to <code>#t</code> and a surface with the same <code>u</code> and <code>v</code> directions is created, then a warning is displayed. If set to <code>#f</code> , an error is displayed. Note: If this option is set to <code>#t</code> and a surface with the same <code>u</code> and <code>v</code> directions is created, modeling problems might appear later.

Appendix A: Commands

skin:options

Skinning option	Description
arc_length, arc_length_u	(Boolean) The <code>arc_length</code> option is used to select the arc length or isoparametric parameterization of the skinning surface. For basic skinning and lofting in isoparametric parameterization, the surface parameter in the <code>v</code> direction follows the cross-section curves. For arc-length parameterization, the surface parameter follows lines of constant length. The default is isoparametric parameterization. In the case of skinning with guide curves, with arc-length parameterization, the guide curve is arc-length parameterized; however, the surface is still isoparametric. The <code>arc_length_u</code> option reparameterizes curves of the skinning or lofting profiles to arc length. The default is <code>#f</code> for both options.
closed	(Boolean) Used to construct a solid body closed in <code>v</code> (that is, a torus). A solid body is constructed only when all the wires supplied are closed. At least three profiles must be provided to create a closed body. The default is <code>#f</code> .
estimate_loft_tanfac	(Boolean) When this option is switched on, the weight factor for the tangent continuous of the loft is determined such that it minimizes the average radius of curvature of the lofting surfaces. The resulting bodies should support shelling to greater thickness and also blending of their edges to larger blend radii. The default is <code>#f</code> .
gap_type	(String) Specifies the type of gap that is placed between the new faces. The type can be: <ul style="list-style-type: none"> "extended" (default) – extending the surfaces and intersecting "rounded" – tangent surface to both lateral faces "chamfered" – a linear fill between both lateral faces
guidePreference	(String) Specifies how an over-constrained guide is resolved. It is an enumerated type with the following possible values: <ul style="list-style-type: none"> If "constrain_to_guide" (default) is specified, then the resulting lofting surface always stays with the defining guide curve. If "constrain_to_tangent" is specified, then the lofting surface always follows the tangent constraint.
match_vertices	(Boolean) Suppresses the vertex-matching algorithm that ensures that all profiles consist of the same number of coedges. A heuristic approach is used to determine which vertex pairs are good matches. Profile coedges are then split where additional vertices are needed. This option is forced to <code>#t</code> if the coedge numbers of the profiles are not equal. The default is <code>#t</code> .
merge_wirecoedges	(Boolean) When this option is set to <code>#t</code> , the G1 vertices of the skinning and lofting wire profiles are removed by merging adjacent coedges/edges. This improves operations such as blending and shelling as it reduces the coedge/edge count and the number of surfaces, and eliminates near tangent edges. The default is <code>#t</code> .

Appendix A: Commands

skin:options

Skinning option	Description
no_new_twist_vertices	(Boolean) The algorithm that minimizes the surface twist can add vertices to some of the profiles if none of the existing vertices match well. This option allows you to force the algorithm to choose matching vertices from the existing vertices. The default is #f.
no_twist	(Boolean) Minimizes the twist of the surface produced. Twist minimization aligns closed curves such that the start of the second curve is aligned to the start of the first curve. Even if the shape of a body is unaffected by twisting, a surface with a twist could produce unexpected results when faceting and rendering. The default is #t.
periodic	(Boolean) Allows the construction of loft bodies that are periodic in v, that is, bodies that close back on themselves smoothly (continuously) at the start and end profiles. This option is activated in the skinning APIs by giving the closed option a value of 2. In Scheme, this is achieved by setting the periodic flag to #t. As for the closed option, at least three profiles must be supplied to create a periodic loft body.
perpendicular	(Boolean) The take-off vector is a tangent vector going out of the starting edge or surface and into the skinned or lofted surface. The perpendicular option (for lofting only) is used to specify the direction of the take-off vector, perpendicular to the coedge or in the loft direction. (This removes any restriction that the take-off vector for the loft has to be determined by the cross product of the coedge tangent vector and the surface normal scaled by the tangent factor.) The default is #f.
postprocess_stitch	(Boolean) Stitches the resulting lofting body to the original bodies from which its coedge definition came. This option only works with api_loft_coedges (lofting) and not with any skinning operation (api_skin_wires). It is identical in nature to the stitching operation performed in api_loft_faces. The default is #t.
self_int_test	(Integer) Checks for self-intersecting skin surfaces. Commonly, self-intersecting skin surfaces can be made based on poor tangent factor magnitudes, poor profiles, or an incorrect use of the perpendicular option. If set to 0, the check does not occur and a body is built; however, modeling problems might appear later. If set to 2, the check is performed but a warning message is displayed. The default is 1.

Appendix A: Commands

solid?

solid?

This Scheme extension determines whether a Scheme object is a solid.

Syntax

(solid? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

solid:area

This Scheme extension returns the surface area of a solid. It returns a pair of values: the total face area of the body and the relative accuracy (absolute accuracy/area) achieved in the computation.

Cases that are treated analytically (tolerance is 0) are planes with straight or elliptical edges, cones with straight edges, circular cylinders with elliptical edges, and special cases of latitudinal and longitudinal edges on spheres.

Syntax

(solid:area solid-body [tolerance=0.01])

Returns

REAL LIST

Arguments

Argument	Description
solid-body	Specifies a solid body. If the solid body entity has not been explicitly defined, then the argument should be (entity # [#]), where the first # is its entity number and the second # is its part number. Argument type: BODY
tolerance	Optional. Specifies the tolerance, that is, the accuracy of the calculation. Argument type: REAL

Appendix A: Commands

solid:massprop

solid:massprop

This Scheme extension analyzes the mass properties of a solid.

Syntax

```
(solid:massprop entity [compute-type])
```

Returns

LIST

Arguments

Argument	Description
entity	Specifies a solid body to compute the mass properties. Argument type: BODY
compute-type	Optional. Specifies the type of calculation to perform. Valid options are: <ul style="list-style-type: none">• 0: Volume and tolerance only• 1: Volume, tolerance, center of mass, principal moments, and principal axes• 2: Volume, center of mass, principal moments, principal axes, and inertial tensor Argument type: INTEGER

Appendix A: Commands

sort

sort

This Scheme extension sorts a supplied list based on the sort criteria.

Syntax

```
(sort sort-criteria sort-list)
```

Returns

REAL

Arguments

Argument	Description
sort-criteria	Specifies the criteria with which to sort the supplied list. The criteria can be <, <=, >, >= ... Argument type: STRING
sort-list	Specifies a list. Argument type: LIST

Errors

List

Examples

```
(sort < (list 1 2 3 4 9 8 7 6))
```

Appendix A: Commands

string:head

string:head

This Scheme extension returns a substring of the input string.

The returned string starts at the first character of the input string and contains `snum` number of characters. If `snum` is larger than the string length, an empty string is returned.

Syntax

`(string:head string snum)`

Returns

STRING

Arguments

Argument	Description
<code>string</code>	Specifies an input string. Argument type: STRING
<code>snum</code>	Specifies the number of characters to be returned. If <code>snum</code> is larger than the string length, then an empty string is returned. Argument type: INTEGER

Appendix A: Commands

string:tail

string:tail

This Scheme extension returns a substring of the input string.

The returned string starts at the `snum`-th character of the input string and contains `string` until the last character. If `snum` is larger than the string length, an empty string is returned.

Syntax

```
(string:tail string snum)
```

Returns

STRING

Arguments

Argument	Description
string	Specifies an input string. Argument type: STRING
snum	Specifies the number of characters to be returned. If <code>snum</code> is larger than the string length, then an empty string is returned. Argument type: INTEGER

Appendix A: Commands

sweep:law

sweep:law

This Scheme function creates a surface or solid by sweeping a profile along a path. The path can be defined as a path, a distance, a vector, or an axis (position and vector).

This Scheme function is complex. In some cases, the provided input defines topologically correct output. However, it can cause self-intersections, in which case, the function fails.

See [sweep:options on page 840](#).

Syntax

```
(sweep:law profile {path | distance | vector | axis} [sweep-options])
```

Returns

ENTITY

Arguments

Argument	Description
profile	Specifies a profile. It is a pointer to a wire body, a face, or an edge that, in turn, defines the sweep geometry and becomes the base of the solid or the edge of the surface. A planar sheet body containing nonadjacent faces is also permissible. Argument type: WIRE BODY FACE EDGE
path	Specifies wire body or an edge along which the profile is swept. It can be defined as a distance if <code>profile</code> is a planar face. It can be defined as a vector if "rail_law" is used (no twist). It can be defined as an axis (defined as a position and vector) if "rail_law" is used (no twist). Argument type: WIRE BODY EDGE
distance	Defines the distance to sweep along the face normal. Argument type: REAL
vector	Defines the direction and distance to sweep. Argument type: GVECTOR
axis	Specifies is a position and vector that defines the axis to revolve around. The amount to revolve is controlled by the "sweep_angle" option. Argument type: POSITION GVECTOR
sweep-options	Optional. This argument contains the <code>Sweep_Options</code> data structure. This data structure is created by <code>sweep:options</code> . If <code>sweep-options</code> is not specified, the default sweep option values are used. Argument type: SWEEP OPTIONS

Appendix A: Commands

sweep:law

Examples

Sweeping an Edge Along a Vector

```
(sde:clear)
(define edge1 (wire-body (edge:linear (position 10 0 0)
                                         (position 10 10 0))))
(define sweep1 (sweep:law edge1 (gvector 10 0 0)))
```

Sweeping a Face Along a Distance

```
(sde:clear)
(define face2 (face:plane (position 0 20 0) 10 10 (gvector 0 0 1)))
(define sweep2 (sweep:law face2 5))
```

Revolving a Face

```
(sde:clear)
; Create a solid block.
(define block1 (solid:block (position -10 -10 0) (position 25 25 25)))
; Separate faces from block1.
(define entities (entity:faces block1))
; Extract a single planar face.
(define facel (car (cdr (cdr entities))))
; Verify transform applied.
(define fix (entity:fix-transform block1))
; Define extent of sweep.
(define opts (sweep:options "sweep_angle" 60))
; Revolve the planar face by a gvector around a position.
(define sweep3 (sweep:law facel (position -10 -10 -10)
                           (gvector 1 0 0) opts))
```

Revolving a Wire

```
(sde:clear)
; Create 4 linear edges.
(define edge1 (edge:linear (position 0 0 0) (position 20 0 0)))
(define edge2 (edge:linear (position 20 0 0) (position 20 20 0)))
(define edge3 (edge:linear (position 20 20 0) (position 0 20 0)))
(define edge4 (edge:linear (position 0 20 0) (position 0 0 0)))
; Create a wire body from the 4 edges.
(define wirebody (wire-body (list edge1 edge2 edge3 edge4)))
; Create a solid by revolving the wire body.
(define sweep4 (sweep:law wirebody (position 0 0 0) (gvector 0 1 0)
                           sweep:options "sweep_angle" 60)))
```

Sweeping a Planar Face

```
(sde:clear)
; Create a solid block.
(define block1 (solid:block (position -10 -10 -10)
                           (position 30 30 30)))
; Get a list of the faces of the solid block.
```

Appendix A: Commands

sweep:options

```
(define faces (entity:faces block1))
; Select a single face to sweep.
(define face (car (cdr (cdr faces))))
; Create a new solid by sweeping the face along a gvector.
(define sweep5 (sweep:law face (gvector 0 -5 0)
(sweep:options "draft_angle" -45)))
```

sweep:options

This Scheme function sets the options for `sdegeo:sweep`. It defines elements in the `sweep-options` data structure that are used later for the `sdegeo:sweep` operation. See [sdegeo:sweep on page 701](#).

Multiple pairs of `sweep-option-name` and `value` can be specified simultaneously. For example:

```
(sweep:options "draft_angle" 5 "solid" #f)
```

Syntax

```
(sweep:options sweep-option-name {value | location direction})
```

Returns

`sweep-options`

Arguments

Argument	Description
<code>sweep-option-name</code>	<p>Specifies an option. The following sweep options can be used:</p> <ul style="list-style-type: none">• "draft_angle"• "gap_type"• "miter"• "rigid"• "sweep_angle" <p>The option "draft_angle" is a real number that represents the angle with which the swept profile is to draft while sweeping. Drafting has two mutually exclusive options: "draft_angle" and "draft_law". The default for "draft_angle" is 0.</p> <p>One application of drafting is for molded items. As the profile is swept, the ending profile has been offset by an equal distance, which helps with the removal of the item from a mold. Extreme draft angles or draft laws can result in errors due to self-intersecting bodies, incomplete lateral faces (likely at corners), or unsupported topologies. Argument type: STRING</p>

Appendix A: Commands

sweep:options

Argument	Description
value	<p>Specifies a value for the named option.</p> <p>If value is a real, it does not require delimiters.</p> <p>If value is a string representing a law, it should be enclosed in double quotation marks.</p> <p>If value is a law, only the variable name for the law is required.</p> <p>Argument type: STRING law REAL GVECTOR ENTITY BOOLEAN INTEGER</p>
location	Specifies a location. Argument type: POSITION
direction	Specifies a direction. Argument type: GVECTOR

Examples

```
(define plist1 (list (position 0 0 0) (position 20 0 0)
                      (position 20 20 0) (position 20 20 20)))
(define start1 (gvector 1 0 0))
(define end1 (gvector 0 0 10))
(define path1 (edge:spline plist1 start1 end1))
(render:rebuild)
(define edgelist1 (list
    (edge:linear (position 0 3 3) (position 0 3 -3))
    (edge:linear (position 0 3 -3) (position 0 -3 -3))
    (edge:linear (position 0 -3 -3) (position 0 -3 3))
    (edge:linear (position 0 -3 3) (position 0 3 3))))
(define profile1 (wire-body edgelist1))
(define sweep1 (sweep:law profile1 path1))
```

Appendix A: Commands

system:command

system:command

This Scheme function executes a system command from the Scheme command-line window. The command results are displayed in the UNIX window. If you use the system:command Scheme function, then the return value must be processed correctly; otherwise, Sentaurus Workbench cannot find possible errors. For details, see [Error Signaling to Sentaurus Workbench on page 319](#).

Syntax

(system:command cmd)

Returns

None

Arguments

Argument	Description
cmd	Specifies a system command. Argument type: STRING

system:getenv

This Scheme function returns the value of an environment variable.

Syntax

(system:getenv name-string)

Returns

Value of the environment variable if it is found; otherwise, it returns #f

Arguments

Argument	Description
name-string	Specifies a name of an environment variable. Argument type: STRING

Appendix A: Commands

timer:end

timer:end

This Scheme command stops the timer.

The commands `timer:start`, `timer:end`, `timer:show-time`, and `timer:get-time` are used to measure the performance of a command or a series of commands. They measure only the CPU time required to execute the command and not any delays incurred from entering the commands into Scheme.

Syntax

`(timer:end)`

Returns

STRING

timer:get-time

This Scheme command calculates and returns the amount of time elapsed since `timer:start` was executed.

The commands `timer:start`, `timer:end`, `timer:show-time`, and `timer:get-time` are used to measure the performance of a command or a series of commands. They measure only the CPU time required to execute the command and not any delays incurred from entering the commands into Scheme. They are used most often to determine the time required to execute one or more commands.

The command `timer:get-time` can be executed or interspersed any number of times throughout a command or a series of commands. You also can use `timer:show-time` to display the amount of time elapsed since `timer:start` was executed.

Syntax

`(timer:get-time)`

Returns

REAL

Appendix A: Commands

timer:show-time

timer:show-time

This Scheme command calculates and returns the amount of time elapsed since timer:start was executed.

The commands timer:start, timer:end, timer:show-time, and timer:get-time are used to measure the performance of a command or a series of commands. They measure only the CPU time required to execute the command and not any delays incurred from entering the commands into Scheme, or any other interference or condition.

The command timer:show-time can be executed or interspersed any number of times throughout a command or a series of commands. It returns a real number only after timer:start is executed. If timer:start has not been executed, timer:show-time returns zero (0). You also can use timer:get-time to display the amount of time elapsed since timer:start was executed.

Syntax

(timer:show-time)

Returns

REAL

timer:start

This Scheme command starts an internal clock or timer.

The commands timer:start, timer:end, timer:show-time, and timer:get-time are used to measure the performance of a command or a series of commands. They measure only the CPU time required to execute the command and not any delays incurred from entering the commands into Scheme.

Syntax

(timer:start)

Returns

STRING

Appendix A: Commands

transform:reflection

transform:reflection

This Scheme function creates a transform to mirror an object through an axis. It is used to define a reflection transform, which can be used in sdegeo:mirror-selected.

Syntax

```
(transform:reflection plane-position plane-direction)
```

Returns

transform

Arguments

Argument	Description
plane-position	Specifies the location to mirror an object. Argument type: POSITION
plane-direction	Specifies the normal of the mirror in the plane. Argument type: GVECTOR

Examples

```
(sde:clear)
(define mb (sdegeo:create-rectangle (position 1 1 0) (position 2 2 0)
                                      "Silicon" "xx"))
(define tr1 (transform:reflection (position 0 0 0) (gvector -1 0 0)))
(set! tr1 (transform:reflection (position 0 0 0) (gvector 0 1 0)))
(sdegeo:mirror-selected (get-body-list) tr1 #t)
```

Appendix A: Commands

transform:rotation

transform:rotation

This Scheme function creates a transform to rotate an object about an axis. It defines a rotation transform, which can be used in sdegeo:rotate-selected.

Syntax

```
(transform:rotation origin-position axis-direction angle)
```

Returns

transform

Arguments

Argument	Description
origin-position	Specifies the start location of the rotation. Argument type: POSITION
axis-direction	Specifies the axis direction of rotation. Argument type: GVECTOR
angle	Specifies the angle in degrees to rotate the object. Argument type: REAL

Examples

```
(sde:clear)
(sdegeo:create-circle (position 0 0 0) 0.1 "PolySilicon" "xx")
(define mb (sdegeo:create-rectangle (position 1 1 0) (position 2 2 0)
    "Silicon" "xx"))
(define tr1 (transform:rotation (position 0 0 0) (gvector 0 0 1) 45))
(sdegeo:rotate-selected mb tr1 #t 7)
```

Appendix A: Commands

transform:scaling

transform:scaling

This Scheme function defines a scaling transform, which can be used in sdegeo:scale-selected.

Although the Scheme function accepts a scaling factor (x-scale, y-scale, or z-scale) 0 or less, only a positive scale factor is recommended. When uniform scaling is used, only the x-scale term can be supplied.

In this case, all three components of the gvector are multiplied by the same x-scale factor.

Syntax

```
(transform:scaling x-scale [y-scale z-scale])
```

Returns

transform

Arguments

Argument	Description
x-scale	Specifies the scaling factor for the x-axis direction. Argument type: REAL
y-scale	Optional. Specifies the scaling factor for the y-axis direction. Argument type: REAL
z-scale	Optional. Specifies the scaling factor for the z-axis direction. Argument type: REAL

Examples

```
(sde:clear)
(define mb (sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
                                      "Silicon" "xx"))
(define tr1 (transform:scaling 1.0 2.0 1.0))
(sdegeo:scale-selected mb tr1)
```

Appendix A: Commands

transform:translation

transform:translation

This Scheme function is used to define a translation transform, which can be used in sdegeo:translate-selected.

Syntax

```
(transform:translation gvector)
```

Returns

transform

Arguments

Argument	Description
gvector	Argument type: GVECTOR

Examples

```
(sde:clear)
(define mb (sdegeo:create-rectangle (position 0 0 0) (position 1 1 0)
                                      "Silicon" "xx"))
(define tr1 (transform:translation (gvector 3.0 0.0 0.0)))
(sdegeo:translate-selected mb tr1 #t 5)
```

Appendix A: Commands

util:make-bot-contact

util:make-bot-contact

This Scheme extension assigns a contact to the bottom of a 2D or 3D device. In two dimensions, the bottom edges of the device are defined as contacts. In three dimensions, the bottom faces of the device are defined as contacts.

Syntax

```
(util:make-bot-contact contact-name [contact-argument])
```

Returns

None

Arguments

Argument	Description
contact-name	Specifies the name of a contact. If the given contact is not yet defined, then util:make-bot-contact will define the contact. Argument type: STRING
contact-argument	Optional. Specifies either a color (color name or RGB color) in two dimensions or a face pattern string ("solid" "##" " " "==" " // " "\\" ":" "<><>" "[] []") in three dimensions. Argument type: INTEGER RGB COLOR (2D) or STRING (3D)

Examples

Example: Two Dimensions

```
(sde:clear)
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "x1")
(sdegeo:create-rectangle (position 3 10 0) (position 7 14 0)
    "PolySilicon" "x2")
(sdegeo:insert-vertex (position 1 0 0))
(sdegeo:insert-vertex (position 5 0 0))
(sdegeo:insert-vertex (position 9 0 0))
(sdegeo:insert-vertex (position 4 14 0))
(sdegeo:insert-vertex (position 5 14 0))
(sdegeo:insert-vertex (position 6 14 0))

(util:make-bot-contact "c1" GREEN)
(util:make-top-contact "c2" BLUE)
```

Example: Three Dimensions

```
(sde:clear)
(sde:set-process-up-direction "+z")
(sdegeo:create-rectangle (position 0 0 0) (position 10 10 0)
    "Silicon" "x1")
```

Appendix A: Commands

util:make-top-contact

```
(sdegeo:create-rectangle (position 3 10 0) (position 7 14 0)
    "PolySilicon" "x2")
(sdegeo:insert-vertex (position 1 0 0))
(sdegeo:insert-vertex (position 5 0 0))
(sdegeo:insert-vertex (position 9 0 0))
(sdegeo:insert-vertex (position 4 14 0))
(sdegeo:insert-vertex (position 5 14 0))
(sdegeo:insert-vertex (position 6 14 0))
(entity:rotate (get-body-list) 0 0 0 1 0 0 90)
(sdegeo:extrude (get-body-list) 5)

(util:make-bot-contact "c1" "##")
(util:make-top-contact "c2" "//")
```

util:make-top-contact

This Scheme extension assigns a contact to the top of a 2D or 3D device. In two dimensions, the top edges of the device are defined as contacts. In three dimensions, the top faces of the device are defined as contacts.

Syntax

```
(util:make-top-contact contact-name [contact-argument])
```

Returns

None

Arguments

Argument	Description
contact-name	Specifies the name of a contact. If the given contact is not yet defined, then util:make-top-contact will define the contact. Argument type: STRING
contact-argument	Optional. Specifies either a color (color name or RGB color) in two dimensions or a face pattern string ("solid" "##" " " "=" " // " "\\" ": :" "<><>" "[] []") in three dimensions. Argument type: INTEGER RGB COLOR (2D) or STRING (3D)

Examples

See examples of [util:make-bot-contact on page 849](#).

Appendix A: Commands

vertex?

vertex?

This Scheme function determines whether a Scheme object is a vertex.

Syntax

(vertex? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

view:set-point-size

This Scheme function sets the size of the rendered vertices in the main window of the GUI.

Syntax

(view:set-point-size vertex-size)

Returns

None

Arguments

Argument	Description
vertex-size	Specifies the vertex size. The default is 10, and 0 switches off the vertex rendering and removes the vertex markings. Argument type: INTEGER

Appendix A: Commands

wire?

wire?

This Scheme function determines whether a Scheme object is a wire.

Syntax

(wire? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

wire-body?

This Scheme function determines whether a Scheme object is a wire body.

Syntax

(wire-body? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

Appendix A: Commands

wire:planar?

wire:planar?

This Scheme function determines whether a Scheme object is a planar wire.

Syntax

(wire:planar? object)

Returns

BOOLEAN

Arguments

Argument	Description
object	Specifies the Scheme object to be queried. Argument type: SCHEME OBJECT

Index of Scheme Extensions

A

afm-smooth-layers 107, 342

B

bbox 343
bbox-exact 344
body? 331, 345
build-csv-lens 346

C

color:rgb 228, 347
complete-edge-list 348
convert-to-degree 349
convert-to-radian 349
curve:circular? 331
curve:elliptical? 331
curve:linear? 331
curve? 331

D

define 63

E

edge:circular 351
edge:circular? 331, 352
edge:curve? 331
edge:elliptical? 331, 352
edge:end 353
edge:length 353
edge:linear 156, 354
edge:linear? 331, 355
edge:mid-point 356
edge:spline? 331
edge:start 356
edge:type 357
edge? 331, 350
entity:box 330, 357

entity:copy 358
entity:debug 328, 359
entity:deep-copy 360
entity:delete 189, 360
entity:dist 361
entity:edges 329, 330, 362
entity:erase 362
entity:faces 156, 329, 363
entity:loops 329, 364
entity:lumps 329, 365
entity:set-color 366
entity:shells 329, 367
entity:vertices 329, 368
entity? 331
env:set-tolerance 369
env:tolerance 370
erf 370
erfc 371
exists-empty-mask-name 168, 371
exists-mask-name 168, 372
extract-interface-normal-offset-refwindow 243, 373
extract-interface-offset-refwindow 243, 374
extract-refpolyhedron 242, 375
extract-refwindow 242, 376

F

face:area 377
face:conical? 331, 378
face:cylindrical? 331, 378
face:planar? 331, 379
face:plane-normal 379
face:spherical? 331, 380
face:spline? 332, 380
face:toroidal? 332, 381
face? 332
filter:type 323
find-body-id 324, 382
find-body-id-drs 383

Index of Scheme Extensions

find-drs-id 383
find-edge-id 324, 384
find-edge-id-drs 385
find-face-id 117, 324, 330, 385
find-face-id-drs 386
find-mask 168
find-material-id 324, 325, 386
find-region 387
find-region-id 324, 325, 388
find-vertex-id 324, 325, 388
find-vertex-id-drs 389

G

generic:get 325
get-body-list 389
get-drs-list 390
get-empty-mask-list 168, 390
get-mask-list 168, 390
gvector 391
gvector:- 393
gvector:+ 392
gvector:copy 393
gvector:cross 394
gvector:dot 394
gvector:from-to 395
gvector:length 395
gvector:parallel? 396
gvector:perpendicular? 396
gvector:reverse 397
gvector:scale 397
gvector:set! 398
gvector:set-x! 399
gvector:set-y! 400
gvector:set-z! 401
gvector:transform 402
gvector:unitize 402
gvector:x 403
gvector:y 403
gvector:z 404
gvector? 392

J

journal:abort 404
journal:append 405

journal:clean 56, 406
journal:load 56, 407
journal:off 56, 407
journal:on 56, 408
journal:pause 56, 409
journal:resume 56, 409
journal:save 56, 409
journal:step 55, 410

L

load 54
loop:external? 332, 411
loop? 332, 411
lump? 332, 412

M

mask-refevalwin-extract-2d 168, 413
mask-refevalwin-extract-3d 168, 414
member? 415
merge-collinear-edges-2d 98, 103, 416

O

option:set 275

P

part:entities 323, 328, 417
part:load 54, 418
part:save 419
part:save-selection 420
part:set-name 421
PolygonSTI 214
PolygonWaferMask 214
PolyHedronCylinder 215
PolyHedronEllipticCylinder 215
PolyHedronEpiDiamond 217
PolyHedronSTI 217
PolyHedronSTIacc 219
PolyHedronSTIaccv 220
position 422
position:- 424
position:+ 424
position:distance 425
position:set! 426

Index of Scheme Extensions

position:set-x! 427
position:set-y! 427
position:set-z! 428
position:x 428
position:y 429
position:z 429
position? 332, 423
protect-all-contacts 234, 430

R

random-sd 431
remove-body-ABA 81, 432
remove-body-BAB 81, 433
render:rebuild 69, 275, 434
roll 56, 434

S

sde:add-material 143, 144, 435
sde:back-coord 436
sde:bg-image-transparency 334, 436
sde:bool-regularise 437
sde:bottom-coord 437
sde:build-mesh 276, 438
sde:change-datex-color-scheme 67, 439
sde:check-3d-license-status 439
sde:check-model 440
sde:checkout-3d-license 440
sde:clear 77, 441
sde:create-bg-image 333, 442
sde:create-dialog 334, 443
sde:define-parameter 62, 444
sde:delay-graphics-update 445
sde:delay-graphics-update? 445
sde:delete-bg-image 333, 446
sde:delete-materials 446
sde:delete-parameter 62, 447
sde:dialog-add-input 334, 448
sde:dialog-add-pixmap 334, 449
sde:dialog-delete 334, 450
sde:dialog-ok-command 334, 451
sde:dialog-show 334, 452
sde:display 452
sde:display-err 453
sde:display-std 454

sde:draw-ruler 455
sde:dump-non-default-options 457
sde:exact-coords? 457
sde:extract-tdr-boundary 458
sde:fix-imprint 458
sde:fix-orientation 459
sde:front-coord 459
sde:ft_scalar 460
sde:get-backwards-compatibility 461
sde:get-datex-color-scheme 67, 461
sde:get-default-material 327, 462
sde:get-view-params 60, 462
sde:gui-get-integer 337, 463
sde:gui-get-real 337, 464
sde:gui-get-string 337, 465
sde:hide 466
sde:hide-bg-image 333, 466
sde:hide-contact 72, 467
sde:hide-interface 467
sde:hide-mask 168, 468
sde:hide-material 469
sde:hide-region 470
sde:hide-ruler 471
sde:info 326, 471
sde:left-coord 472
sde:load-sat 55, 472
sde:material-type 473
sde:max-x 473
sde:max-y 474
sde:max-z 474
sde:merge-materials 475
sde:min-x 476
sde:min-y 476
sde:min-z 477
sde:new-region-name 477
sde:off-lights 477
sde:offset-mask 168, 170, 478
sde:on-lights 478
sde:open-model 53, 58, 479
sde:part-load 480
sde:pick-point-on-wp 337, 481
sde:pick-two-points-on-wp 338, 481
sde:post-message 338, 482
sde:project-name 482
sde:refresh 483

Index of Scheme Extensions

sde:rename-regions 484
sde:restore-cursor 484
sde:right-coord 485
sde:save-model 53, 58, 486
sde:save-parameters 487
sde:save-tcl-parameters 488
sde:scale-scene 72, 490
sde:scmwin-get-font-families 57, 490
sde:scmwin-get-font-family 57, 491
sde:scmwin-get-font-size 58, 491
sde:scmwin-get-font-style 58, 491
sde:scmwin-get-window-height 58, 492
sde:scmwin-select-font 58, 492
sde:scmwin-set-font-family 58, 492
sde:scmwin-set-font-size 58, 493
sde:scmwin-set-prefs 58, 493
sde:scmwin-set-window-height 58, 494
sde:scmwin-suppress-output 494
sde:selected-entities 37, 323, 495
sde:selected-refeval-windows 495
sde:separate-lumps 145, 321, 327, 496
sde:set-background-color 58, 498
sde:set-backwards-compatibility 499
sde:set-default-material 327, 500
sde:set-menubar-font-size 59, 500
sde:set-process-up-direction 167, 501
sde:set-project-name 502
sde:setrefprops 271, 509
sde:set-region-counter 326
sde:set-rendering-mode 502
sde:set-selection-level 323, 503
sde:set-translucency 504
sde:setup-grid 510
sde:set-view-mode 505
sde:set-view-operator 505
sde:set-view-params 60, 506
sde:set-window-position 59, 507
sde:set-window-size 59, 507
sde:set-window-style 508
sde:show 511
sde:showattribs 512
sde:show-bg-image 333, 512
sde:show-contact 72, 513
sde:show-grid 513
sde:show-interface 514
sde:show-mask 168, 514
sde:show-material 515
sde:show-pcurves 516
sde:show-region 517
sde:split-solid 518
sde:stripextension 519
sde:substring 519
sde:test-entity 520
sde:toggle-lights 520
sde:top-coord 521
sde:tr-get 521
sde:use-camera-manipulator 50, 65
sde:view-filter-reset 522
sde:view-set-light-intensity 522
sde:view-set-visible-area 523
sde:wait-cursor 523
sde>window-select-2d 103, 524
sde>window-select-3d 525
sde:xshow 526
sde:xshow-contact 72, 527
sde:xshow-interface 528
sde:xshow-mask 168, 529
sde:xshow-material 530
sde:xshow-region 531
sde:zoom-all 532
sdedr:append-cmd-file 532
sdedr:clear 533
sdedr:clear-multibox-definitions 533
sdedr:clear-multibox-placements 533
sdedr:clear-profile-definitions 534
sdedr:clear-profile-placements 534
sdedr:clear-refinement-definitions 535
sdedr:clear-refinement-placements 535
sdedr:clear-ref-windows 534
sdedr:clear-submesh-placement-transform 536
sdedr:convert-mask-to-drs-body 536
sdedr:define-1d-external-profile 260, 537
sdedr:define-analytical-profile 260, 539
sdedr:define-analytical-profile-placement 240,
 260, 540
sdedr:define-body-interface-refwin 242, 542
sdedr:define-constant-profile 255, 543
sdedr:define-constant-profile-material 255, 544
sdedr:define-constant-profile-placement 255, 545
sdedr:define-constant-profile-region 256, 546

Index of Scheme Extensions

sdedr:define-erf-profile 259, 547
sdedr:define-gaussian-profile 259, 548
sdedr:define-multibox-placement 550
sdedr:define-multibox-size 251, 551
sdedr:define-particle-profile 269, 552
sdedr:define-particle-profile-placement 269, 553
sdedr:define-refeval-window 239, 240, 241, 248, 251, 255, 269, 554
sdedr:define-refinement-function 248, 557
sdedr:define-refinement-material 248, 558
sdedr:define-refinement-placement 248, 559
sdedr:define-refinement-region 248, 560
sdedr:define-refinement-size 248, 561
sdedr:define-submesh 263, 562
sdedr:define-submesh-placement 263, 563
sdedr:delete-multibox-placement 564
sdedr:delete-profile-placement 565
sdedr:delete-refeval-window 244, 565
sdedr:delete-refinement-placement 566
sdedr:delete-submesh-placement 567
sdedr:del-selected-drentity 564
sdedr:get-cmdprecision 567
sdedr:get-definition-list 567
sdedr:get-placement-list 568
sdedr:hide-mbox 568
sdedr:hide-profile 569
sdedr:hide-refinement 569
sdedr:hide-rewin 570
sdedr:offset-block 253, 571
sdedr:offset-interface 253, 572
sdedr:read-cmd-file 54, 573
sdedr:redefine-refeval-window 574
sdedr:refine-box 575
sdedr:refine-doping 576
sdedr:refine-interface 577
sdedr:set-cmdprecision 578
sdedr:set-title 579
sdedr:show-mbox 579
sdedr:show-profile 580
sdedr:show-refinement 580
sdedr:show-rewin 581
sdedr:transform-submesh-placement 263, 582
sdedr:write-cmd-file 583
sdedr:write-scaled-cmd-file 155, 583
sdeepi:create-layerstack 287, 294, 584
sdeepi:publish-global-vars 307, 585
sdeepi:scm 586
sdeepi:tcl 587
sdegeo:2d-cut 91, 588
sdegeo:3d-cut 146, 589
sdegeo:align-horizontal 93, 103, 591
sdegeo:align-horizontal-aut 93, 103, 592
sdegeo:align-to-line 94, 103, 593
sdegeo:align-vertical 93, 103, 594
sdegeo:align-vertical-aut 93, 103, 595
sdegeo:average-edge-length 94, 102, 103, 596
sdegeo:body-trim 597
sdegeo:bool-intersect 142, 598
sdegeo:bool-subtract 142, 599
sdegeo:bool-unite 141, 600
sdegeo:break-nearly-axis-aligned-edges 100, 103, 601
sdegeo:chamfer 110, 602
sdegeo:chamfer-2d 90, 604
sdegeo:check-overlap 81, 605
sdegeo:chop-domain 151, 606
sdegeo:chull2d 91, 607
sdegeo:contact-sets 607
sdegeo:create-circle 608
sdegeo:create-cone 609
sdegeo:create-cuboid 105, 117, 611
sdegeo:create-cylinder 105, 612
sdegeo:create-ellipse 614
sdegeo:create-ellipsoid 106, 615, 616
sdegeo:create-linear-edge 617
sdegeo:create-ot-ellipsoid 617
sdegeo:create-ot-sphere 619
sdegeo:create-polygon 620
sdegeo:create-polyline-wire 119, 621
sdegeo:create-prism 622
sdegeo:create-pyramid 623
sdegeo:create-rectangle 145, 625
sdegeo:create-reg-polygon 626
sdegeo:create-ruled-region 627
sdegeo:create-sphere 106, 628
sdegeo:create-spline-wire 119, 629
sdegeo:create-torus 630
sdegeo:create-triangle 85, 632
sdegeo:curve-intersect 633
sdegeo:define-3d-contact-by-polygon 634

Index of Scheme Extensions

sdegeo:define-contact-set 228, 635
sdegeo:define-coord-sys 155, 636
sdegeo:define-work-plane 152, 637
sdegeo:delete-collinear-edges 95, 103, 639
sdegeo:delete-contact-boundary-edges 231, 639
sdegeo:delete-contact-boundary-faces 231, 640
sdegeo:delete-contact-edges 231, 641
sdegeo:delete-contact-faces 231, 641
sdegeo:delete-contact-set 229, 642
sdegeo:delete-coord-sys 155, 642
sdegeo:delete-edges 103, 643
sdegeo:delete-nearly-collinear-edges 96, 103, 644
sdegeo:delete-region 117, 144, 645
sdegeo:delete-short-edges 94, 97, 104, 645
sdegeo:delete-vertices 88, 95, 104, 646
sdegeo:delete-work-plane 154, 646
sdegeo:del-short-edges 99, 104, 638
sdegeo:distance 328, 647
sdegeo:dnce 98, 104, 647
sdegeo:extend-device 150, 648
sdegeo:extrude 120, 650
sdegeo:face-find-interior-point 651
sdegeo:fillet 111, 330, 652
sdegeo:fillet-2d 89, 654
sdegeo:find-closest-edge 325, 655
sdegeo:find-closest-face 325, 655
sdegeo:find-closest-vertex 325, 656
sdegeo:find-touching-faces 657
sdegeo:find-touching-faces-global 658
sdegeo:get-active-work-plane 658
sdegeo:get-auto-region-naming 326, 659
sdegeo:get-contact-edgelist 659
sdegeo:get-contact-facelist 660
sdegeo:get-current-contact-set 660
sdegeo:get-default-boolean 661
sdegeo:get-region-counter 661
sdegeo:imprint-circular-wire 233, 662
sdegeo:imprint-contact 663
sdegeo:imprint-polygonal-wire 233, 664
sdegeo:imprint-rectangular-wire 233, 665
sdegeo:imprint-triangular-wire 666
sdegeo:insert-vertex 85, 232, 666
sdegeo:max-edge-length 102, 104, 667
sdegeo:min-edge-length 102, 104, 668
sdegeo:mirror-selected 163, 164, 669
sdegeo:move-2d-regions 87, 670
sdegeo:move-edge 87, 671
sdegeo:move-vertex 86, 672
sdegeo:point-entity-relationship 673
sdegeo:polygonal-split 92, 674
sdegeo:prune-vertices 98, 104, 675
sdegeo:ray-test 676
sdegeo:reflect 677
sdegeo:rename-contact 678
sdegeo:revolve 679
sdegeo:rotate-selected 162, 164, 167, 680
sdegeo:scale 155, 682
sdegeo:scale-selected 158, 161, 164, 683
sdegeo:set-active-coord-sys 155, 684
sdegeo:set-active-work-plane 152, 684
sdegeo:set-auto-region-naming 326, 685
sdegeo:set-contact 230, 231, 686
sdegeo:set-contact-boundary-edges 687
sdegeo:set-contact-boundary-faces 688
sdegeo:set-contact-edges 689
sdegeo:set-contact-faces 690
sdegeo:set-contact-faces-by-polygon 691
sdegeo:set-current-contact-set 692
sdegeo:set-default-boolean 80, 145, 693
sdegeo:set-region-counter 694
sdegeo:set-region-counter-aut 695
sdegeo:skin-wire 137
sdegeo:skin-wire-guide 139
sdegeo:skin-wires 696
sdegeo:skin-wires-guides 697
sdegeo:skin-wires-normal 138, 698
sdegeo:skin-wires-vectors 139, 699
sdegeo:split-insert-device 148, 700
sdegeo:sweep 123, 125, 127, 130, 701
sdegeo:sweep-corner 128, 703
sdegeo:taper-faces 113, 117, 118, 704
sdegeo:translate 705
sdegeo:translate-selected 164, 706
sdegeo:vsmooth 101, 104, 707
sdeicwb:clear 707
sdeicwb:contact 168, 175, 708
sdeicwb:create-boxes-from-layer 713
sdeicwb:define-refinement-from-layer 175, 714
sdeicwb:down 175, 717

Index of Scheme Extensions

sdeicwb:gds2mac 168, 718
sdeicwb:generate-mask-by-layer-name 167, 168, 173, 721
sdeicwb:get-back 173, 722
sdeicwb:get-dimension 722
sdeicwb:get-domains 723
sdeicwb:get-front 173, 723
sdeicwb:get-global-bot 174, 724
sdeicwb:get-global-top 174, 724
sdeicwb:get-label 725
sdeicwb:get-label-for-layer 725
sdeicwb:get-labels 726
sdeicwb:get-layer-ids 172, 726
sdeicwb:get-layer-names 171, 726
sdeicwb:get-layer-polygon-midpoints 727
sdeicwb:get-left 173, 728
sdeicwb:get-polygon-bounding-boxes-by-layer-name 728
sdeicwb:get-polygon-by-name 729
sdeicwb:get-polygon-names-by-layer-name 729
sdeicwb:get-region-bot 174, 730
sdeicwb:get-region-top 174, 731
sdeicwb:get-right 173, 732
sdeicwb:get-xmax 173, 732
sdeicwb:get-xmin 173, 733
sdeicwb:get-ymax 173, 733
sdeicwb:get-ymin 173, 734
sdeicwb:get-zmax 173, 734
sdeicwb:get-zmin 173, 735
sdeicwb:load-file 171, 736
sdeicwb:mapreader 737
sdeicwb:set-domain 172, 738
sdeicwb:stretch 172, 738
sdeicwb:up 175, 739
sdeio:read-dfise-mask 54, 168, 740
sdeio:read-tdr 741
sdeio:read-tdr-bnd 54, 742
sdeio:save-1d-tdr-bnd 743
sdeio:save-tdr-bnd 270, 275, 745
sdepe:add-substrate 178, 190, 748
sdepe:clean 749
sdepe:define-pe-domain 178, 190, 750
sdepe:depo 178, 193, 752
sdepe:doping-constant-placement 178, 754
sdepe:etch-material 178, 205, 755
sdepe:extend-masks 168, 756
sdepe:fill-device 178, 210, 757
sdepe:generate-domainboundary 758
sdepe:generate-empty-mask 168, 759
sdepe:generate-mask 169, 760
sdepe:icon_layer 178, 211, 762
sdepe:implant 178, 764
sdepe:pattern 178, 190, 766
sdepe:photo 178, 192, 767
sdepe:polish-device 178, 210, 768
sdepe:remove 178, 221, 769
sdepe:trim-masks 169, 770
sdesnmesh:axisaligned 279, 771
sdesnmesh:boundary 282, 283, 774
sdesnmesh:boundary-clear 775
sdesnmesh:delaunizer 280, 776
sdesnmesh:delaunizer-tolerance 280, 778
sdesnmesh:interpolate 779
sdesnmesh:iocontrols 780
sdesnmesh:offsetting 253, 781
sdesnmesh:qualityreport 782
sdesnmesh:tensor 281, 784
sdesnmesh:tools 284, 785
sdesp:begin 786
sdesp:define-step 787
sdesp:finalize 787
sdesp:restore-state 788
sdestr:capitalize 789
sdestr:casetfold 790
sdestr:center 791
sdestr:count 792
sdestr:endswith 793
sdestr:expandtabs 794
sdestr:find 795
sdestr:format 796
sdestr:index 798
sdestr:isalnum 799
sdestr:isalpha 800
sdestr:isdecimal 801
sdestr:isdigit 802
sdestr:islower 803
sdestr:isnumeric 804
sdestr:isspace 805
sdestr:istitle 806
sdestr:isupper 807

Index of Scheme Extensions

sdestr:join 808
sdestr:length 809
sdestr:ljust 810
sdestr:lower 811
sdestr:lstrip 812
sdestr:partition 813
sdestr:replace 814
sdestr:rfind 815
sdestr:rindex 816
sdestr:rjust 817
sdestr:rpartition 818
sdestr:rsplit 819
sdestr:rstrip 820
sdestr:split 821
sdestr:splitlines 822
sdestr:startswith 824
sdestr:swapcase 825
sdestr:upper 826
sdestr:zfill 827
set-interface-contact 828
shell? 332, 829
skin:options 830
solid:area 833
solid:closed? 332
solid:manifold? 332
solid:massprop 834
solid? 332, 833
sort 835
string:head 836
string:tail 837
sweep:law 838

sweep:options 123, 131, 840
system:command 842
system:getenv 842

T

timer:end 843
timer:get-time 843
timer:show-time 844
timer:start 844
transform:reflection 164, 845
transform:rotation 164, 846
transform:scaling 164, 847
transform:translation 161, 164, 848

U

util:make-bot-contact 849
util:make-top-contact 850

V

vertex:position 327
vertex? 332, 851
view:set-point-size 69, 851

W

wire:closed? 332
wire:planar? 332, 853
wire? 332, 852
wire-body? 332, 852