Custom Layout SKILL Functions Reference

Product Version 5.0
July 2002

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Contents

Preface . . . 23
Related Documents . . . 23
Typographic and Syntax Conventions . . . 24
Single Characters Used with Data Types . . . 25

1
Layout Editor Functions . . . 27
Layout Editor SKILL Cross-Reference Table . . . 34
Environment Variable Functions . . . 38
Graphic Environment Variables . . . 38
Layout Environment Variables . . . 38
Environment Variable Table . . . 40
envGetVal . . . 70
envSetVal . . . 71
leAutoRoute . . . 72
leEIPZoomAbsoluteScale . . . 73
leEnvLoad . . . 74
leGetEnv . . . 75
leSetEnv . . . 76
leToggleGravity . . . 77
leToggleMaintainConnections . . . 78
Object Creation and Plotting Functions . . . 79
leCreateAutoInstPin . . . 82
leCreateAutoPin . . . 83
leCreateContact . . . 84
leCreatePath . . . 86
leCreatePin . . . 88
leDefineMPPTemplate . . . 90
Using the leDefineMPPTemplate Function . . . 95
lePlot . . . 98
Object Editing Functions . . . 103

leAttachFig . . . 104
leChopShape . . . 105
leConvertShapeToPolygon . . . 107
leDefineExternalPins . . . 108
leDefineInternalPins . . . 109
leDefinePPPins . . . 110
leDefineWeaklyConnectedPins . . . 111
leFlattenInst . . . 112
leMakeCell . . . 114
leMergeShapes . . . 116
leModifyCorner . . . 117
leMoveCellViewOrigin . . . 119
lePasteFigs . . . 120
leSizeShape . . . 122
leSplitShape . . . 123
leStretchShape . . . 125
leYankFigs . . . 127
Selection Functions . . . 129
leDeiconifyLSW . . . 130
leGetEntryLayer . . . 131
leGetLSWBBox . . . 132
leGetValidLayerList . . . 133
leIconifyLSW . . . 135
leIsFigSelectable . . . 136
leIsInstSelectable . . . 137
leIsLayerSelectable . . . 138
leIsLayerValid . . . 139
leIsLayerVisible . . . 140
leIsLSWIconified . . . 141
leIsPinSelectable . . . 142
leRaiseLSW . . . 143
leRemapLSW . . . 144
leResizeLSW . . . 145
leSetAllLayerSelectable . . . 146
leSetAllLayerValid . . . 147
leSetAllLayerVisible . . . 148

leSetEntryLayer . . . 149
leSetFormSnapMode . . . 150
leSetInstSelectable . . . 151
leSetLayerAttributes . . . 152
leSetLayerSelectable . . . 153
leSetLayerValid . . . 154
leSetLayerVisible . . . 155
leSetLSWBBox . . . 156
leSetPinSelectable . . . 158
leUnmapLSW . . . 159
leZoomToPoint . . . 160
leZoomToSelSet . . . 161
Reference Point Functions . . . 162
Reference Point Setting . . . 162
Moving the Cursor Using the Keyboard . . . 162
Procedural Access . . . 162
leGetRefPoint . . . 163
leIsRefPointActive . . . 164
leMoveCursor . . . 165
leMoveCursorToRefPoint . . . 166
leSetRefPoint . . . 167
leSetRefPointInactive . . . 168
Ruler Functions . . . 169
leClearAllRuler . . . 170
leCreateRuler . . . 171
Search and Replace Functions . . . 172
leReplace . . . 173
leReplaceAnyInstMaster . . . 174
leSearchHierarchy . . . 175
Contact Functions . . . 178
leGetContactDefaultParam . . . 179
leGetContactNameArray . . . 180
leGetContactParam . . . 181
leGetContactRule . . . 182
leIsAnyContact . . . 183
leIsAnyContactMaster . . . 184

leIsContact . . . 185
leIsContactMaster . . . 186
leIsContactName . . . 187
leSetContactParam . . . 188
Hierarchy Traversal Functions . . . 190
leDescend . . . 191
leEditInPlace . . . 192
Binary and Unary Functions . . . 193
leLayerAnd . . . 194
leLayerAndNot . . . 195
leLayerOr . . . 196
leLayerSize . . . 197
leLayerXor . . . 198
Layer Purpose Icon List Function . . . 199
hiMakeLPChoiceList . . . 200
Bindkey Functions . . . 201
cmdCtrlOption . . . 202
cmdOption . . . 203
cmdShiftOption . . . 205
leSelBoxOrStretch . . . 207
Interactive Functions . . . 208
hiLayerDispMainForm . . . 209
leCloseWindow . . . 210
leEditDesignProperties . . . 211
leHiAbout . . . 212
leHiAddShapeToNet . . . 213
leHiAttach . . . 214
leHiChop . . . 215
leHiClearRuler . . . 216
leHiConvertShapeToPolygon . . . 217
leHiCopy . . . 218
leHiCreateBend . . . 219
leHiCreateChoiceOfPin . . . 220
leHiCreateCircle . . . 221
leHiCreateContact . . . 222
leHiCreateDonut . . . 223

leHiCreateEllipse 224
leHiCreateInst 225
leHiCreateLabel 226
leHiCreatePath 227
leHiCreatePin 228
leHiCreatePinsFromLabels 229
leHiCreatePolygon 230
leHiCreateRect 231
leHiCreateRuler 232
leHiCreateSymDev 233
leHiCreateSymPin 234
leHiCreateTaper 235
leHiCreateTrl 236
leHiDelete 237
leHiDeleteAllAreaViewLevel 238
leHiDeleteAreaViewLevel 239
leHiDeleteShapeFromNet 240
leHiDescend 241
leHiEditDisplayOptions 242
leHiEditEditorOptions 243
leHiEditInPlace 244
leHiEditProp 245
leHiFlatten 246
leHiLayerGen 247
leHiLayerTap 248
leHiMakeCell 249
leHiMarkNet 250
leHiMerge 251
leHiModifyCorner 252
leHiMousePopUp 253
leHiMove 254
leHiMoveOrigin 255
leHiOptionLayer 256
leHiPaste 257
leHiPlotQueueStatus 258
leHiPropagateNets 259

leHiReShape . . . 260
leHiRotate . . . 261
leHiSearch . . . 262
leHiSetAreaViewLevel . . . 263
leHiSetRefPoint . . . 264
leHiSetValidLayer . . . 265
leHiShowSelSet . . . 266
leHiSize . . . 267
leHiSplit . . . 268
leHiStretch . . . 269
leHiSubmitPlot . . . 270
leHiSummary . . . 271
leHiTree . . . 272
leHiYank . . . 273
leiDiscardEdits . . . 274

2
Parameterized Cell Functions . . . 275

Safety Rules for Creating SKILL Pcells . . . 278
Calling SKILL Procedures from within Pcells . . . 279
Recommended, Supported SKILL Functions for Pcells . . . 279
Finding Supported SKILL Functions . . . 280
Using print Functions in a Pcell . . . 281
Enclosing the Body of Code in a let or prog for Local Variables . . . 282
What to Avoid When Creating Pcells . . . 282
About Pcell Super and Submaster Cells . . . 283
Using the SKILL Operator ~> with Pcells . . . 284
pc Functions for SKILL Pcell Code . . . 285
pcExprToString . . . 285
pcFix . . . 286
pcRound . . . 288
pcTechFile . . . 289
Parameterized Cell SKILL Cross-Reference Table . . . 290
Graphic Parameterized Cell SKILL Functions . . . 294
pcDefineCondition . . . 295

pcDefineInheritParam 297
pcDefineParamLabel 298
pcDefineParamLayer 299
pcDefineParamPath 301
pcDefineParamPolygon 303
pcDefineParamProp 305
pcDefineParamRect 306
pcDefineParamRefPointObject 307
pcDefinePathRefPointObject 308
pcDefinePCell 309
pcDefineRepeat 312
pcDefineSteppedObject 315
pcDefineStretchLine 317
pcDeleteCondition 319
pcDeleteParamLayer 320
pcDeleteParamProp 321
pcDeleteParamShape 322
pcDeleteRefPoint 323
pcDeleteRepeat 324
pcDeleteSteppedObject 325
pcGetConditions 326
pcGetInheritParamDefn 327
pcGetInheritParams 328
pcGetOffsetPath 329
pcGetOffsetPolygon 331
pcGetParameters 333
pcGetParamLabelDefn 334
pcGetParamLabels 335
pcGetParamLayers 336
pcGetParamLayerDefn 337
pcGetParamProps 338
pcGetParamShapeDefn 339
pcGetParamShapes 340
pcGetRefPointDefn 341
pcGetRefPoints 342
pcGetRepeatDefn 343

pcGetRepeats . . . 344
pcGetSteppedObjectDefn . . . 345
pcGetSteppedObjects . . . 346
pcGetStretchDefn . . . 347
pcGetStretches . . . 348
pcGetStretchSummary . . . 349
pcHICompileToSkill . . . 350
pcHIDefineCondition . . . 351
pcHIDefineInheritedParameter . . . 352
pcHIDefineLabel . . . 353
pcHIDefineLayer . . . 354
pcHIDefineParamCell . . . 355
pcHIDefineParameterizedShape . . . 356
pcHIDefineParamRefPointObject . . . 357
pcHIDefinePathRefPointObject . . . 358
pcHIDefineProp . . . 359
pcHIDefineRepeat . . . 360
pcHIDefineSteppedObject . . . 361
pcHIDefineStretch . . . 362
pcHIDeleteCondition . . . 363
pcHIDeleteLayer . . . 364
pcHIDeleteParameterizedShape . . . 365
pcHIDeleteProp . . . 366
pcHIDeleteRefPointObject . . . 367
pcHIDeleteRepeat . . . 368
pcHIDeleteSteppedObject . . . 369
pcHIDisplayCondition . . . 370
pcHIDisplayInheritedParameter . . . 371
pcHIDisplayLayer . . . 372
pcHIDisplayParameterizedShape . . . 373
pcHIDisplayParams . . . 374
pcHIDisplayProp . . . 375
pcHIDisplayRefPointObject . . . 376
pcHIDisplayRepeat . . . 377
pcHIDisplaySteppedObject . . . 378
pcHIEditParameters . . . 379

pcHIModifyCondition . . . 380
pcHIModifyLabel . . . 381
pcHIModifyLayer . . . 382
pcHIModifyParams . . . 383
pcHIModifyRefPointObject . . . 384
pcHIModifyRepeat . . . 385
pcHIModifySteppedObject . . . 386
pcHIModifyStretchLine . . . 387
pcHIQualifyStretchLine . . . 388
pcHIRedefineStretchLine . . . 389
pcHISummarizeParams . . . 390
pcModifyParam . . . 391
pcRedefineStretchLine . . . 392
pcRestrictStretchToObjects . . . 394
pcSkillGen . . . 395
auHiUltraPCell . . . 397
Pcell Compiler Customization SKILL Functions . . . 399
pcUserAdjustParameters . . . 400
pcUserGenerateArray . . . 401
pcUserGenerateInstance . . . 402
pcUserGenerateInstancesOfMaster . . . 403
pcUserGenerateLPP . . . 405
pcUserGeneratePin . . . 406
pcUserGenerateProperty . . . 407
pcUserGenerateShape . . . 408
pcUserGenerateTerminal . . . 409
pcUserInitRepeat . . . 410
pcUserPostProcessCellView . . . 412
pcUserPostProcessObject . . . 413
pcUserPreProcessCellView . . . 414
pcUserSetTermNetName . . . 415

3
Compactor Functions . . . 417
Introduction . . . 421

syActivateConstraint 423
syCompactView 424
syCreateAlignmentConstraint 426
syCreateFence 428
syCreateJogConstraint 430
syCreateMagneticConstraint 431
syCreatePathWidthConstraint 433
syCreateSeparationConstraint 434
syCreateWireWidthConstraint 437
syDeactivateConstraint 438
syDeleteConstraint 439
syDRC 440
syERC 441
syExtractView 442
syFrameView 444
syGetAlignmentConstraint 446
syGetAutoJog 447
syGetBottomAbutted 448
syGetDimensions 449
syGetEnv 450
syGetFixLowLeftBoundary 451
syGetForceChildCellCompact 452
syGetGeoOptions 453
syGetHardCell 455
syGetInitialDirection 456
syGetLeftAbutted 457
syGetMagneticConstraint 458
syGetMaxAutoJogs 459
syGetMaxIterations 460
syGetPeelbackDistance 461
syGetPFrame 462
syGetPostProcess 463
syGetPreserveCellRow 465
syGetPreserveWireWidth 466
syGetRightAbutted 467
syGetSeparationConstraint 468

syGetTopAbutted . . . 469
syGetUseChildBdy . . . 470
syGetUserConstraint . . . 471
syGetWireJogConstraint . . . 473
syGetWireWidthConstraint . . . 474
syGetXAnchorBoundary . . . 475
syGetXPinGrid . . . 476
syGetXPinGridOffset . . . 477
syGetYAnchorBoundary . . . 478
syGetYPinGrid . . . 479
syGetYPinGridOffset . . . 480
syReadAlignmentConstraint . . . 481
syReadMagneticConstraint . . . 483
syReadSeparationConstraint . . . 485
syReadWireJogConstraint . . . 487
syReadWireWidthConstraint . . . 488
syRefCellConstraint . . . 489
syRefEdgeConstraint . . . 490
sySetAutoJog . . . 492
sySetAutoJogByLayer . . . 493
sySetAutoJogByNet . . . 495
sySetAutoJogByWire . . . 496
sySetBottomAbutted . . . 497
sySetEnv . . . 498
sySetFixLowLeftBoundary . . . 499
sySetForceChildCellCompact . . . 500
sySetGeoOptions . . . 501
sySetHardCell . . . 503
sySetInitialDirection . . . 504
sySetLeftAbutted . . . 505
sySetMaxAutoJogs . . . 506
sySetMaxIterations . . . 507
sySetPeelbackDistance . . . 509
sySetPFrame . . . 510
sySetPostProcess . . . 511
sySetPreserveCellRow . . . 513

sySetPreservePathWidth . . . 514
sySetPreserveWireWidth . . . 515
sySetRightAbutted . . . 516
sySetTopAbutted . . . 517
sySetUseChildBdy . . . 518
sySetXAnchorBoundary . . . 519
sySetXPinGrid . . . 520
sySetXPinGridOffset . . . 521
sySetYAnchorBoundary . . . 522
sySetYFirst . . . 523
sySetYPinGrid . . . 524
sySetYPinGridOffset . . . 525
sySymToGeo . . . 526
sySymToPolygon . . . 527
syUserAllDel . . . 528
syUserAllOff . . . 529
syUserAllOn . . . 530
syUserAllXDel . . . 531
syUserAllXOff . . . 532
syUserAllXOn . . . 533
syUserAllYDel . . . 534
syUserAllYOff . . . 535
syUserAllYOn . . . 536

4
Structure Compiler Functions . . . 537

Introduction . . . 538
scAddCell . . . 539
scAddColumn . . . 541
scAddMaster . . . 543
scAddRow . . . 544
scBias . . . 546
scCellHeight . . . 547
scCellsPerColumn . . . 548
scCellsPerRow . . . 550

scCellWidth . . . 552
scChangeBias . . . 553
scChangeCell . . . 554
scChangeIcon . . . 556
scChangeMaster . . . 558
scChangeOrient . . . 560
scCompileArray . . . 562
scDeleteCell . . . 564
scDeleteColumn . . . 566
scDeletePlane . . . 568
scDeleteRow . . . 569
scDrawPlane . . . 570
scGetPlanePoint . . . 571
scIcon . . . 572
scMaster . . . 574
scOrient . . . 576
scPlane . . . 578
scPromotePin . . . 579
scResizePlane . . . 581
scStretchEmptyCell . . . 582
scSwapCells . . . 584
scSwapColumns . . . 586
scSwapRows . . . 588
scUnusedIcon . . . 590

5
Placement and Routing Translation Functions . . . 591

Introduction . . . 592
Translator User Interface Functions . . . 592
iccDisplayExportForm . . . 593
iccDisplayImportForm . . . 594
iccExportCellview . . . 595
iccImportCellview . . . 598
iccNewRules . . . 600
iccOpenCurrentCellviewRules . . . 601

iccOpenRules . . . 602
iccStartICC . . . 603
Interprocess Communication Functions . . . 604
iccIsConnected . . . 605
iccSendCommand . . . 606
iccSendSkillCommand . . . 607

6
Virtuoso XL Functions . . . 609

Introduction . . . 612
lxCmdOptions . . . 613
lxCmdShiftOptions . . . 614
lxEditPartitioning . . . 615
lxEditPinPlacement . . . 616
lxEditPlacementStyle . . . 617
lxGetLXInfo . . . 618
lxHiAlign . . . 619
lxHiChain . . . 620
lxHiClone . . . 621
lxHiConnectInstPin . . . 622
lxHiCreateInstFromSch . . . 623
lxHiCreateMPP . . . 624
lxHiEditComponentTypes . . . 625
lxHiHideIncNets . . . 626
lxHiLockSelected . . . 627
lxHiMoveAutomatically . . . 628
lxHiPlcDisableCongestionDisplay . . . 629
lxHiPlcEnableCongestionDisplay . . . 630
lxHiProbe . . . 631
lxHiReInitDesign . . . 632
lxHiSetCorrespondence . . . 633
lxHiSetOptions . . . 634
lxHiShowIncNets . . . 635
lxHiStack . . . 636
lxHiSwapComps . . . 637

lxHiUnlockSelected 638
lxHiUpdateCellViewPair 639
lxHiUpdateComponentsAndNets 640
lxHiUpdateDeviceCorr 641
lxHiUpdateLayoutParameters 642
lxHiUpdateSchematicParameters 643
lxHiUpdateSwitchViews 644
lxHiVerifyDesign 645
lxHiVerifyStatus 646
lxPermPermutePins 647
lxPlcAppendPlaceSetupFieldValue 648
lxPlcGetPlaceSetupFieldValue 650
lxPlcIsPlaceSetupFieldEnabled 651
lxPlcReplacePlaceSetupField 652
lxPlcSetPlaceSetupFieldEnableState 654
lxPlcSetPlaceSetupFieldValue 656
lxProbeRemoveAll 658
lxToggleIncNets 659
lxTpSaveTemplate 660
Wire Editing Functions 661
leWeAddVia 662
leWeCheckRoutes 663
leWeDisableAsManyAsFitMenuItem 664
leWeDisableCopyRouteMirrorMenuItem 665
leWeDisableViaPatternMenuItem 666
leWeFinishRoute 667
leWeHiCheckRoutes 668
leWeHiCopyRoute 669
leWeHiCriticWire 670
leWeHiPull 671
leWeHiRouteOptions 672
leWePathWidthMenuCB 673
leWePickUpDroppedWires 674
leWeRefresh 675
leWeReports 676
leWeSetGatherBusWiresMenuItemText 677

leWeSetInteractiveCheckingText 678
leWeToggle 679

7
Virtuoso Constraint Manager Functions 681
Common Generator Functions 688
cmxfGenIsId 688
cmxfStopGen 689
Category Functions 690
cmxfCatAttrGetDefVal 690
cmxfCatAttrGetType 691
cmxfCatGetId 692
cmxfCatGetName 693
cmxfCatGetNumAttrs 694
cmxfCatGetWeight 695
cmxfCatIsAttr 696
cmxfCatIsId 697
cmxfGenCatToCon 698
cmxfGenSuperCat 699
cmxfGenSuperCatToCat 700
cmxfSCatGetId 701
cmxfSCatGetName 702
cmxfSCatIsId 703
cmxfStartGenCatToCon 704
cmxfStartGenSuperCat 705
cmxfStartGenSuperCatToCat 706
Net Functions 707
cmxfGenNetToCon 707
cmxfNetIsConstrained 708
cmxfStartGenNetToCon 709
Fig Functions 710
cmxfFigIsConstrained 710
cmxfGenFigToCon 711
cmxfShapeNameGen 712
cmxfStartGenFigToCon 713

Axis Functions 714
cmxfAxisCreate 714
cmxfAxisDelete 716
cmxfAxisGetCV 717
cmxfAxisGetDirection 718
cmxfAxisGetId 719
cmxfAxisGetName 720
cmxfAxisGetX 721
cmxfAxisGetY 722
cmxfAxisIsActive 723
cmxfAxisIsConstrained 724
cmxfAxisIsHidden 725
cmxfAxisIsId 726
cmxfAxisIsInherited 727
cmxfAxisIsReadOnly 728
cmxfAxisIsValid 729
cmxfAxisModify 730
cmxfAxisNameGen 731
cmxfAxisSetActive 732
cmxfAxisSetDirection 733
cmxfAxisSetX 734
cmxfAxisSetY 735
cmxfGenAxis 736
cmxfGenAxisToCon 737
cmxfStartGenAxis 738
cmxfStartGenAxisToCon 739
Cluster Functions 740
cmxfClstrChngMems 740
cmxfClstrCreate 742
cmxfClstrDelete 743
cmxfClstrGetBBox 744
cmxfClstrGetCenter 745
cmxfClstrGetCV 746
cmxfClstrGetId 747
cmxfClstrGetName 748
cmxfClstrGetSize 749

cmxfClstrHasMems . . . 750
cmxfClstrIsActive . . . 751
cmxfClstrIsConstrained . . . 752
cmxfClstrIsHidden . . . 753
cmxfClstrIsId . . . 754
cmxfClstrIsInherited . . . 755
cmxfClstrIsMember . . . 756
cmxfClstrIsReadOnly . . . 757
cmxfClstrIsValid . . . 758
cmxfClstrNameGen . . . 759
cmxfClstrSetActive . . . 760
cmxfGenClstr . . . 761
cmxfGenClstrToCon . . . 762
cmxfGenClstrToFig . . . 763
cmxfGenClstrToMemName . . . 764
cmxfStartGenClstr . . . 765
cmxfStartGenClstrToCon . . . 766
cmxfStartGenClstrToFig . . . 767
cmxfStartGenClstrToMemName . . . 768
Net-Class Functions . . . 769
cmxfGenNC . . . 769
cmxfGenNCToCon . . . 770
cmxfGenNCToMemName . . . 771
cmxfGenNCToNet . . . 772
cmxfNCChngMems . . . 773
cmxfNCCreate . . . 774
cmxfNCDelete . . . 775
cmxfNCGetCV . . . 776
cmxfNCGetId . . . 777
cmxfNCGetName . . . 778
cmxfNCGetSize . . . 779
cmxfNCHasMem . . . 780
cmxfNCIsActive . . . 781
cmxfNCIsConstrained . . . 782
cmxfNCIsHidden . . . 783
cmxfNCIsId . . . 784

cmxfNCIsInherited . . . 785
cmxfNCIsMember . . . 786
cmxfNCIsReadOnly . . . 787
cmxfNCIsValid . . . 788
cmxfNCNameGen . . . 789
cmxfNCSetActive . . . 790
cmxfStartGenNC . . . 791
cmxfStartGenNCToCon . . . 792
cmxfStartGenNCToMemName . . . 793
cmxfStartGenNCToNet . . . 794
Constraint Functions . . . 795
cmxfConChngMems . . . 795
cmxfConCreate . . . 797
cmxfConDelete . . . 799
cmxfConGetAttrVal . . . 800
cmxfConGetCatId . . . 801
cmxfConGetCV . . . 802
cmxfConGetId . . . 803
cmxfConGetName . . . 804
cmxfConGetRefFig . . . 805
cmxfConGetWeight . . . 806
cmxfConHasMems . . . 807
cmxfConIsActive . . . 808
cmxfConIsHidden . . . 809
cmxfConIsId . . . 810
cmxfConIsInherited . . . 811
cmxfConIsOverCon . . . 812
cmxfConIsReadOnly . . . 813
cmxfConIsSatisfied . . . 814
cmxfConIsValid . . . 815
cmxfConModify . . . 816
cmxfConNameGen . . . 817
cmxfConSetActive . . . 818
cmxfConSetAttrVal . . . 819
cmxfConSetWeight . . . 820
cmxfConVerify . . . 821

cmxfGenCon 822
cmxfGenConToFig 823
cmxfGenConToMemName 824
cmxfGenConToNet 825
cmxfStartGenCon 826
cmxfStartGenConToFig 827
cmxfStartGenConToMemName 828
cmxfStartGenConToNet 829
Report and GUI Functions 830
cmxfAsc 830
cmxfGuiReportGen 832
cmxfGuiStartUp 833

Preface

This manual describes custom layout SKILL functions for layout editor, parameterized cells, compactor, structure compiler, placement and routing translation, Virtuoso XL, and Virtuoso constraint manager.

This manual assumes that you understand the SKILL programming language.

This preface discusses the following topics:

- Related Documents on page 23
- Typographic and Syntax Conventions on page 24
- Single Characters Used with Data Types on page 25

Related Documents

The Cadence® SKILL functions defined in this manual are often used with other Cadence products.

The following manuals give you more information about SKILL functions.

- To learn about using Relative Object Design (ROD) functions to create pcells, refer to *Relative Object Design User Guide*.
- To learn SKILL language basics, refer to the *SKILL Language Reference* and the *SKILL Language User Guide*.
- To learn about online help interface or the graphics editor, refer to the *Design Framework II SKILL Reference Manual*.
- To learn about the SKILL Development toolbox, refer to *SKILL Development Help*.

This manual contains a chapter each about SKILL functions for the following applications:

- Virtuoso® layout editor

  For more information about Virtuoso layout editor, read the *Virtuoso Layout Editor User Guide*.
- Virtuoso parameterized cells

For more information about Virtuoso parameterized cells, see the *Virtuoso Parameterized Cell Reference*.

- Virtuoso compactor

  For more information about the Virtuoso compactor, see the *Virtuoso Compactor Reference Manual*.

- Structure compiler

  For more information about the structure compiler, see the *Structure Compiler Reference*.

- Virtuoso placement and routing translators

  For more information translation to the Virtuoso custom placer, the Virtuoso custom router, and the Cadence chip assembly router, see the *Virtuoso Custom Placement and Routing Preparation Guide*.

- Virtuoso layout accelerator (Virtuoso XL)

  For more information about Virtuoso XL, read the *Virtuoso XL Layout Editor User Guide*.

- Virtuoso constraint manager

  For more information about Virtuoso XL, read the *Virtuoso Constraint Manager User Guide*.

Typographic and Syntax Conventions

This section describes typographic and syntax conventions used in this manual.

`text`	Indicates text you must type exactly as it is presented.
`z_argument`	Indicates text that you must replace with an appropriate argument. The prefix (in this case, *`z_`*) indicates the data type the argument can accept. Do not type the data type or underscore.
`[ ]`	Denotes optional arguments. When used with vertical bars, they enclose a list of choices from which you can choose one.
`{ }`	Used with vertical bars and encloses a list of choices from which you must choose one.

|	Separates a choice of options.
…	Indicates that you can repeat the previous argument.
=>	Precedes the values returned by a Cadence® SKILL language function.
/	Separates the possible values that can be returned by a Cadence SKILL language function.
text	Indicates names of manuals, menu commands, form buttons, and form fields.

Important

The language requires many characters not included in the preceding list. You must type these characters exactly as they are shown in the syntax.

Single Characters Used with Data Types

SKILL supports several data types, including integer and floating-point numbers, character strings, arrays, and a highly flexible linked list structure for representing aggregates of data.

For symbolic computation, SKILL has data types for dealing with symbols and functions that represent the type of value you can assign to the field.
.

Prefix	Internal Name	Data Type
a	array	array
b	ddUserType	Boolean
c	opfcontext	OPF context
d	dbobject	Cadence database object (CDBA)
e	envobj	environment
f	flonum	floating-point number
F	opffile	OPF file ID
g	general	any data type
G	gdmSpecIlUserType	gdm spec

Prefix	Internal Name	Data Type
h	hdbobject	hierarchical database configuration object
l	list	linked list
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	—
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list
r	defstruct	defstruct
R	rodObj	ROD object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	—
w	wtype	window type
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

1

Layout Editor Functions

This section provides syntax, descriptions, and examples for the Cadence® SKILL functions associated with the Virtuoso® layout editor menu commands.

Layout Editor SKILL Cross-Reference Table on page 34

Environment Variable Functions on page 38

- Graphic Environment Variables on page 38
- Layout Environment Variables on page 38
- Environment Variable Table on page 40
- envGetVal on page 70
- envSetVal on page 71
- leAutoRoute on page 72
- leEIPZoomAbsoluteScale on page 73
- leEnvLoad on page 74
- leGetEnv on page 75
- leSetEnv on page 76
- leToggleGravity on page 77
- leToggleMaintainConnections on page 78

Object Creation and Plotting Functions on page 79

- leCreateAutoInstPin on page 82
- leCreateAutoPin on page 83
- leCreateContact on page 84
- leCreatePath on page 86

leCreatePin on page 88

leDefineMPPTemplate on page 90

lePlot on page 98

Object Editing Functions on page 103

leAttachFig on page 104

leChopShape on page 105

leConvertShapeToPolygon on page 107

leDefineExternalPins on page 108

leDefineInternalPins on page 109

leDefinePPPins on page 110

leDefineWeaklyConnectedPins on page 111

leFlattenInst on page 112

leMakeCell on page 114

leMergeShapes on page 116

leModifyCorner on page 117

leMoveCellViewOrigin on page 119

lePasteFigs on page 120

leSizeShape on page 122

leSplitShape on page 123

leStretchShape on page 125

leYankFigs on page 127

Selection Functions on page 129

leDeiconifyLSW on page 130

leGetEntryLayer on page 131

leGetLSWBBox on page 132

leGetValidLayerList on page 133

leIconifyLSW on page 135

leIsFigSelectable on page 136

leIsInstSelectable on page 137

leIsLayerSelectable on page 138

leIsLayerValid on page 139

leIsLayerVisible on page 140

leIsLSWIconified on page 141

leIsPinSelectable on page 142

leRaiseLSW on page 143

leRemapLSW on page 144

leResizeLSW on page 145

leSetAllLayerSelectable on page 146

leSetAllLayerValid on page 147

leSetAllLayerVisible on page 148

leSetEntryLayer on page 149

leSetFormSnapMode on page 150

leSetInstSelectable on page 151

leSetLayerAttributes on page 152

leSetLayerSelectable on page 153

leSetLayerValid on page 154

leSetLayerVisible on page 155

leSetLSWBBox on page 156

leSetPinSelectable on page 158

leUnmapLSW on page 159

leZoomToPoint on page 160

leZoomToSelSet on page 161

Reference Point Functions on page 162

- Reference Point Setting on page 162
- Moving the Cursor Using the Keyboard on page 162
- Procedural Access on page 162
- leGetRefPoint on page 163
- leIsRefPointActive on page 164
- leMoveCursor on page 165
- leMoveCursorToRefPoint on page 166
- leSetRefPoint on page 167
- leSetRefPointInactive on page 168

Ruler Functions on page 169

- leClearAllRuler on page 170
- leCreateRuler on page 171

Search and Replace Functions on page 172

- leReplace on page 173
- leReplaceAnyInstMaster on page 174
- leSearchHierarchy on page 175

Contact Functions on page 178

- leGetContactDefaultParam on page 179
- leGetContactNameArray on page 180
- leGetContactParam on page 181
- leGetContactRule on page 182
- leIsAnyContact on page 183
- leIsAnyContactMaster on page 184
- leIsContact on page 185
- leIsContactMaster on page 186

leIsContactName on page 187

leSetContactParam on page 188

Hierarchy Traversal Functions on page 190

leDescend on page 191

leEditInPlace on page 192

Binary and Unary Functions on page 193

leLayerAnd on page 194

leLayerAndNot on page 195

leLayerOr on page 196

leLayerSize on page 197

leLayerXor on page 198

Layer Purpose Icon List Function on page 199

hiMakeLPChoiceList on page 200

Bindkey Functions on page 201

cmdCtrlOption on page 202

cmdOption on page 203

cmdShiftOption on page 205

leSelBoxOrStretch on page 207

Interactive Functions on page 208

hiLayerDispMainForm on page 209

leCloseWindow on page 210

leEditDesignProperties on page 211

leHiAbout on page 212

leHiAddShapeToNet on page 213

leHiAttach on page 214

leHiChop on page 215

leHiClearRuler on page 216

leHiConvertShapeToPolygon on page 217

leHiCopy on page 218

leHiCreateBend on page 219

leHiCreateChoiceOfPin on page 220

leHiCreateCircle on page 221

leHiCreateContact on page 222

leHiCreateDonut on page 223

leHiCreateEllipse on page 224

leHiCreateInst on page 225

leHiCreateLabel on page 226

leHiCreatePath on page 227

leHiCreatePin on page 228

leHiCreatePinsFromLabels on page 229

leHiCreatePolygon on page 230

leHiCreateRect on page 231

leHiCreateRuler on page 232

leHiCreateSymDev on page 233

leHiCreateSymPin on page 234

leHiCreateTaper on page 235

leHiCreateTrl on page 236

leHiDelete on page 237

leHiDeleteAllAreaViewLevel on page 238

leHiDeleteAreaViewLevel on page 239

leHiDeleteShapeFromNet on page 240

leHiDescend on page 241

leHiEditDisplayOptions on page 242

leHiEditEditorOptions on page 243

leHiEditInPlace on page 244

leHiEditProp on page 245

leHiFlatten on page 246

leHiLayerGen on page 247

leHiLayerTap on page 248

leHiMakeCell on page 249

leHiMarkNet on page 250

leHiMerge on page 251

leHiModifyCorner on page 252

leHiMousePopUp on page 253

leHiMove on page 254

leHiMoveOrigin on page 255

leHiOptionLayer on page 256

leHiPaste on page 257

leHiPlotQueueStatus on page 258

leHiPropagateNets on page 259

leHiReShape on page 260

leHiRotate on page 261

leHiSearch on page 262

leHiSetAreaViewLevel on page 263

leHiSetRefPoint on page 264

leHiSetValidLayer on page 265

leHiShowSelSet on page 266

leHiSize on page 267

leHiSplit on page 268

leHiStretch on page 269

leHiSubmitPlot on page 270

leHiSummary on page 271

leHiTree on page 272

leHiYank on page 273

leiDiscardEdits on page 274

Layout Editor SKILL Cross-Reference Table

This table summarizes the procedural and interactive SKILL functions associated with the Virtuoso layout editor menu commands. A complete listing of the syntax, descriptions, and examples for the layout editor SKILL functions follows this table.

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leCloseWindow	hiCloseWindow	*Window – Close*
leEditDesignProperties	—	*Design – Properties*
leHiAbout	—	*Help – About layout*
leHiAddShapeToNet	—	*Connectivity – Add Shape to Net*
leHiAttach	leAttachFig	*Edit – Other – Attach/Detach*
leHiChop	leChopShape	*Edit – Other – Chop*
leHiClearRuler	leClearAllRuler	*Window – Clear All Rulers*
leHiConvertShapeToPolygon	leConvertShapeToPolygon	*Edit – Other – Convert to Polygon*
leHiCopy	dbCopyFig	*Edit – Copy*
leHiCreateBend	geCreateBend	*(Microwave) Create – Bend*
leHiCreateChoiceOfPin	leCreatePin	*Create – Pin*
leHiCreateCircle	dbCreateEllipse	*Create – Conics – Circle*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiCreateContact	leCreateContact	*Create – Contact*
leHiCreateDonut	dbCreateDonut	*Create – Conics – Donut*
leHiCreateEllipse	dbCreateEllipse	*Create – Conics – Ellipse*
leHiCreateInst	dbCreateSimpleMosaic dbCreateInst	*Create – Instance*
leHiCreateLabel	dbCreateLabel	*Create – Label*
leHiCreatePath	leCreatePath	*Create – Path*
leHiCreatePin	leCreatePin	*Create – Pin*
leHiCreatePinsFromLabels	—	*Create – Pins From Labels*
leHiCreatePolygon	dbCreatePolygon	*Create – Polygon*
leHiCreateRect	dbCreateRect	*Create – Rectangle*
leHiCreateRuler	leCreateRuler	*Window – Create Ruler*
leHiCreateSymDev	dbCreateInst	*Create – Device*
leHiCreateSymPin	dbCreatePin	*Create – Pin*
leHiCreateTaper	geCreateTaper	*(Microwave) Create – Taper*
leHiCreateTrl	geCreateTrl	*(Microwave) Create – Trl*
leHiDelete	dbDeleteObject	*Edit – Delete*
leHiDeleteAllAreaViewLevel	geDeleteAllAreaViewLevel	*Window – Area Display – Delete All*
leHiDeleteAreaViewLevel	geDeleteAreaViewLevel	*Window – Area Display – Delete*
leHiDeleteShapeFromNet	—	*Connectivity – Delete Shape From Net*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiDescend	leDescend	*Design – Hierarchy – Descend*
leHiEditDisplayOptions	envGetVal envSetVal	*Options – Display*
leHiEditEditorOptions	envGetVal envSetVal	*Options – Layout Editor*
leHiEditInPlace	leEditInPlace	*Design – Hierarchy – Edit In Place*
leHiEditProp	—	*Edit – Properties*
leHiFlatten	leFlattenInst	*Edit – Hierarchy – Flatten*
leHiLayerGen	leLayerAnd leLayerAndNot leLayerOr leLayerSize leLayerXor	*Create – Layer Generation*
leHiLayerTap	—	*LSW – Layer Tap*
leHiMakeCell	leMakeCell	*Edit – Hierarchy – Make Cell*
leHiMarkNet	—	*Connectivity – Mark Net*
leHiMerge	leMergeShapes	*Edit – Merge*
leHiModifyCorner	leModifyCorner	*Edit – Other – Modify Corner*
leHiMove	dbMoveFig	*Edit – Move*
leHiMoveOrigin	leMoveCellViewOrigin	*Edit – Other – Move Origin*
leHiPaste	lePasteFigs	*Edit – Other – Paste*
leHiPlotQueueStatus	--	*Design – Plot – Queue Status*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiPropagateNets	--	*Connectivity – Propagate Nets*
leHiReShape	--	*Edit – Reshape*
leHiRotate	--	*Edit – Other – Rotate*
leHiSearch	leSearchHierarchy leReplace	*Edit – Search*
leHiSetAreaViewLevel	geSetAreaViewLevel	*Window – Area Display – Set*
leHiSetRefPoint	leSetRefPoint	
leHiSetValidLayer	leSetLayerValid leIsLayerValid leSetAllLayerSelectable leIsLayerSelectable leSetAllLayerValid leSetAllLayerVisible leIsLayerVisible leSetEntryLayer leSetLayerSelectable leSetLayerVisible	*LSW: Edit – Set Valid Layers*
leHiShowSelSet	--	*Window – Show Selected Set*
leHiSize	leSizeShape	*Edit – Other – Size*
leHiSplit	leSplitShape	*Edit – Other – Split*
leHiStretch	leStretchShape	*Edit – Stretch*
leHiSubmitPlot	lePlot	*Design – Plot – Submit*
leHiSummary	--	*Design – Summary*
leHiTree	--	*Design – Hierarchy – Tree*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiYank	leYankFigs	*Edit – Other – Yank*
leiDiscardEdits	—	*Design – Discard Edits*

Environment Variable Functions

You can set two types of environment variables with the Virtuoso layout editor: graphic and layout.

Graphic Environment Variables

Graphic environment variables control the characteristics of the window display. These variables can be stored in the cellview or the `.cdsenv` file in your home directory. There are several ways to change the default settings of graphic environment variables:

- Set the variable in the layout editor's Options – Display form. If you want to retain these settings for future sessions, save the settings in the CIW – Options – Save Defaults form.
- Copy the variables you want to change from the default `.cdsenv` file to the `.cdsenv` file in your home directory. It is there that you edit the default value. The default file is located at:

  `<install_dir>/tools/dfII/samples/.cdsenv`
- Set the variable in the Command Interpreter Window using

  `envSetVal("graphic" <"name"> <type> <value>)`

  For example:

  ```
  envSetVal("graphic" "iconsOn" 'boolean t)
  ```

Layout Environment Variables

Layout environment variables control how various layout editor commands work. These variables can be stored in the cellview or the `.cdsenv` file in your home directory. There are several ways to change the default settings of layout environment variables:

- Set the variable in the Options – Layout Editor form. If you want to retain these settings for future sessions, save the settings in the CIW – Options – Save Defaults form.

- Copy the variables you want to change from the default `.cdsenv` file to the `.cdsenv` file in your home directory. It is there that you edit the default value. The default file is located at:

  `<install_dir>/tools/dfII/samples/.cdsenv`

- Set the variable in the CIW using

  `envSetVal("layout" <"name"> <type> <value>)`

  For example:

  ```
  envSetVal("layout" "iconsOn" 'boolean t)
  ```

Environment Variable Table

This table contains the Virtuoso layout editor and graphics editor environment variable names, descriptions, types, and values. Many graphic environmental variables have duplicate layout environment variables. In these cases, the layout variable supersedes the graphic variable unless the graphic variable is stored in the cellview.

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`accessEdgesOn`	Determines whether you see the access edges of pins.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	Default Value: `nil` Valid Values: `t` or `nil`
`applyBounce`	Specifies if gravity offset (bounce) values are used.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`arrayDisplay`	Determines how arrays are displayed.	`String`	Default Value: `Full` Valid Values: `Full`, `Border`, `Source`	Default Value: `Full` Valid Values: `Full`, `Border`, `Source`
`askSaveOnReturn`	Determines whether a dialog box appears after an edit-in-place session, while in edit mode, and when changes have been made. The dialog box asks if you want to save your changes to a cellview. If set to `nil`, the dialog box does not appear and the cellview stays open in edit mode.	`Boolean`		Default Value: `t` Valid Values: `t` or `nil`
`autoContact`	Specifies if auto contacts are used. If set to `t`, then `contactWidth`, `contactLength`, `contactJustify`, `contactRows`, and `contactColumns` do not apply.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`autoInstPin`	Determines whether to create a manual or auto symbolic pin.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`autoNetName`	Determines whether shapes are added to nets attached to pins.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`autoPermutePins`	Determines whether pins that are defined as permutable can exchange places automatically during automatic routing.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`autoRefresh`	Specifies whether automatic checking of cellview needing refresh on edit-in-place or descend is on.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`autoSetRefPoint`	Determines whether reference points are set automatically whenever a new point is entered.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`chopShape`	Specifies the default shape of the cutter used by `leHiChop` or `leChopShape`.	`Cyclic`	Default Value: `rectangle` Valid Values: `rectangle`, `polygon`, or `line`	
`contactColumns`	Specifies the number of columns for a contact placement.	`Integer`	Default Value: `1` Valid Values: any integer greater than zero	
`contactDefFromTechfile`	Sets the default location for the Create Contact form's *DeltaX*, *DeltaY*, *Rows*, and *Columns* fields when you start the layout editor. When `t`, the location is set to the technology file. When `nil`, the location is set to the `.cdsenv` file.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`contactDelX`	Specifies the distance between the center points of contacts in the rows of a contact array.	`Floating`	Default Value: `0.0` user units Valid Values: any floating-point number greater than zero	
`contactDelY`	Specifies the distance between the center points of contacts in the columns of a contact array.	`Floating`	Default Value: `0.0` user units Valid Values: any floating-point number greater than zero	
`contactJustify`	Specifies the justification of the contact with respect to its origin.	`Cyclic`	Default Value: `centerCenter` Valid Values: `lowerLeft`, `centerLeft`, `upperLeft`, `lowerCenter`, `centerCenter`, `upperCenter`, `lowerRight`, `centerRight`, or `upperRight`	
`contactLength`	Specifies the default length of a contact, or the length of each contact-cut in a contact array.	`Floating`	Default Value: `0.8` user units Valid Values: any floating-point number greater than zero	
`contactName`	Default type of contact to be placed using `leHiCreateContact` or `leCreateContact`, such as `via`, `ptap`, `ntap`, and so on. You assign contact names using `symContactDevice`.	`String`	Default Value: none Valid Values: any contact type	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`contactResetDelXY`	Sets the default location for the Create Contact form's *DeltaX* and *DeltaY* fields each time you create a new contact or select a new contact type. When `t`, the location is set to the technology file. When `nil`, the last value entered in the fields is retained.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`contactResetRowCol`	Sets the default location for the Create Contact form's *Rows* and *Columns* fields each time you create a new contact or select a new contact type. When `t`, the location is set to the technology file. When `nil`, the last value entered in the fields is retained.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`contactRows`	Specifies the number of rows for a contact placement.	`Integer`	Default Value: `1` Valid Values: any integer greater than zero	
`contactWidth`	Specifies the default width of a contact or the width of each contact-cut in a contact array.	`Floating`	Default Value: `0.8` user units Valid Values: any floating-point number greater than zero	
`copyColumns`	Specifies the default number of columns to copy.	`Integer`	Default Value: `1` Valid Values: any integer greater than zero	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`copyResetRowCol`	Sets the default location for the Copy form's *Rows* and *Columns* fields each time you repeat the Copy command. When `t`, the location is set to the `.cdsenv` file. When `nil`, the last value entered in the fields is retained.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`copyRows`	Specifies the default number of rows to copy.	`Integer`	Default Value: `1` Valid Values: any integer greater than zero	
`copyToLayer`	Specifies whether shapes are copied to a different layer with `leHiCopy`.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`createPinLabel`	Specifies that a pin label be created when using the Create Pin form.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`
`defaultNewCellName`	Specifies the default name of the cell in the Create New File form.	`String`		Default Value: none Valid Values: any cell name
`defaultNewLibName`	Specifies the default name of the library in the Create New File form.	`String`		Default Value: none Valid Values: any library name
`defaultNewViewName`	Specifies the default name of the view in the Create New File form.	`String`		Default Value: none Valid Values: any valid view name
`defaultOpenCellName`	Specifies the default name of the cell in the Open File form.	`String`		Default Value: none Valid Values: any cell name
`defaultOpenLibName`	Specifies the default name of the library in the Open File form.	`String`		Default Value: none Valid Values: any library name

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`defaultOpenViewName`	Specifies the default name of the view in the Open File form.	`String`		Default Value: none Valid Values: any valid view name
`defaultToolName`	Specifies the default name of the tool in the Create New File form.	`String`		Default Value: none Valid Values: any valid tool name
`descendPromptForView`	Determines whether to open the Descend (view name) form when you click on an instance. In the Descend (view name) form, you can choose a different view to open. If there is only one view of the master cell, Descend does not prompt you for a view name. If set to `nil`, the default view is the view name of the selected object.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`devClass`	Specifies the default device class that is defined in the technology file.	`String`	Default Values: none Valid Values: any device class	
`devType`	Specifies the default device type that is defined in the technology file.	`String`	Default Values: none Valid Values: any device type	
`displayPinNames`	Determines whether to display pin name labels in the cellview.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`
`displayRefPoint`	Specifies if the reference point is displayed in the cellview.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
displayStartLevel	Specifies the start level for viewing the hierarchy in the Set Area View Level form.	Integer	Default Value: 0 Valid Values: any integer between 0 and 32	
displayStopLevel	Specifies the stop level for viewing the hierarchy in the Set Area View Level form.	Integer	Default Value: 0 Valid Values: any integer between 0 and 32	
displayStretchHandles	Specifies whether stretch handles are displayed for stretchable pcells.	Boolean	Default Value: t Valid Values: t or nil.	Default Value: t Valid Values: t or nil
dotsOn	Determines whether to display dot pins in the cellview.	Boolean	Default Value: t Valid Values: t or nil	Default Value: t Valid Values: t or nil
drawAxesOn	Determines whether you see the cellview axes.	Boolean	Default Value: t Valid Values: t or nil	Default Value: t Valid Values: t or nil
drawDottedGridOn	Determines whether the grid is displayed as dots (t) or lines (nil), when drawGridOn is t.	Boolean	Default Value: t Valid Values: t or nil	Default Value: t Valid Values: t or nil
drawEngine	Determines whether or not all data in a cellview is rendered. If 0, all data is drawn including that which will be obscured by data on overlying layers. If 1, only data which will ultimately be viewable is drawn.	Integer		Default Value: 0 Valid Values: 0 or 1
drawGridOn	Determines whether the grid is displayed.	Boolean	Default Value: t Valid Values: t or nil	Default Value: t Valid Values: t or nil
drawInstancePins	Determines whether nonmaskable pins are displayed.	Boolean	Default Value: nil Valid Values: t or nil	Default Value: t Valid Values: t or nil

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`drawPRBoundary`	Determines whether the place and route boundary of instances is displayed (`t`) or the instance bBox is displayed (`nil`)	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	Default Value: `nil` Valid Values: `t` or `nil`
`drawSurroundingOn`	Determines whether the surrounding design is displayed when you edit in place.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`
`drfPath`	Specifies the path to the Display Resource File.	`String`		Default Value: none Valid Values: valid path to Display Resource File.
`dynamicHilightOn`	Determines whether dynamic highlighting is on.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`
`extractEnabled`	Enables the extractor to be active when Virtuoso XL is initiated.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`extractStopLevel`	Controls the level at which hierarchical extraction stops.	`Integer`	Default Value: `0` Valid Values: an integer from `0` to `32`	
`filterSize`	Specifies how much of the design displays. A larger filter size allows fewer objects to display, and the screen redraws faster. The `F9` key also controls the value of `filterSize`. `F9` toggles `filterSize` to `0.0`, `3.0`, `10.0`, `25.0`, or `50.0`.	`Floating`	Default Value: `3.0` user units Valid Values: any positive floating-point number	Default Value: `3.0` Valid Values: any positive floating-point number

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`filterSizeDrawingStyle`	Specifies how objects smaller than the filter size display.	`String`	Default Value: `outlined` Valid Values: `empty` or `filled`	Default Value: `filled` Valid Values: `empty` or `outlined`
`fixedWidthPaths`	Specifies whether to override the technology file and use a fixed width by default when creating a path.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`flattenMode`	Specifies the default number of levels of hierarchy of a cell to flatten in `leHiFlatten()` or `leFlattenInst()`.	`Cyclic`	Default Value: one level Valid Values: one level or displayed levels	
`flattenPCells`	Specifies whether pcells are flattened by default by `leHiFlatten()` or `leFlattenInst()`.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`gravityAperture`	Specifies the distance the cursor must be from an object before gravity snaps the cursor to the object.	`Floating`	Default Value: `0.3` user units Valid Values: any positive number	
`gravityBounceX`	Determines the gravity X offset distance. A positive number offsets the cursor away from the target object, a negative number offsets the cursor inside the target object.	`Floating`	Default Value: `0.0` user units Valid Values: any positive or negative floating-point number	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`gravityBounceY`	Determines the gravity Y offset distance. A positive number offsets the cursor away from the target object, a negative number offsets the cursor inside the target object.	`Floating`	Default Value: `0.0` Valid Values: any positive or negative floating-point number	
`gravityDepth`	Specifies the depth in the hierarchy to which gravity snaps. This allows gravity to snap to objects inside nested cells.	`Integer`	Default Value: `0` Valid Values: any positive integer	
`gravityOn`	Specifies whether gravity mode is on. Gravity mode causes the cursor to snap to objects in the cellview. You must also set gravityType for gravityOn to take effect. (see gravityType)	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`gravityType`	Specifies the type of objects to which gravity snaps.	`String`	Valid Values: any combination of the default values or none. Default Value: `list ("centerline" "edge" "pin" "vertex" "end" "midpoint" "nexus" "junction")`	
`gridMultiple`	Default number of minor grid lines or dots between major grid lines or dots.	`Integer`	Default Value: `5` Valid Values: any positive floating-point number	Default Value: `5` Valid Values: any positive floating-point number
`gridSpacing`	Default number between each minor grid line or dot.	`Floating`	Default Value: `1.0` user units Valid Values: any positive floating-point number	Default Value: `1.0` user units Valid Values: any positive floating-point number

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
iconsOn	Determines whether to display the elements in an unexpanded array as rectangles with the symbol for the master cell for each element.	Boolean	Default Value: nil Valid Values: t or nil	Default Value: t Valid Values: t or nil
ignoreBounce	Determines whether you use the gravity offset (bounce) values from the Layout Editor Options form (nil) or the minSpacing value from the design rules in the technology file (t).	Boolean	Default Value: nil Valid Values: t or nil	
instCellName	Specifies the default name of the cell for leHiCreateInst.	String	Default Values: none Valid Values: any cell name	
instColumns	Specifies the number of columns for an arrayed instance.	Integer	Default Value: 1 Valid Values: any integer greater than zero	
instDelX	Specifies the distance between the columns of an arrayed instance or contact measured from edge to edge. This field is grayed out if the number of columns specified is 1.	Floating	Default Value: 0.0 user units Valid Values: any number greater than zero	
instDelY	Specifies the distance between the rows of an arrayed instance or contact measured from edge to edge. This field is grayed out if the number of rows specified is 1.	Floating	Default Value: 0.0 user units Valid Values: any number greater than zero	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`instLabel`	Determines whether the instance name or master name of an instance is displayed.	`String`	Default Value: `master` Valid Values: `instance` or `master`	Default Value: `master` Valid Values: `instance` or `master`
`instLibName`	Specifies the default name of the library for `leHiCreateInst`.	`String`	Default Value: none Valid Values: any library name	
`instName`	Specifies one or more names for the instances you create. You can specify as many names as you want (separated by spaces). As each name is used, it is removed from the list. If all names are used, the function automatically generates a name based on the current master's `instNamePrefix` property and puts it in the form field.	`String`	Default Value: none Valid Values: any cell name	
`instRows`	Specifies the number of rows for an arrayed instance.	`Integer`	Default Value: `1` Valid Values: any integer greater than zero	
`instViewName`	Specifies the name of the view for `leHiCreateInst`.	`String`	Default Value: `layout` Valid Values: `layout`, `schematic`, or `symbolic`	
`keepFirstTermName`	Specifies if the first terminal name is not removed on the Create Pin form when the pin is created.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`keepRuler`	Specifies if a ruler stays visible in the window after the last coordinate is entered.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`labelAttach`	Specifies whether the system prompts you to attach labels to an object.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`labelDrafting`	Specifies whether labels are restricted to rotate no more than 90 degrees.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`labelFontStyle`	Specifies the default font of labels.	`Cyclic`	Default Value: `stick` Valid Values: `stick`, `gothic`, `roman`, `math`, `euroStyle`, `swedish`, `mask`, or `script`	
`labelHeight`	Specifies the height of labels in user units.	`Floating`	Default Value: `1.0` user units Valid Values: any number greater than zero	
`labelJustify`	Specifies the default justification of a label with respect to its origin.	`Cyclic`	Default Value: `centerCenter` Valid Values: `lowerLeft`, `centerLeft`, `upperLeft`, `lowerCenter`, `centerCenter`, `upperCenter`, `lowerRight`, `centerRight`, or `upperRight`	
`labelOverbar`	Determines how text strings containing underscore characters are displayed.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`labelText`	Specifies the default label text. If no text, the function exits.	`String`	Default Value: none Valid Values: any string with no spaces	
`layerTapCycle`	Controls whether the Layer Tap command either selects the layer of the object whose edge is closest to the cursor (nil) or cycles through the layers of overlapped objects (t).	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`leCheckMissingLayers`	Determines if checking is done to identify layers not defined in the technology file. When set to `t`, checking is done on maskLayout cellviews only.	`Boolean`		Default Value: `nil` Valid Values: `t` or `nil`
`leCheckMissingMasters`	Determines if unbound masters will be checked while opening a design. When set to `t`, or if the cellview is not masLayout, the check is done.	`Boolean`		Default Value: `nil` Valid Values: `t` or `nil`
`leDispNetExpr`	Determines if the net expression of a terminal is displayed in the text label of pins.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`
`leWindowBBox`	Specifies the default location and size of a layout window.	`String`	Default Value: `((120 185) (1020 875))` Valid Values: any valid screen coordinates	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`lockAngles`	Specifies that angles of partially selected shapes are not modified when the shapes are stretched. This option has no effect on instances or arrays.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`magnification`	Specifies the magnification with which instances are placed. This setting is not applicable if the number of rows or columns specified is not equal to `1`. (Arrays cannot be magnified.)	`Floating`	Default Value: `1.0` user units Valid Values: any number greater than zero	
`maintainConnections`	Determines whether automatic reconnection mode for pin-path connections is activated.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`markerSeverity`	Determines whether warning and error markers are found and highlighted in `geHiFindMarker( )`.	`String`		Default Value: `both` Valid Values: `warning`, `error`, `both`, or `none`
`markerZoom`	Determines whether to zoom to the next or previous marker while using the schematic editor's Find Marker form. Does not apply to the layout editor.	`Boolean`		Default Value: `nil` Valid Values: `t` or
`markerZoomScale`	Determines how close to the selected objects to zoom.	`Integer`	Default Value: `1` Valid Values: any positive integer	Default Value: `4` Valid Values: any positive integer

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`maxDragFig`	Maximum number of figures outlined during a drag operation (for example, *Move*). When the number of figures exceeds this value, the system displays a bounding box enclosing the figures.	`Integer`	Default Value: `50` Valid Values: any integer from `1` to `50`	Default Value: `500` Valid Values: any integer from `1` to `500`
`modCornerRadius`	Specifies the default modify-corner-radius value.	`Floating`	Default Value: `1.0` user units Valid Values: any positive floating-point number	
`modCornerType`	Specifies the default modify-corner type.	`Cyclic`	Default Value: `radial` Valid Values: `radial` or `chamfer`	
`modalCommands`	Specifies if commands are modal (automatically repeat).	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`moveToLayer`	Specifies whether shapes are moved to a different layer with `leHiMove`.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`mppASCIIFileName`	Specifies the default file name (or path and file name) for the ASCII File field in the Save Multipart Path Templates form and the Template File field in the Load Multipart Path Template form. When you change the ASCII File field, the system changes the Template File field to match.	`String`	Default Value: `techLIbName.mpp.templates.il` Valid Values: `techLIbName.mpp.templates.il` or *`your_value`*`.il` where *`your_value`* is the new value for `mppASCIIFileName`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`mppTemplate`	Specifies the default for the *MPP Template* field in the Create ROD Multipart Path form. You can choose *New* or the name of a specific MPP template you created.	`String`	Default Value: `New` Valid Values: `New` or the name of any MPP template defined in your technology file	
`multiSegRuler`	Specifies if a ruler by default is multisegmented, showing cumulative distance.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`nPtsToTaper`	Specifies the default number of segments in the taper. This is only used for exponentially tapered lines.	`Integer`	Default Value: `20` Valid Values: any positive integer	
`netsOn`	Determines whether nets are highlighted.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`
`numLevels`	Specifies the number of levels to copy in `leYankFigs`.	`Integer`	Default Value: `32` Valid Values: an integer from `0` to `32`	
`numSides`	Specifies the number of sides used to convert a conic to a polygon.	`Integer`	Default Value: `20` Valid Values: any positive integer up to `2047`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`openToStopLevel`	Determines when the cellview data is loaded during rendering. When set to `t` all data is loaded to the window stop level, then rendered. All data is loaded, even data too small to see. When set to `nil` data loads on demand during rendering. This can slightly slow initial redraw, zooms or pans, but will dramatically reduce the amount of data loaded, since data too small to see is not rendered. If you have set the `drawEngine` environment variable to `1`, `drawEngine` takes precedence.	`Boolean`		Default Value: `nil` Valid Values: `t` or `nil`
`orientation`	Specifies the orientation with which instances are placed.	`Cyclic`	Default Value: `R0` Valid Values: `R0`, `R90`, `R180`, or `R270`	
`originMarkersOn`	Determines whether the cell origins are displayed when viewing the bounding boxes of cell instances. The origins appear as plus signs.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
packetDialogBoxes	Determines whether a message box appears if a packet assigned to a layer in the technology file is not defined in the display.drf file. The message box lists the undefined packets.	Boolean		Default Value: t Valid Values: t or nil
partialSelect	Controls whether you can select vertices or line segments of an object.	Boolean	Default Value: nil Valid Values: t or nil	Default Value: t Valid Values: t or nil
pathBeginExt	Specifies the beginning extension of paths with variable end style.	Floating	Default Value: 0.0 user units Valid Values: any positive floating-point number	
pathCL	Determines whether you see path borders and centerlines (yes), path borders only (no), or path centerlines only (only).	String	Default Value: yes Valid Values: yes, no, or only	Default Value: yes Valid Values: yes, no, or only
pathEndExt	Specifies the end extension of paths with variable end style.	Floating	Default Value: 0.0 user units Valid Values: any positive floating-point number	
pathJustify	Specifies the default justification of paths.	Cyclic	Default Value: center Valid Values: left, center, or right	
pathOffset	Specifies the offset of a path centerline from the drawn coordinates.	Floating	Default Value: 0.0 user units Valid Values: any positive floating-point number	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`pathStyle`	Specifies the default end style for paths. The end style can be `truncateExtend` (flush), `extendExtend` (offset), `roundRound` (octagon), or `varExtendExtend` (variable).	`Cyclic`	Default Value: `truncateExtend` Valid Values: `truncateExtend`, `extendExtend`, `roundRound`, or `varExtendExtend`	
`pathWidth`	Specifies the default width of paths.	`Floating`	Default Value: `0.6` user units Valid Values: any positive floating-point number	Default Value: `0.5` user units Valid Values: any positive floating-point number in user units
`pinAccessDir`	Specifies the default access direction of a pin.	`String`	Default Value: `list ("left" "right" "top" "bottom")` Valid Values: any combination of `left`, `right`, `top`, or `bottom`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`pinSelectionMode`	Determines the selectability of pins and shapes. When set to `ignoreLPP` pins are selectable even if the LPP they are on is not selectable. When set to `useLPPIncludeShapes` pins and shapes are selectable only if the LSW pin button is on and the LPP of the pin or shape is set to selectable in the LSW. When set to `useLPPExcludeShapes` only pins are selectable and only if the LSW pin button is set to on.	`Cyclic`	Default Value: `ingnoreLPP` Valid Values: `ignoreLPP`, `useLPPIncludeShapes`, `useLPPExcludeShapes`	
`pinShape`	Specifies the default type of pin shape. When you choose auto, the pin function prompts for path centerline. A rectangle pin is automatically generated from the path centerline. The pin access direction is automatically set.	`Cyclic`	Default Value: `rectangle` Valid Values: `rectangle`, `polygon`, `dot`, or `auto pin`	
`pinTextLayer`	Specifies the layer of the layer-purpose pair used for pin text displays and on the Pin Name Display form. These are the labels created for pin names.	`String`	Default Value: `text` Valid Values: any valid layer	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`pinTextPurpose`	Specifies the purpose of the layer-purpose pair used for pin text displays and on the Pin Name Display form. These are the labels created for pin names.	`String`	Default Value: `drawing` Valid Values: any valid purpose	
`pinTextSameLayer`	Determines whether to create the pin name on the same layer as the pin being created or create it on the pin name layer.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	
`pinType`	Specifies the default pin type.	`String`	Default Value: none Valid Values: any symbolic pin type defined in the technology file	
`pinWidth`	Specifies the default pin width.	`Floating`	Default Value: `0.6` user units Valid Values: any positive floating-point number	
`pinXPitch`	Specifies the horizontal distance between the center points of pins in an array.	`Floating`	Default Value: `0.0` user units Valid Values: any positive floating-point number	
`pinYPitch`	Specifies the vertical distance between the center points of pins in an array.	`Floating`	Default Value: `0.0` user units Valid Values: any positive floating-point number	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
pinsAreSymbolic	Determines whether to display the Create Symbolic Pin form (t) or the Create Shape Pin form (nil) by default. If this is set to t but no symbolic pins are defined in your technology file, the Create Shape Pin form appears.	Boolean	Default Value: t Valid Values: t or nil	
polygonArc	When set to t, indicates that the next three points you enter in the polygon define an arc. Resets to nil when you enter the third point.	Boolean	Default Value: nil Valid Values: t or nil	
preserveAlignInfoOn	Specifies whether alignment between ROD objects is preserved or broken when a ROD object is moved or modified.	Boolean	Default Value: t Valid Values: t or nil	
preservePins	Specifies whether pin information is preserved when an instance is flattened.	Boolean	Default Value: nil Valid Values: t or nil	
preserveRodObjects	Specifies whether ROD properties are preserved when an instance is flattened.	Boolean	Default Value: nil Valid Values: t or nil	
propClass	Specifies the default property class that appears on the Properties command form.	Cyclic	Default Value: attributes Valid Values: unknown, attributes, parameters, connectivity, or other	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`readCellviewWarningOn`	Determines whether, when you open a cellview containing aligned ROD objects in read-only mode, the system displays a warning in the CIW stating that alignments in the design data might not be current.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	Default Value: `nil` Valid Values: `t` or `nil`
`recursionCheck`	Specifies whether to check for recursive hierarchy when creating instances.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`removeChopShape`	Specifies whether the area cut by `leHiChop` or `leChopShape` should be removed.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`replaceFigs`	Specifies whether selected figures are replaced with a new cell created from the selected figures. This setting is used by `leHiMakeCell` and `leMakeCell`.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`reshapeType`	Specifies the default shape used by `leHiReShape`.	`Cyclic`	Default Value: `rectangle` Valid Values: `unknown`, `line`, or `rectangle`	
`rodAutoName`	Specifies that rectangles, polygons and/or shape pins be created as ROD objects. These shapes are given either system- or user-assigned names.	`String`	Default Value: none Valid Values: `pin`, `rectangle`, `polygon`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
segSnapMode	Determines how *Edit* command entry points snap to the grid.	String	Default Value: orthogonal Valid Values: orthogonal, diagonal, anyAngle, horizontal, or vertical	Default Value: anyAngle Valid Values: orthogonal, diagonal, anyAngle, horizontal, or vertical
selectionAperture	Determines the distance the cursor must be within before gravity snaps the cursor.	Integer	Default Value: 5 Valid Values: any positive integer	Default Value: 5 Valid Values: any positive integer
selectOnlyInstWithSelectableShapes	Determines whether instances are selectable. If t an instance is selected only if the LSW Inst button is on and the instance or instances contain shapes that are on selectable layers. If nil an instance or instances are selected when the LSW Inst button is on, even if none of the shapes within them are on selectable layers.	Boolean	Default Value: nil Valid Values: t or nil	
snapMode	Determines how *Create* command entry points snap to the grid.	String	Default Value: orthogonal Valid Values: orthogonal, diagonal, anyAngle, L90XFirst, or L90YFirst	Default Value: anyAngle Valid Values: orthogonal, diagonal, anyAngle, L90XFirst, or L90YFirst
snapToGrid	Determines whether the graphics editor snap-to-grid mechanism is enabled.	Boolean	Default Value: nil Valid Values: t or nil	Default Value: t Valid Values: t or nil

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`startLevel`	Number of the highest level in the hierarchy visible.	`Integer`	Default Value: `0` Valid Values: any integer from `0` to `32`	Default Value: `0` Valid Values: any integer from `0` to `32`
`stopLevel`	Number of the lowest level in the hierarchy visible.	`Integer`	Default Value: `0` Valid Values: any integer from `0` to `32`	Default Value: `0` Valid Values: any integer from `0` to `32`
`stretchPcellApplyToName`	For pcells, in the Stretch in X and Stretch in Y forms, determines whether the *Reference Dimension* (default) and *Minimum and Maximum* fields apply to Name only (t) or to the whole expression (nil).	`Boolean`	Default Value: t Valid Values: t or nil	
`stretchHandlesLayer`	Specifies the layer that stretch handles are displayed on for stretchable pcells.	`String`		Default Value: `y0 drawing` Valid Values:any valid layer-pupose pair
`taperStyle`	Specifies the default type of taper.	`Cyclic`	Default Value: `linear` Valid Values: `unknown`, `linear`, or `exponential`	
`termDirection`	Specifies the default I/O type of a pin.	`Cyclic`	Default Value: `inputOutput` Valid Values: `input`, `output`, `inputOutput`, `switch`, or `jumper`	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`termName`	Specifies one or more names for terminals on which pins are created. You can specify as many names as you want (separated by spaces). As each name is used, it is removed from the list. If all names are used, the field in the Pin form becomes empty, and you cannot create any more pins until you specify a terminal name.	`String`	Default Values: none Valid Values: any terminal name	
`textJustificationOn`	Determines whether the origins of labels are displayed. The origins appear as plus signs.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`
`tinyInstDetail`	Determines whether to display all the details of an instance, even those smaller than the size set in the variable, `filterSize`. When `nil`, the cellview redraws faster.	`Boolean`	Default Value: `nil` Valid Values: `t` or `nil`	Default Value: `nil` Valid Values: `t` or `nil`
`trlBendFac`	Specifies the maximum width (as a multiple of its standard width) that standard bends in a transmission line can achieve before they are automatically chamfered. This attribute is used only for bend, style bend.	`Floating`	Default Value: `1.0` user units Valid Values: any positive floating-point number	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`trlBendStyle`	Specifies the types of bends used in creating a transmission line.	`Cyclic`	Default Value: `bend` Valid Values: `bend`, `chamfer`, or `radial`	
`trlChamFac`	Specifies the width of a chamfered bend in a transmission line as a multiple of its standard width. This attribute is used only for bend, style chamfer.	`Floating`	Default Value: `0.6` user units Valid Values: any positive floating-point number	
`trlNPtsToPI`	Specifies the number of segments in a radial bend in a transmission line as a function of the number of segments in a 180-degree radial bend.	`Integer`	Default Value: `20` Valid Values: any positive integer in user units	
`trlRadFac`	Specifies the radius of a radial bend in a transmission line as a multiple of the transmission-line width. This attribute is used only for bend-style radial.	`Floating`	Default Value: `3.0` user units Valid Values: any positive floating-point number	
`trlWidth1`	Specifies the width of the first end of a transmission line bend.	`Floating`	Default Value: `20.0` user units Valid Values: any positive floating-point number	
`trlWidth2`	Specifies the width of the second end of a transmission line bend.	`Floating`	Default Value: `20.0` user units Valid Values: any positive floating-point number	

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
`updatePCellIncrement`	Specifies how often the system updates pcell parameters and regenerates a pcell during a stretch operation.	`Floating`	Default Value: Every grid snap, as defined by the technology file variable `mfgGridResolution`.	
`useDefaultVia`	Specifies whether the default via is used or a list of defined vias appears. If `t`, the via that has defaultVia defined in the technology file is used; if no via has defaultVia defined, the first via found in the list is used. If `nil` and multiple vias are defined, the Select Contact form appears.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`useEditorBackgroundColorForLSW`	Specifies the background color for the layer purpose icons on the Layer Selection Window. When set to `t`, the background will be the background color of the layout editor cell design window. When set to `nil`, the background color will be the background color set in the `.Xdefaults` file.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	
`useTrueBBox`	Specifies whether the cellview bounding box, including labels (`t`), or the instance master bounding box (`nil`) displays. When `nil`, large designs open faster.	`Boolean`	Default Value: `t` Valid Values: `t` or `nil`	Default Value: `t` Valid Values: `t` or `nil`

Variable Name	Description	Type	Layout Editor Value	Graphics Editor Value
viewNameToTypeList	Specifies the correct view name associated with the default tool name in the Create New File form.	String		Default Value: none Valid Values: any valid view name
xSnapSpacing	Controls the minimum distance the cursor moves in the X direction.	Floating	Default Value: 0.1 user unit Valid Values: any positive floating-point number	Default Value: 1.0 user unit Valid Values: any positive floating-point number
ySnapSpacing	Controls the minimum distance the cursor moves in the Y direction.	Floating	Default Value: 0.1 user unit Valid Values: any positive floating-point number	Default Value: 1.0 user unit Valid Values: any positive floating-point number
yankShape	Specifies the default shape used by leHiYank.	Cyclic	Default Value: rectangle Valid Values: rectangle or polygon	
zoomToFig	Specifies whether the cellview zooms to objects found in a search.	Boolean	Default Value: nil Valid Values: t or nil	

Environment Variables Used Internally

The following graphic and layout editor environment variables are for internal use only or are reserved for future use. Changing the values of these variables may cause the layout editor or other Cadence software to behave improperly or quit. Many of these variables are read-only and cannot be changed.

abutPerpSnapOn, abutServerOn, alignHiliteArrowsOn, alignHiliteLayer, alignHiliteObjectsOn, coalesceLimit, coalesceRatio, confirmDelete, connStatus, contactInstLibName, criticality, eipRedraw, evalILExprOn, figOrder, interNum, jumpersOn, markerShowIgnore, minEnclosureCheckingOn, minLabelSize, minRefreshArea, minSpacingCheckingOn, minStippleSize, minWidthCheckinOn, moreDialogs, mppSubShapeHiliteLayer, netName, outlineOnly, pickNCreate, pinInstLibName, probeTextOn, pruneSize, toggle, transformMarkers, transformMarkersOn, viewNameList

envGetVal

For information on `envGetVal`, see the *User Interface SKILL Functions Reference* manual.

envSetVal

For information on `envSetVal`, see the *User Interface SKILL Functions Reference* manual.

leAutoRoute

```
leAutoRoute(
     )
     => t/nil
```

Description

Runs the *Perform Route* command in a Virtuoso layout accelerator window. The design is immediately exported, routed, and imported back to the open cellview. The Export to Router and Import to Router forms do not appear. All environmental options for exporting and importing the design are set in the `.cdsenv` file.

leEIPZoomAbsoluteScale

```
leEIPZoomAbsoluteScale(
     )
     => t/nil
```

Description

Fits the master cell of the instance you want to edit in the window. The window is redrawn so the edit-in-place cell fills it.

leEnvLoad

```
leEnvLoad(
     )
     => t/nil
```

Description

Loads the layout editor environment variables from the `.cdsenv` files. This file must be in your home directory. The variables are read into memory, but the windows are not updated with the new values. When new windows are created, they assume the new values.

leGetEnv

```
leGetEnv(
     t_name
     )
     => g_value/nil
```

Description

Returns the value currently assigned to the layout environment properties with the variable *t_name*.

Arguments

t_name	Name of the layout environment variable.

Value Returned

g_value/nil	Returns the value of the variable *t_name*. If the specified variable does not exist, it returns `nil`.

Example

```
leGetEnv( "gravityOn" )
```

Returns the value of the variable `gravityOn`.

leSetEnv

```
leSetEnv(
     t_name
     g_value
     )
     => t/nil
```

Description

Sets the layout environment properties with the variable *t_name* to the value *g_value*.

Arguments

t_name	Name of the layout environment property variable assigned the value *g_value*. Valid Values: any valid SKILL expression
g_value	Value of the variable *t_name*. Valid Values: any value that matches the value type defined for the variable

Value Returned

`t/nil`	Returns `t` if the value is assigned; otherwise, it returns `nil`.

Example

```
leSetEnv( "gravityOn" nil )
```

Sets gravity off in the current window.

leToggleGravity

```
leToggleGravity(
     )
     => t/nil
```

Description

Turns gravity on or off.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if gravity is toggled; otherwise, it returns `nil`.

Example

```
leToggleGravity()
```

Sets gravity off if it was previously on, or on if it was previously off.

leToggleMaintainConnections

```
leToggleMaintainConnections(
     )
     => t/nil
```

Description

Turns Maintain Connections on or off.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if Maintain Connections is toggled; otherwise, it returns `nil`.

Example

```
leToggleMaintainConnections()
```

Sets Maintain Connections off if it was previously on, or on if it was previously off.

Object Creation and Plotting Functions

You can create a variety of objects and templates for multipart paths with the layout editor. These objects are described in this section.

Arrays

Arrays are created using `dbMosaic` objects. You can create simple mosaics only. Although you can manipulate the location and orientation of complex mosaics, you cannot create them, nor can you change their composition. You must use the structure compiler tool for these operations.

Circles

Circles are created using `dbEllipse` objects. Circles are stored as ellipses, where the bounding box of the ellipse is a square. A circle does not have to remain a circle; you can stretch it into an ellipse.

Contacts

Contacts are created using `dbInst` objects. Contacts are defined in the technology file and must be parameterized cells with specific parameter names.

Donuts

Donuts are created using `dbDonut` objects.

Ellipses

Ellipses are created using `dbEllipse` objects.

Instances

Instances are created using `dbInst` objects.

Labels

Labels are created using `dbLabel` objects.

Multipart Path Templates

One way to define a template for a mulitpart path is by specifying the `leDefineMPPTemplate` function in an ASCII file. A multipart path is a single relative object design (ROD) object consisting of one or more parts at level zero in the hierarchy on the same or on different layers.

Paths

Paths are created using `dbPath` objects. You can use the *Path* command to create multilayer paths, which are multiple paths on different layers with contact instances connecting the different layers.

Pins

Rectangle, polygon, dot, and symbolic (instance) type pins can be created. They are created as `dbRect`, `dbPolygon`, `dbDot`, or `dbInst` types, with `dbPins` attached to them. From a database point of view, a pin is a logical object, not a physical object; the object seen is just the figure used to represent the pin. Pins can be created on any layer and can be of any size.

The system prompts you for the terminal name.

Polygons

Polygons are created using `dbPolygon` objects.

Procedural Access

All figure types can be created procedurally. Many of these procedures are identical to database functions and maintain a consistent naming convention.

Rectangles

Rectangles are created using `dbRect` objects.

Tapered Transmission Lines

Tapered transmission lines are a special type of transmission line: they are a single transmission line segment that has different widths at either end. The transition from one

width to the other can be either linear or exponential. Tapered transmission lines are represented in the database in the same way as transmission lines.

Transmission Line Bends

Transmission line bends are special types of transmission lines. They contain three points and, therefore, define two segments and one bend. Unlike normal transmission lines, the two segments of a transmission line bend can have different widths. Transmission line bends are represented in the database in the same way as transmission lines.

Transmission Lines

Transmission lines are similar to paths, except that you can specify how bends in transmission lines are built. You can specify standard bends, chamfered bends, or radial bends. Transmission lines are useful in extremely high-frequency circuits.

leCreateAutoInstPin

```
leCreateAutoInstPin(
     d_cellViewId
     l_point
     t_termName
     t_termDir
     )
     => d_pinShapeId/nil
```

Description

Creates a pin shape in cellview *d_cellViewId*.

Arguments

d_cellViewId	Database ID of the cellview in which the pin is created.
l_point	The point at which to create the `instPin`, a `symDevice` pin defined in your technology file.
t_termName	Name of the terminal to which the pin is attached. If no such terminal exists, one is created. If the terminal must be created, it is created on a net with the same name as the terminal. Valid Values: any valid SKILL expression
t_termDir	Specifies the I/O type (direction) of the pin. Valid Values: `input`, `output`, `inputOutput`, `switch`, or `jumper`

Value Returned

d_pinShapeId/`nil`	Returns the object ID for the pin if the pin is created; otherwise, it returns `nil`.

Example

```
leCreateAutoInstPin( cell1 10:10 "term1" "input" )
```

Creates a pin in cellview `cell1` at `10:10`. The pin is attached to terminal `term1`, and the access direction of the pin is `input`. Returns the object ID of the pin.

leCreateAutoPin

```
leCreateAutoPin(
     d_cellViewId
     l_point
     t_termName
     t_termDir
     )
     => d_pinShapeId/nil
```

Description

Creates a pin shape in cellview *d_cellViewId*.

Arguments

d_cellViewId	Database ID of the cellview in which the pin is created.
l_point	The point at which to create the rectangle pin.
t_termName	Name of the terminal to which the pin is attached. If no such terminal exists, one is created. If the terminal must be created, it is created on a net with the same name as the terminal. Valid Values: any valid SKILL expression
t_termDir	Specifies the I/O type (direction) of the pin. Valid Values: `input`, `output`, `inputOutput`, `switch`, or `jumper`

Value Returned

d_pinShapeId/`nil`	Returns the object ID for the pin if the pin is created; otherwise, it returns `nil`.

Example

```
leCreateAutoPin( cell1 list(10:10) "term1" "input" )
```

Creates a pin in cellview `cell1` at `10:10`. The pin is attached to terminal `term1`, and the access direction of the pin is `input`. Returns the object ID of the pin.

leCreateContact

```
leCreateContact(
     d_cellViewId
     t_contact
     l_origin
     t_orient
     n_width
     n_length
     x_rows
     x_cols
     n_xPitch
     n_yPitch
     t_xBias
     t_yBias
     )
     => d_instId/nil
```

Description

Creates *t_contact* type of instances in cellview *d_cellViewId* with the specified attributes. The specified contact must be defined in the library of *d_cellViewId*.

Arguments

d_cellViewId	Database ID of the cell used to create the instance.
t_contact	Specifies the type of contact. Valid Values: any valid contact type as defined in this library's technology file
l_origin	Coordinates for the origin of the contact.
t_orient	Mirrors and rotates the contact clockwise. Valid Values: R0, R90, R180, R270, MX, MXR90, MY, or MYR90
n_width	Specifies the width of the contact, or the width of each contact-cut in a contact array. Valid Values: any number greater than zero
n_length	Specifies the length of the contact, or the length of each contact-cut in a contact array. Valid Values: any number greater than zero

x_rows	Specifies the number of rows in the array. Valid Values: an integer greater than zero
x_cols	Specifies the number of columns in the array. Valid Values: an integer greater than zero
n_xPitch	Specifies the stepping distance between the columns of the contact. Valid Values: any number greater than or equal to zero
n_yPitch	Specifies the stepping distance between the rows of the contact. Valid Values: any number greater than or equal to zero
t_xBias	Specifies the location of the contact origin. Valid Values: `left`, `center`, or `right`
t_yBias	Specifies the location of the contact origin. Valid Values: `top`, `center`, or `bottom`

Value Returned

d_instId/`nil`	Returns the instance ID if the contact is created; otherwise, it returns nil.

Example

```
leCreateContact( cell1 "con1" 0:0 "R0" 1 1 1 1 0 0 "center" "center" )
```

Creates a one-by-one array of a type `con1` contact in `cell1`. The origin of the contact is located at `0:0`. It is one user unit square, and the contact origin is in the center of the contact.

Interactive Function

```
leHiCreateContact( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you for the type of contact, the origin of the contact, the width and length of the contact, the number of rows and columns in the array, the X and Y pitch, and the orientation of X and Y bias. If you do not specify *w_windowId*, the layout editor uses the current window.

leCreatePath

```
leCreatePath(
     d_cellViewId
     l_layerPurposePair
     l_points
     n_width
     [t_pathStyle]
     [n_offset]
     [t_justification]
     )
     => d_pathId/nil
```

Description

Creates a path in cellview *d_cellViewId* on the specified layer with point list *l_points* and width *n_width*.

Arguments

d_cellViewId	Database ID of the cellview in which the path is created.
l_layerPurposePair	Layer-purpose pair of the path. Valid Values: any valid layer name and purpose if other than `drawing`
l_points	List of coordinates used to create the path.
n_width	Specifies the width of the current segment of the path. Valid Values: any non-negative number in user units
t_pathStyle	Specifies how the path segment ends are built for the current path segment. Valid Values: `truncateExtend`, `extendExtend`, `roundRound`, or `varExtendExtend`
n_offset	Specifies the offset from the points you enter to the actual path. Valid Values: a positive or negative number
t_justification	Specifies whether the path points correspond to the centerline of the path, its left edge, or its right edge. The offset is relative to the specified edge. Valid Values: `left`, `center`, or `right`

Value Returned

d_pathId/nil	Returns the object ID of the path if the path is created; otherwise, it returns nil.

Example

```
leCreatePath( cell2 list("metal1R") list(0:0 10:0 10:10) 5 )
```

Creates a path in cellview cell2 on layer metal1R with the coordinates 0:0, 10:0, 10:10 and returns the object ID of the path. The path has a width of 5.

Interactive Function

```
leHiCreatePath( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to enter the layer, coordinates, width, and options. If you do not specify *w_windowId*, the layout editor uses the current window.

leCreatePin

```
leCreatePin(
     d_cellViewId
     l_layerPurposePair
     t_shape
     l_points
     t_termName
     t_termDir
     l_accessDir
     )
     => d_pinShapeId/nil
```

Description

Creates a pin shape in cellview *d_cellViewId* on the specified layer.

Arguments

d_cellViewId	Database ID of the cellview in which the pin is created.
l_layerPurposePair	Specifies the layer-purpose pair on which to create the pin. Valid Values: any valid layer and purpose name
t_shape	Type of shape to use for the pin. Valid Values: `rectangle` or `polygon`
l_points	List of points to use for the creation of the pin. Valid Values: two for a rectangle and more than two for a polygon pin
t_termName	Name of the terminal to which the pin is attached. If no such terminal exists, one is created. If the terminal must be created, it is created on a net with the same name as the terminal. Valid Values: any valid SKILL expression
t_termDir	Specifies the I/O type (direction) of the pin. Valid Values: `input`, `output`, `inputOutput`, `switch`, or `jumper`
l_accessDir	List of access directions of the pin. This field is used only for rectangular pins. Valid Values: `none`, `top`, `bottom`, `left`, or `right`

Value Returned

d_pinShapeId/nil	Returns the object ID for the pin if the pin is created; otherwise, it returns nil.

Example

```
leCreatePin( geGetEditRep() "metal1R" "rectangle" list(0:0 10:10) "term1" "input"
list("left" "right") )
```

Creates a pin in the current cellview on layer metal1R from 0:0 to 10:10. The pin is attached to terminal term1; the I/O type of the pin input and the access directions of the pin are left and right. Returns the object ID of the pin.

Interactive Function

```
leHiCreatePin( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to enter the terminal name, pin shape, I/O type, and access direction for the pin. If you do not specify *w_windowId*, the layout editor uses the current window.

leDefineMPPTemplate

```
leDefineMPPTemplate(
          ?techId             d_techId
          ?name               S_name
          ?layer              txl_layer
          [?width             n_width]
          [?justification     S_justification]
          [?offset            n_offset]
          [?endType           S_endType]
          [?beginExt          n_beginExt]
          [?endExt            n_endExt]
          [?choppable         g_choppable]

          [ROD Connectivity Arguments]
          [?offsetSubPath     l_offsetSubpathArgs...]
          [?encSubPath        l_encSubpathArgs...]
          [?subRect           l_subrectArgs...]

     ) ; end leDefineMPPTemplate
     => t | nil

          ; ROD Connectivity Arguments
          [?termIOType              S_termIOType]
          [?pin                     g_pin]
          [?pinAccessDir            tl_pinAccessDir]
          [?pinLabel                g_pinLabel]
          [?pinLabelHeight          n_pinLabelHeight]
          [?pinLabelLayer           txl_pinLabelLayer]
          [?pinLabelFont            S_pinLabelFont]
          [?pinLabelDrafting        g_pinLabelDrafting]
          [?pinLabelOrient          S_pinLabelOrient]
          [?pinLabelOffsetPoint     l_pinLabelOffsetPoint]
          [?pinLabelJust            S_pinLabelJust]
          [?pinLabelRefHandle       S_pinLabelRefHandle]
          ; end of ROD Connectivity Arguments

          ;l_offsetSubpathArgs Offset Subpath Arguments
          list(
               list(
                    ?layer            txl_layer
                    [?width           n_width]
                    [?sep             n_sep]
                    [?justification   S_justification]
                    [?beginOffset     n_beginOffset]
                    [?endOffset       n_endOffset]
                    [?choppable       g_choppable]
                    [Repeat ROD Connectivity Arguments here]
                    ) ;End of first offset subpath list
                    ...
               ) ;End of offset subpath lists
```

```
;End of l_offsetSubpathArgs

;l_encSubpathArgs Enclosure Subpath Arguments
list(
     list(
          ?layer            txl_layer
          [?enclosure       n_enclosure]
          [?beginOffset     n_beginOffset]
          [?endOffset       n_endOffset]
          [?choppable       g_choppable]
          [Repeat ROD Connectivity Arguments here]
          ) ;End of first enclosure subpath list
          ...
     ) ;End of enclosure subpath lists
;End of l_encSubpathArgs

;l_subrectArgs Subrectangle Arguments
list(
     list(
          ?layer            txl_layer
          [?width           n_width]
          [?length          n_length]
          [?gap             S_gap]
          [?sep             n_sep]
          [?justification   S_justification]
          [?beginOffset     n_beginOffset]
          [?endOffset       n_endOffset]
          [?space           n_space]
          [?choppable       g_choppable]
          [Repeat ROD Connectivity Arguments here]
          ) ;End of first subrectangle list
          ...
     ) ;End of subrectangle lists
;End of l_subrectArgs
```

Description

The syntax of this function provides the structure for defining the field names and default values for one multipart path (MPP) in an ASCII file. This information is referred to as an MPP *template*. A multipart path is a single relative object design (ROD) object consisting of one or more parts at level zero in the hierarchy on the same or on different layers. The purpose of an MPP template is to create MPPs in layout cellviews using predefined values. You can define any number of MPP templates in a single ASCII file by specifying `leDefineMPPTemplate` once for each MPP template; each template must be identified by a template name (*t_name*) that is unique within the ASCII file.

When you load an ASCII MPP template file, the system checks the names of the ASCII MPP templates against the names of MPP templates that already exist in the technology file in

virtual memory. If there is a name conflict, the system displays a message in the CIW saying that an MPP template in the ASCII file is replacing an existing MPP template. The existing template is overwritten in virtual memory only. If you do not want to overwrite the original MPP template in your binary technology library on disk, do not save changes to the technology file. You can also avoid overwriting existing templates by changing the matching names in your ASCII file. For more information about what to do, see "Problem: Warning in CIW: MPP template name is replacing an existing MPP template" in the *Virtuoso Relative Object Design User Guide*.

The system merges the ASCII MPP template definitions into the technology file in virtual memory. When you choose the Virtuoso® layout editor *Create – Multipart Path* command (and press F3), the system displays the names of all MPP templates in the pull-down *MPP Template* field on the *Create Multipart Path* form.

You can load ASCII files in any of the following ways:

- In the Virtuoso® layout editor, with the *Create – Multipart Path* command, using the *Load Template* option.
- In the CIW, with the following statement:

  ```
  load "pathToFile/templateFilename.il"
  ```

  For example,

  ```
  load "../skill/mppTemplates.il"
  ```

- In the `.cdsinit` file
  You can identify ASCII MPP template files in your `.cdsinit` file. The system automatically executes the `.cdsinit` file when the Cadence® design framework II product starts, and loads the identified files. For information about the `.cdsinit` file, see the *Virtuoso Layout Editor User Guide*.
- In the `libInit.il` file
  You can attach ASCII MPP template files to your library by identifying them in a `libInit.il` file. The system automatically executes the `libInit.il` file when a library is opened and loads attached files.

  Note: Avoid using graphical SKILL functions such as `hiSetBindKey` in your `libInit.il` file; doing so will result in an error. For more information see "What to Avoid in the libInit.il File" in the *Virtuoso Parameterized Cell Reference*.

  To attach a file to a library,

  - Place the file within the library directory.
  - In the library directory, create (or update) a file named `libInit.il`.
  - Within the `libInit.il` file, write a `load` statement like this:

```
load "pathToFile/templateFilename.il"
```

Note: Loading MPP templates from an ASCII file affects only the temporary version of your technology library in virtual memory. If you want the loaded templates and changes you make to their values with the *Create – Multipart Path* form to persist beyond the end of the current editing session, you must save the changes to your binary technology library on disk before you exit the software. Remember that MPP templates loaded from ASCII files overwrite templates with matching names in the technology file in virtual memory, and when you save changes to your technology file, the virtual memory version overwrites your technology library on disk.

Arguments

For the *d_techId* and *t_name* arguments, see the description below.

All other arguments for `leDefineMPPTemplate` have the same names, definitions, valid values, and default values as the arguments for the `rodCreatePath` function. For a detailed description of each argument, see the "Arguments" section of the `rodCreatePath` function in the *Virtuoso Relative Object Design User Guide*. (The links in the syntax statement go to the related argument definition sections in the *Relative Object Design User Guide*.)

d_techId	Database ID of the technology library associated with the current cellview.
t_name	Character string enclosed in double quotation marks specifying the name of the MPP template. The name must be unique within the MPP templates in the ASCII file. Do not assign the name `New`; it is a reserved name. Default: none

Value Returned

`t`	Returned if the argument list is created successfully.
`nil`	Returned if the argument list is not created successfully.

Differences Between leDefineMPPTemplate and rodCreatePath Syntax

The syntax for specifying an MPP template with `leDefineMPPTemplate` is based on the syntax of the `rodCreatePath` function. Most arguments are the same and have the same default values. The differences between the syntax for an MPP template specified with `leDefineMPPTemplate` and `rodCreatePath` are summarized below:

- For the arguments that `leDefineMPPTemplate` and `rodCreatePath` have in common, the argument definitions, valid values, and default values are identical.
- `leDefineMPPTemplate` has one additional argument: *`t_name`*
- For some arguments, the data type is more restricted for `leDefineMPPTemplate` than it is for `rodCreatePath`. Specifically, you can enter either a character string or a symbol for `rodCreatePath` arguments with the data type *`S_`*; for the equivalent `leDefineMPPTemplate` arguments, you must enter a character string (the data type is *`t_`* for text).
- You cannot specify an expression as a value for any of the `leDefineMPPTemplate` arguments.
- `leDefineMPPTemplate` does not contain the following `rodCreatePath` arguments:

  `d_cvId`
  The current cellview is used.

  `S_name`
  The system generates a default ROD name when you create an MPP in a cellview; you can change it in the *ROD Name* field on the Create ROD Multipart Path form or in the Edit Properties form.

  `S_netName`
  When the MPP template you choose includes the definition of pin connectivity, to create an MPP in a layout cellview, you must enter a value for the *Net Name* field in the Create ROD Multipart Path form.

  `S_termName`
  When the MPP template includes the definition of pin connectivity, the system assigns the net name you enter for terminal name.

  `l_pts`
  Click in the layout cellview to specify the point list.

  `l_prop`
  There is no property field in the Create Multipart Path forms, so this argument has been omitted.

- `leDefineMPPTemplate` does not contain the `rodCreatePath` arguments listed below because these arguments represent an alternate way of specifying a point list for an MPP, and the point list varies for each occurrence of an MPP within a layout cellview.

  `dl_fromObj` *`txf_size`*
  `l_startHandle` *`l_endHandle`*

For detailed information about the `rodCreatePath` function, its arguments, and the default values, see the `rodCreatePath` section of the *Virtuoso Relative Object Design User Guide*.

Using the leDefineMPPTemplate Function

You can create an ASCII MPP template file either

- with a text editor
- or with the Virtuoso® layout editor by using the *Save Template* option on the Create Multipart Path form and saving to ASCII

The following example shows an ASCII file containing the definitions for two MPP templates.

- When you create an ASCII template file with a text editor, you need to substitute your own library and technology file names for the variables *`myLibraryName`* and *`myTechFileName`* in the example below.
- If you enter an invalid technology file name, the system displays a warning message in the CIW and no templates are loaded.
- When you save the values in the Create Multipart Path form to ASCII, the system replaces the *`myLibraryName`* and *`myTechFileName`* variables with the name of the current library and the name of the current technology file.

Note: If you load an ASCII file that was created with the Create Multipart Path form, make sure you are using the same library and technology file that was current when the ASCII file was created.

```
;; This procedure identifies the technology library and technology file names, and
;; opens the technology file in read-only mode.If the technology library does not
;; open, the procedure displays an error message and exits.If the technology library
;; opens,the procedure continues, and reads the leDefineMPPTemplate functions,
;; each of which define the fields and values for a multipart path (MPP).

prog(
     ( tech techLibName techFileName)
          techLibName =  "myLibraryName"
          techFileName = "myTechFileName"
          tech =         techOpenTechFile( techLibName techFileName "r" )

          if(tech==nil
          then
               return(nil)
          ) ; end if

     ; Template1 defines an MPP consisting of only a master path.

     leDefineMPPTemplate(
          ?techId             tech
```

```
      ?name               "template1"
      ?layer              "poly1"
      ?width              0.600000
      ?choppable          t
      ?endType            "flush"
      ?justification      "center"
      ?offset             0.700000
      ?termIOType         "switch"
      ?pin                t
      ?pinAccessDir       list( "bottom" "left" "right" )

) ; end leDefineMPPTemplate function for template1

; Template2 defines an MPP consisting of a master path, three offset subpaths,
; three enclosure subpaths, and three sets of subrectangles.

leDefineMPPTemplate(
                    ; Master path
                    ?techId          tech
                    ?name            "Template2"
                    ?layer           "cellBoundary"
                    ?width           0.600000

          ; Three offset subpaths

          ?offsetSubPath
                    list(
                          list(
                                ?layer           "pwell"
                                ?width           1.000000
                                ?choppable       t
                                ?sep             2.000000
                          ) ; end of first sublist
                          list(
                                ?layer           "poly1"
                                ?width           1.000000
                                ?choppable       t
                                ?sep             4.000000
                          ) ; end of second sublist
                          list(
                                ?layer           "metal1"
                                ?width           1.000000
                                ?choppable       t
                                ?sep             6.000000
                          ) ; end of third sublist
                    ) ; end of offset subpath lists

          ; Three enclosure subpaths

          ?encSubPath
                    list(
                          list(
                                ?layer           "metal2"
                                ?enclosure       0.100000
                                ?choppable       t
                                ?beginOffset     0.100000
                                ?endOffset       0.200000
                          ) ; end of first sublist
                          list(
                                ?layer           "metal3"
                                ?enclosure       0.150000
```

```
                                        ?choppable      t
                                        ?beginOffset    0.200000
                                        ?endOffset      0.300000
                                  ) ; end of second sublist
                                  list(
                                        ?layer          "ndiff"
                                        ?enclosure      0.180000
                                        ?choppable      t
                                        ?beginOffset    0.300000
                                        ?endOffset      0.400000
                                  ) ; end of third sublist
                            ) ; end of enclosure subpath lists

            ; Three sets of subrectangles

            ?subRect
                            list(
                                  list(
                                        ?layer          "pdiff"
                                        ?width          0.500000
                                        ?length         0.600000
                                        ?choppable      t
                                        ?sep            0.200000
                                        ?space          1.000000
                                  ) ; end of first sublist
                                  list(
                                        ?layer          "pwell"
                                        ?width          0.600000
                                        ?length         0.800000
                                        ?choppable      t
                                        ?sep            0.500000
                                        ?space          3.000000
                                  ) ; end of second sublist
                                  list(
                                        ?layer          "poly1"
                                        ?width          0.800000
                                        ?length         1.000000
                                        ?choppable      t
                                        ?sep            2.000000
                                        ?space          5.000000
                                  ) ; end of third list
                             ) ; end of subrectangle lists
    ) ; end leDefineMPPTemplate function for template2

) ; end prog
```

lePlot

```
lePlot(
     [t_fileName]
     )
     => t/nil
```

Description

Procedural plot submission command that generates the plot defined in the *t_fileName* plot template file. Plot options are stored in a disembodied property list called `lePlotOptions`. When `lePlot()` loads the plot template file, it reads the plot options from the `lePlotOptions` property list. There are two ways to create the plot template file: use `vi`, or save the options in the Submit Plot form to a file. The plot template file is used by `lePlot()` and by the *Load* command in the Submit Plot form. However, the following options are ignored by the *Load* command: `library`, `cell`, `view`, and `plotSize`.

Arguments

t_fileName	A plot template file that stores plot options in a disembodied property list named `lePlotOptions`.

Value Returned

`t/nil`	`t` if the operation was successful, `nil` otherwise.

Examples

```
lePlot("myPlotTemplateFile")
```

Plots a cellview as specified in `myPlotTemplateFile`.

Sample lePlotOptions template file

```
lePlotOptions = '(nil
     area              "whole"
     view              "layout"
     cell              "Inv"
     library           "master"
     plot              "cellview"
     bBox              ((0.0 0.0) (10.0 36.0))
     noteText          "For Your Review"
     notes             t
     header            t
     plotterType       "postscript1"
```

```
     papersize         "A"
     plotter           "HP"
     display           "display"
     outputfile        ""
     time              "now"
     tmpdir            "/usr/tmp"
     copy              1
     scale             7408.333333
     center            nil
     mailto            "user_name"
     mail              t
     orientation       "automatic"
     plotsize          (2.916667 10.5)
     offset            (0.0 0.0)
     unit              "inches"
     gridMultiple      5
     gridSpacing       1.000000
     gridType          "Dots"
     stopLevel         32
     startLevel        0
     arrayDisplay      "Full"
     instName          "instance"
     pathCL            "yes"
     drawAxesOn        t
     iconsOn           nil
)
```

You can also insert the `lePlotOptions` into the `.cdsinit` file. Each time you print, the options from that template file are used. However, the library name, cell name, and the view name in the `lePlotOptions` template file are not used, they are generated by the cellview window from which you are printing.

Plot Options and Descriptions

Plot Option	Type	Description
`area`	String	`whole` plots entire cellview, `select` plots selected area, `reselect` plots a reselected area.
`arrayDisplay`	String	`Full` prints all instances in the array, `Border` prints only the instances around the outside edge of the array, `Source` prints only the instance in the lower left corner of the array.
`bBox`	Floating	Specifies in user units the part of the cellview to be plotted. Defaults to the cellview's bounding box.
`cell`	String	Name of the cell to be plotted. Used by `lePlot()` only.

Plot Option	Type	Description
`center`	Boolean	If `t`, the design is centered on the page.
`copy`	Integer	Number of copies to print.
`display`	String	Display name that specifies the display packets to use when generating the plot.
`drawAxesOn`	Boolean	If `t`, the axes are plotted.
`gridMultiple`	Integer	Major grid lines or dots are displayed every `gridMultiple` units. Major grid lines are thicker than minor grid lines. Major grid dots are larger than minor grid dots. This corresponds to the *Major Spacing* field in the Plot Display Options form.
`gridSpacing`	Integer	Minor grid lines or dots are displayed every `gridSpacing` units. Minor grid lines are thinner than major grid lines. Minor grid dots are smaller than major grid dots. This corresponds to the *Minor Spacing* field in the Plot Display Options form.
`gridType`	String	Specifies grid type of `None`, `Dots`, or `Lines`.
`header`	Boolean	If `t`, displays a header and legend on the plot.
`iconsOn`	Boolean	If `t`, displays only outlines of instances in arrays.
`instName`	Boolean	If `t`, displays instance name.
`library`	String	Name of the cell's library to be plotted. Used by `lePlot()` only.
`mail`	Boolean	If `t`, plot notification is e-mailed to addresses specified by `mailto`. If `nil`, plot notification is not e-mailed.
`mailto`	String	Addresses to receive plot notification. Separate addresses with spaces.
`notes`	Boolean	If `t`, displays information specified by `noteText`.
`noteText`	String	Text displayed on the plot if `notes` is `t`.

Plot Option	Type	Description
`offset`	Floating	Specifies the X and Y origin of the cellview or viewing area in measurements specified by `unit`. The offset is a pair of floating-point values. If the plot spans more than one page, the offset is from the bottom left corner.
`orientation`	String	Sets what edge of the paper to use as the top. `portrait` plots the cellview as it appears in window, `landscape` rotates the cellview 90 degrees counterclockwise, `automatic` plots the cellview whichever way fits best.
`outputfile`	String	Saves plot information to the specified file instead of sending it to the printer.
`papersize`	String	Name of paper size for specified plotter.
`pathCL`	String	If `yes`, plots paths with center lines; if `no`, plots paths without centerlines; if `only`, plots centerlines only.
`plotsize`	Floating	Specifies the width and height of the resulting plot in inches. If `plotsize` is not specified, one of the following occurs: ■ If `scale` is specified, the `plotsize` is the `bBox` size scaled. ■ If area is `whole`, the `plotsize` is the size of the `bBox` for the plotter and `papersize`. ■ If none of the above is true, the `plotsize` defaults to 8x10.5 inches. Used by `lePlot()` only.
`plotter`	String	Name of the plotter specified in the `.cdsplotinit` file.
`scale`	Floating	Specifies the magnification of the plot in measurements specified by `unit`.
`startLevel`	Integer	Specifies the first level in the design hierarchy to be plotted.
`stopLevel`	Integer	Specifies the last level in the design hierarchy to be plotted.

Plot Option	Type	Description
`time`	String	Specifies the time to plot the design. The plot is stored in `tmpdir` until it is plotted. Use one of the following formats to set time: - `now` (plot immediately) - *`hh`*`:`*`mm`* `am/pm` *`day`* (plot at specified time, for example: `03:15 PM Wed`)
`tmpdir`	String	Specifies the directory where the plot is to be stored if it is not sent to the plotter.
`unit`	String	Specifies the type of measurement for `plotsize`, `scale`, and `offset`. Valid units are `inches`, `cm`, `mm`, and `feet`. Default is `inches`.
`view`	String	Name of the cell's view to be plotted. Used by `lePlot()` only.

Object Editing Functions

This section describes the procedural equivalents to the interactive editing functions on the *Edit* menu. These procedural functions perform like the interactive functions but have no user interface associated with them. They can be used to implement new user-defined functions.

leAttachFig

```
leAttachFig(
     d_figId1
     [d_figId2]
     )
     => t/nil
```

Description

Attaches *d_figId1* to *d_figId2* by making *d_figId1* a child of *d_figId2*. If *d_figId1* is already attached to another object, it is detached from it. If *d_figId2* is not specified, *d_figId1* is detached from any object it is currently attached to without attaching it to anything else.

Arguments

d_figId1	Database ID of the child object.
d_figId2	Database ID of the parent object.

Value Returned

`t/nil`	Returns `t` if the child is attached; otherwise, it returns `nil`.

Example

```
leAttachFig( fig1 fig2 )
```

Makes the object `fig1` a child of the object `fig2` and returns `t`.

```
leAttachFig( fig1 )
```

Detaches `fig1` from any object it is currently attached to without attaching it to anything else and returns `t`.

Interactive Function

`leHiAttach( [w_windowId] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the child object and the parent object. If you do not specify *w_windowId*, the layout editor uses the current window.

leChopShape

```
leChopShape(
     d_shapeId
     l_points
     g_closed
     [g_remove]
     [x_sides]
     )
     => l_newShapes/nil
```

Description

Cuts the shape *d_shapeId* using the chop shape *l_points*.

Arguments

d_shapeId	Database ID of the shape to be cut.
l_points	List of points describing the chop shape.
g_closed	Specifies whether the chop shape is closed or not. Valid Values: `t` or `nil`
g_remove	Specifies whether to remove the objects inside the chop shape. Default Value: `nil` Valid Values: `t` or `nil`
x_sides	Specifies the number of sides to be used when converting conic shapes to polygons before cutting them. Default Value: `20` Valid Values: any positive integer

Value Returned

l_newShapes/`nil`	Returns the list of shapes resulting from the operation if the shape is chopped; otherwise, it returns `nil`.

Example

```
leChopShape( shape2 list(0:0 0:10 8:8) nil nil 10 )
```

Using the coordinates in the list, cuts a triangle out of the shape `shape2` after converting it to a polygon with 10 sides. Does not close the chop shape and does not remove any object. Returns the object IDs of the new shapes.

Note: The layout editor does not remove objects if the chop shape is not closed. An open chop shape does not clearly indicate which objects are “inside.”

Interactive Function

`leHiChop( [`*`w_windowId`*`] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the shape to be chopped, to create the chop shape, and specify if the shape is closed. If you do not specify *`w_windowId`*, the layout editor uses the current window.

leConvertShapeToPolygon

```
leConvertShapeToPolygon(
     d_shapeId
     [x_sides]
     )
     => d_polygonId/nil
```

Description

Converts shape *d_shapeId* to an equivalent polygon. If it is a conic, the polygon has the specified number of sides.

Arguments

d_shapeId	Database ID of the shape to convert.
x_sides	Specifies the number of sides to use when converting conic shapes. Default Value: `20` Valid Values: any positive integer up to `2047`.

Value Returned

d_polygonId/`nil`	Returns the object ID of the polygon if it is created; otherwise, it returns `nil`.

Example

```
leConvertShapeToPolygon( shape2 10 )
```

Converts the conic shape `shape2` to a polygon with 10 sides and returns the object ID for the polygon.

Interactive Function

```
leHiConvertShapeToPolygon( [w_windowId] ) => d_polygonId/nil
```

Enter this function with only the window ID argument; the system prompts you to point to the shape to convert. If you do not specify *w_windowId*, the layout editor uses the current window.

leDefineExternalPins

```
leDefineExternalPins()
     => t/nil
```

Description

Starts the Define External Pins enter function in the current window. The Define External Pins enter function provides a user interface for defining one or more pins on a net as being externally connected outside of a device.

For more information, see the Virtuoso Layout Editor online documentation.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the command executed successfully and `nil` if an error occurred.

Example

```
when(leDefineExternalPins()
         info("Define External Pins executed successfully.\n")
)
```

Starts the Define External Pins enter function. Prints the message when the command successfully finishes.

leDefineInternalPins

```
leDefineInternalPins()
     => t/nil
```

Description

Starts the Define Internal Pins enter function in the current window. The Define Internal Pins enter function provides a user interface for defining one or more pins on a net as being internally connected within a device.

For more information, see the Virtuoso Layout Editor online documentation.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the command executed successfully and `nil` if an error occurred.

Example

```
when(leDefineInternalPins()
         info("Define Internal Pins executed successfully.\n")
)
```

Starts the Define Internal Pins enter function. Prints the message when the command successfully finishes.

leDefinePPPins

```
leDefinePPPins(
     [w_windowId]
     )
     => t/nil
```

Description

Starts the Define Pseudo Parallel Connected Net enter function in the current window. Corresponds to the *Connectivity – Define Pins – Pseudo Parallel Connect* command. The Define Pseudo Parallel Connected Net form provides a user interface for defining one or more pins on a net as being pseudo parallel connected. If you do not specify *w_windowId*, the layout editor uses the current window.

For more information, see the Virtuoso Layout Editor online documentation.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t/nil	Returns t if the command executed successfully and nil if an error occurred.

Example

```
when(leDefinePPPins()
         info("Define Pseudo Parallel Connected Net executed successfully.\n")
)
```

Starts the Define Pseudo Parallel Connected Net enter function. Prints the message when the command successfully finishes.

leDefineWeaklyConnectedPins

```
leDefineWeaklyConnectedPins()
     => t/nil
```

Description

Starts the Define Weak Pins enter function in the current window. The Define Weak Pins enter function provides a user interface for defining one or more pins on a net as being weakly connected within a device.

For more information, see the Virtuoso Layout Editor online documentation.

Arguments

None

Value Returned

`t/nil`	Returns `t` if the command executed successfully and `nil` if an error occurred.

Example

```
when(leDefineWeaklyConnectedPins()
         info("Define Weakly Connected Pins executed successfully.\n")
)
```

Starts the Define Weak Pins enter function. Prints the message when the command successfully finishes.

leFlattenInst

```
leFlattenInst(
     d_instId
     x_levels
     [g_flattenPCells]
     [g_preservePins]
     [g_preserveRODobjs]
     )
     => t/nil
```

Description

Flattens instance *d_instId* up through *x_levels* of hierarchy.

Arguments

d_instId	Database ID of the instance to flatten.
x_levels	Number of levels of hierarchy through which to flatten the instance. Valid Values: an integer between 0 and 20
g_flattenPCells	Specifies whether pcells are flattened. Default Value: `nil` Valid Values: `t` or `nil`
g_preservePins	Specifies whether pin connectivity information is preserved. Default Value: `nil` Valid Values: `t` or `nil`
g_preserveRODobjs	Specifies whether ROD properties are preserved. Default Value: `nil` Valid Values: `t` or `nil`

Value Returned

`t/nil`	Returns `t` if the instance is flattened; otherwise, it returns `nil`.

Example

```
leFlattenInst( instance1 2 nil )
```

Flattens the instance `instance1` through two levels of the hierarchy and does not flatten any pcells contained in the instance.

Interactive Function

```
leHiFlatten( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to point to the instance to flatten and specify the number of levels through which to flatten it. If you do not specify *w_windowId*, the layout editor uses the current window.

leMakeCell

```
leMakeCell(
     l_figs
     t_libName
     t_cellName
     t_viewName
     g_replace
     )
     => d_cellViewId/nil
```

Description

Creates the cellview specified by *t_libName*, *t_cellName*, and *t_viewName*. If the cellview already exists, it is overwritten.

Arguments

l_figs	List of objects contained in the original cellview.
t_libName	Specifies the name of the new cellview's library. Valid Values: any valid library name
t_cellName	Specifies the name of the new cellview's cell. Valid Values: any valid cellview name
t_viewName	Specifies the name of the new cellview's view. Valid Values: any valid view name
g_replace	Specifies whether the objects selected should be replaced by an instance of the new cellview. If this option is not selected, the original objects are not modified. Valid Values: `t` or `nil`

Value Returned

d_cellViewId/`nil`	Returns the new cellview ID if a new cellview is created; otherwise, it returns `nil`.

Example

```
leMakeCell( list(fig1 fig2) "mylib" "mycell" "layout" nil )
```

Creates a cellview containing copies of the objects `fig1` and `fig2` in a `layout` view of `mycell` in `mylib`. The objects in the original cellview are not replaced by the new cell. Returns the database ID of the new cellview.

Interactive Function

`leHiMakeCell( [`*`w_windowId`*`] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the objects contained in the new cellview; specify the library, cell, and view names; specify whether to replace the selected objects; and indicate an origin for the new cellview. If you do not specify *`w_windowId`*, the layout editor uses the current window.

leMergeShapes

```
leMergeShapes(
     l_shapes
     [x_sides]
     )
     => l_newShapes/nil
```

Description

Merges the list of shapes *l_shapes.* The input shapes can be on multiple layers. Merges only shapes on the same layer. Returns the list of shapes resulting from the operation.

Arguments

l_shapes	List of shapes to be merged.
x_sides	Specifies the number of sides to use when converting conic shapes to polygons before merging them. Default Value: `20` Valid Values: any positive integer

Value Returned

l_newShapes/`nil`	Returns the list of shapes resulting from the merge; otherwise, it returns `nil`.

Example

```
leMergeShapes( list(fig1 fig2 fig3) )
```

Merges the objects `fig1`, `fig2`, and `fig3` after converting any conics to polygons with the default of 20 sides and returns the object ID of the resulting shape.

Interactive Function

```
leHiMerge( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to point to the shapes to be merged. If you do not specify *w_windowId*, the layout editor uses the current window.

leModifyCorner

```
leModifyCorner(
     d_figId
     l_selArray
     g_chamfer
     x_distance
     [x_sides]
     )
     => t/nil
```

Description

Rounds or cuts off selected corners of the rectangle or polygon specified by *d_figID*.

Arguments

d_figId	The database ID of the polygon or rectangle to modify.
l_selArray	A list of Boolean values indicating whether each vertex in the shape can be modified. The Boolean values must match the number and order of vertices in the shape. (Use *d_shapeId*~>Points to obtain the vertices of the shape.)
g_chamfer	A Boolean value specifying whether you want to create a chamfer or a rounded corner. Valid Value: t (chamfer) or nil (round)
x_distance	The distance from the vertex to begin modifying the corner. The distance used is half the length of the longest line segment adjacent to the corner or the value supplied for *x_distance*, whichever is shorter. Valid Values: any positive floating value up to half the length of the longest adjacent line segment
x_sides	The number of sides to use when converting a corner to a curve. Default Value: 20 Valid Values: any positive integer

Value Returned

t/nil	Returns t if the corner is modified; otherwise, returns nil.

Example

```
leModifyCorner( fig1 t nil t 4.0 t )
```

Converts selected corners of `fig1` to chamfers using a distance of four user units. Returns `t`.

Interactive Function

```
leHiModifyCorner( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to select the corners to modify. If you do not specify *w_windowId,* the layout editor uses the current window.

leMoveCellViewOrigin

```
leMoveCellViewOrigin(
     d_cellViewId
     l_point
     )
     => t/nil
```

Description

Moves the contents of cellview *d_cellViewId* so that the origin is at *l_point*.

Arguments

d_cellViewId	Database ID for the cellview whose origin is moved.
l_point	Specifies the new origin for the cellview.

Value Returned

t/nil	Returns t if the origin moves; otherwise, it returns nil.

Example

```
leMoveCellViewOrigin( cell2 14:14 )
```

Moves the contents of cellview cell2 a negative 14 user units in the X direction and a negative 14 user units in the Y direction, so that the origin is at the original 14:14 location. Returns t.

Interactive Function

```
leHiMoveOrigin( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to point to the new origin. If you do not specify *w_windowId*, the layout editor uses the current window.

lePasteFigs

```
lePasteFigs(
     d_cellViewId
     l_destPt
     [t_rotation]
     )
     => t/nil
```

Description

Places the contents of the yank buffer in cellview *d_cellViewId* at the destination point *l_destPt*.

lePasteFigs works in layout editor windows opened using menus. It does not work in nongraphical windows opened with dbOpenCellViewByType.

See leYankFigs for information about copying objects into the yank buffer.

Arguments

d_cellViewId	Database ID for the cellview in which to place the yanked objects.
l_destPt	Coordinate specified as the destination point for the reference point of the yanked objects.
t_rotation	Rotates the contact clockwise, and mirrors the contact. Default Value: R0 Valid Values: R0, R90, R180, R270, MX, MXR90, MY, or MYR90

Value Returned

t/nil	Returns t if the objects are placed; otherwise, it returns nil.

Example

```
lePasteFigs( geGetEditCellView() 100:100 )
```

Pastes the objects currently in the yank buffer, placing the reference point of the yanked objects at 100:100.

Interactive Function

`leHiPaste( [`*`w_windowId`*`] ) => t/nil`

Enter this function with the window ID argument only; the system prompts you to point to the location to place the yanked objects. If you do not specify *`w_windowId`*, the layout editor uses the current window.

leSizeShape

```
leSizeShape(
     d_shapeId
     g_sizeFactor
     )
     => l_shapes/nil
```

Description

Sizes shape *d_shapeId* by *g_sizeFactor*.

Arguments

d_shapeId	Database ID of the shape to size.
g_sizeFactor	Specifies the amount by which the shape is sized in user units. Positive values oversize objects; negative values undersize objects. Valid Values: any positive or negative number

Value Returned

l_shapes/nil	Returns a list of the resulting shapes if they are sized; otherwise, it returns nil.

Example

```
leSizeShape( shape1 2 )
```

Oversizes the shape shape1 by a factor of 2.

Interactive Function

```
leHiSize( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to point to the shape to size and specify a size factor. If you do not specify *w_windowId*, the layout editor uses the current window.

leSplitShape

```
leSplitShape(
     d_shapeId
     l_delta
     l_splitLine
     g_leftOrRight
     [g_lockAngles]
     )
     => t/nil
```

Description

Splits a shape and moves one portion of it. Uses a multisegment split line to divide the shape.

Arguments

d_shapeId	The database ID of the shape you want to redraw.
l_delta	List specifying distances in the X and Y direction to move the split portion.
l_splitLine	List of points defining the split line.
g_leftOrRight	Boolean specifying whether the part to be moved is to the left or right of the split line. Left and right are defined by the order of the points in the split line. Valid Values: `t` (left) or `nil` (right)
g_lockAngles	Boolean specifying whether angles between edges should be preserved. Default Value: `t` Valid Values: `t` or `nil`

Return Values

`t/nil`	Returns `t` if the operation is successful. Otherwise returns `nil`.

Example

```
leSplitShape(car(geGetSelectedSet()) list(10 12) list(0:0 0:10) t t)
```

Moves the split portion of selected shape 10 units in the X direction and 12 units in the Y direction. The stretch group is to the left of the split line and the angles of the object do not change.

leStretchShape

```
leStretchShape(
     d_shapeId
     l_delta
     l_freePoints
     [g_lockAngles]
     )
     => t/nil
```

Description

Stretches shape *d_shapeId* by the delta *l_delta.* Donuts cannot be stretched.

Arguments

d_shapeId	Database ID of the shape.
l_delta	List of the X and Y distances to stretch the shape. You can specify the X and Y distances in list format, `list ( 0 2 )` or in coordinate format, `0:2`.
l_freePoints	List of Booleans indicating whether each point in the shape is free to be moved or not. The Booleans must match the number and order of points in the shape. If all of the vertices of the shape are marked as free, the entire shape is moved. (You can use `geGetSelSetFigPoint(d_shapeId)` to return the list of Booleans corresponding to the point list of shape *d_shapeId*.)
g_lockAngles	Specifies that angles of partially selected shapes should not be modified when the shapes are stretched. This option has no effect on instances or arrays. Valid Values: `t` or `nil`

Value Returned

`t/nil`	Returns `t` if the shape is stretched; otherwise, it returns `nil`.

Example

```
leStretchShape( shapeA list( 0 2 ) list(nil t t nil) t )
```

Stretches the right edge of shape `shapeA` 2 units in the Y direction. Two points of the shape are free to move and the angles of the shape cannot be changed. You can also use `geGetSelSet()` and `geGetSelSetFigPoint()` to get the shape ID and list of Booleans as shown here:

```
leStretchShape( car(geGetSelSet()) 0:2
geGetSelSetFigPoint( car(geGetSelSet()) ) t )
```

Interactive Function

`leHiStretch( [`*`w_windowId`*`] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the shape to stretch, specify the distance to stretch, and specify freepoints. If you do not specify *`w_windowId`*, the layout editor uses the current window.

leYankFigs

```
leYankFigs(
     d_cellViewId
     l_ptArray
     x_nLevels
     [l_refPt]
     [x_nSides]
     )
     => t/nil
```

Description

Copies specified objects into a yank buffer for later pasting. Copies objects in cellview *d_cellViewId* that fall within the shape defined by the points in *l_ptArray* and that are in the levels of hierarchy specified by *x_nLevels*. Optionally uses *l_refPt* as the reference point for the yank and *x_nSides* as the number of polygon sides to use when converting objects intersected by the yank shape.

See lePasteFigs for more information about placing the contents of the yank buffer into a cellview.

Arguments

d_cellViewId	Database ID for the cellview containing the objects to yank.
l_ptArray	List of the coordinates defining the yank shape.
x_nLevels	Specifies the number of hierarchy levels to traverse.
l_refPt	A reference point used for later placing the yanked objects with `lePasteFigs`. Default Value: Coordinates of the lower left point of the bounding box for the yank shape Valid Values: coordinates of any point
x_nSides	Specifies the number of sides to use when converting objects to polygons. Default Value: `20` Valid Values: any positive integer

Value Returned

`t/nil`	Returns `t` if the objects are copied; otherwise, it returns `nil`.

Example

```
leYankFigs( geGetEditCellView() list( 0:0 10:0 10:10 0:10 ) 2 ) )
```

Copies all objects and partial objects in the current cellview that fall in the square region 0:0 to 10:10 into a buffer for future pasting. Objects and partial objects are copied down to two levels.

Interactive Function

`leHiYank( [`*`w_windowId`*`] ) => t/nil`

Enter this function with the window ID argument only; the system prompts you to enter the coordinates of the yank shape. If you do not specify *`w_windowId`*, the layout editor uses the current window.

Selection Functions

Most of the selection functions are global functions used by all applications. Layer selectability is independent of the layer used for shape creation. Each library has its own entry layer, which is the layer that all shape creation functions use.

leDeiconifyLSW

```
leDeiconifyLSW(
     )
     => t/nil
```

Description

Deiconifies the Layer Selection Window (LSW). It can be used if the LSW has been iconified. If the LSW is already displayed, this function has no effect.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the LSW is deiconified; otherwise, it returns `nil`.

Example

```
leDeiconifyLSW()
```

Deiconifies the Layer Selection Window.

leGetEntryLayer

```
leGetEntryLayer(
     [d_techFileId]
     )
     => l_layerPurposePair/nil
```

Description

Returns the entry layer for technology file *d_techFileId* as a list containing the layer name and layer purpose. If *d_techFileId* is not specified, the current technology file is used.

Arguments

d_techFileId Database ID for the technology file specified.

Value Returned

l_layerPurposePair/nil
Returns a list of the layer name and purpose if an entry layer for technology file *d_techFileId* exists; otherwise, it returns nil.

Example

```
leGetEntryLayer( myTech )
```

Returns the layer name and layer purpose for the entry layer in the technology file myTech.

leGetLSWBBox

```
leGetLSWBBox(
     [g_includesBorder]
     )
     => bBox/nil
```

Description

Returns the bounding box of the Layer Selection Window (LSW). The bounding box is set by either leSetLSWBBox(), Opus.LSWGeometry, or by moving the LSW window.

Arguments

[g_includesBorder]

If `nil`, the returned bounding box excludes the border and window title. If `t`, the returned bounding box includes the border and window title. By default `includesBorder` is `nil`.

Value Returned

bBox/`nil` Returns the bounding box of the LSW, or `nil` if the LSW has not been displayed.

Example 1

```
leGetLSWBBox()
((20 20)
     (220 820))
```

Returns the bounding box of the LSW excluding the border and window title.

Example 2

```
leGetLSWBBox(t)
((15 15)
     (225 845))
```

Returns the bounding box of the LSW including the border and window title. Border and window title size may be different depending on the window manager. This example is done in OpenWindows.

leGetValidLayerList

```
leGetValidLayerList(
     [d_techFileId]
     )
     => l_layerIdList/nil
```

Description

Returns a list of the valid entry layers for technology file *d_techFileId*. A layout view must be open for `leGetValidLayerList` to return the list of valid entry layers.

Arguments

d_techFileId	Database ID for the technology file specified.

Value Returned

l_layerIdList/`nil`	Returns a list of the valid entry layers for technology file *d_techFileId*; otherwise, it returns `nil`.

Example

```
leGetValidLayerList( myTech )
```

Returns the following list of layer-purpose pairs:

```
(nwell drawing
 pwell drawing
 ndiff drawing
 pdiff drawing
 diff drawing
 nsd drawing
 psd drawing
 Ptap drawing
 Ntap drawing
 nimplant drawing
 pimplant drawing
 poly1 drawing
 poly2 drawing
 metal1 drawing
 metal2 drawing
```

```
metal3 drawing
via drawing
via2 drawing)
```

leIconifyLSW

```
leIconifyLSW(
     )
     => t/nil
```

Description

Iconifies the Layer Selection Window (LSW). If the LSW is already iconified, this function has no effect.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the LSW is iconified; otherwise, it returns `nil`.

Example

```
leIconifyLSW()
```

Iconifies the Layer Selection Window.

leIsFigSelectable

```
leIsFigSelectable(
     d_figId
     )
     => t/nil
```

Description

Checks whether the figure *d_figId* is selectable.

Arguments

d_figId	Database ID of the figure to check.

Value Returned

t/nil	Returns t if figure *d_figId* is selectable; otherwise, it returns nil.

Example

```
leIsFigSelectable( fig4 )
```

Returns t if figure fig4 is selectable.

leIsInstSelectable

```
leIsInstSelectable(
     [d_techFileId]
     )
     => t/nil
```

Description

Checks whether the instances in *d_techFileId* are selectable. If *d_techFileId* is not specified, the current technology file is used.

Arguments

d_techFileId	Database ID of the technology file to check.

Value Returned

t/nil	Returns t if instances in *d_techFileId* are selectable; otherwise, it returns nil.

Example

```
leIsInstSelectable( myTech)
```

Returns t if instances in myTech are selectable.

leIsLayerSelectable

```
leIsLayerSelectable(
     l_layerPurposePair
     [d_techFileId]
     )
     => t/nil
```

Description

Checks whether the layer *l_layerPurposePair* is selectable.

Arguments

l_layerPurposePair
: Layer-purpose pair to check for selectability.

d_techFileId
: Database ID of the technology file containing the layer-purpose pair.

Value Returned

`t/nil`
: Returns `t` if layer *l_layerPurposePair* is selectable; otherwise, it returns `nil`.

Example

```
leIsLayerSelectable( list("poly1" "drawing") )
```

Returns `t` if layer-purpose pair `poly1 drawing` is selectable.

leIsLayerValid

```
leIsLayerValid(
     l_layerPurposePair
     [d_techFileId]
     )
     => t/nil
```

Description

Checks whether layer *l_layerPurposePair* is a valid layer. If *d_techFileId* is not specified, the current technology file is used.

Arguments

l_layerPurposePair
Layer-purpose pair to be checked for validity.

d_techFileId Database ID of the technology file that is checked.

Value Returned

`t/nil` Returns `t` if layer *d_layerPurposePair* is a valid layer; otherwise, it returns `nil`.

Example

```
leIsLayerValid( list("metal1" "drawing") lib2 )
```

Returns `t` if layer-purpose pair `metal1 drawing` in `lib2` is a valid layer.

leIsLayerVisible

```
leIsLayerVisible(
     l_layerPurposePair
     [d_techFileId]
     )
     => t/nil
```

Description

This procedure checks whether the layer specified by *l_layerPurposePair* is visible. If *d_techFileId* is not specified, the current technology file is used.

Arguments

l_layerPurposePair
Layer-purpose pair checked for visibility.

d_techFileId Database ID of the technology file containing the layer-purpose pair checked for visibility.

Value Returned

`t/nil` Returns `t` if layer *l_layerPurposePair* is visible; otherwise, it returns `nil`.

Example

```
leIsLayerVisible( list("layer1" "drawing") )
```

Returns `t` if `layer1` is a visible layer.

leIsLSWIconified

```
leIsLSWIconified(
     )
     => t/nil
```

Description

Checks if the Layer Selection Window (LSW) is iconified.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the LSW is iconified; returns `nil` if the LSW is not iconified.

Example

```
leIsLSWIconified()
```

Returns `t`.

leIsPinSelectable

```
leIsPinSelectable(
     [d_techFileId]
     )
     => t/nil
```

Description

Checks whether the pins in *d_techFileId* are selectable. If *d_techFileId* is not specified, the current technology file is used.

Arguments

d_techFileId	Database ID of the technology file to check.

Value Returned

t/nil	Returns t if pins in library *d_techFileId* are selectable; otherwise, it returns nil.

Example

```
leIsPinSelectable( myTechFile )
```

Returns t if pins in myTechFile are selectable.

leRaiseLSW

```
leRaiseLSW(
     )
     => t/nil
```

Description

Raises the Layer Selection Window (LSW) to the top of the window stack. It can be used if the LSW has been iconified. If the LSW is already at the top of the window stack, this function has no effect.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the LSW is raised to the top of the window stack; otherwise, it returns `nil`.

leRemapLSW

```
leRemapLSW()
     => t/nil
```

Description

If the Layer Selection Window (LSW) has been unmapped (made invisible) by `leUnmapLSW`, it restores the LSW window.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the LSW is restored; otherwise, it returns `nil`.

leResizeLSW

```
leResizeLSW(
     l_bBox
     )
     => t/nil
```

Description

Sets the Layer Selection Window's (LSW) bounding box to *l_bBox*. The bounding box specifies the lower left and upper right corners of the LSW window. See leSetLSWBBox() for more description.

leResizeLSW(l_bBox) is equivalent to leSetLSWBBox(l_bBox nil).

To verify the bounding box of the LSW, use leGetLSWBBox().

Arguments

l_bBox	List of two window coordinates specifying the lower left corner and the upper right corner of the LSW.

Value Returned

t/nil	Returns t if the LSW's bounding box has changed; otherwise, it returns nil.

Example

```
leResizeLSW( list( 20:20 220:820) )
```

Moves the LSW lower left corner to 20:20 and stretches the upper right corner to 220:820. The resulting LSW, excluding the border and window title, is 200 pixels wide and 800 pixels high.

leSetAllLayerSelectable

```
leSetAllLayerSelectable(
     g_isSelectable
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the selectability of all layers in technology file *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used.

Arguments

g_isSelectable	Specifies whether all layers in library *d_techFileId* are selectable. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file containing the layers.

Value Returned

`t/nil`	Returns `t` if the selectability of all layers is set to the specified value; otherwise, it returns `nil`.

Example

```
leSetAllLayerSelectable( t myTechFile )
```

Sets all layers in technology file `myTechFile` to be selectable.

leSetAllLayerValid

```
leSetAllLayerValid(
     g_isValid
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the validity of all layers in technology file *d_techFileId* as specified by *g_isValid*. If *d_techFileId* is not specified, the current technology file is used.

Arguments

g_isValid	Specifies whether the layers in technology file *d_techFileId* are valid. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file containing the layers.

Value Returned

`t/nil`	Returns `t` if the layers are set to the specified validity; otherwise, it returns `nil`.

Example

```
leSetAllLayerValid( t myTechFile)
```

Sets all layers in the technology file `myTechFile` to be valid.

leSetAllLayerVisible

```
leSetAllLayerVisible(
     g_isVisible
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the visibility of all layers in technology file *d_techFileId* as specified by *g_isVisible*. If *d_techFileId* is not specified, the current technology file is used.

Arguments

g_isVisible	Specifies whether all layers in technology file *d_techFileId* are visible. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file containing the layers.

Value Returned

`t/nil`	Returns `t` if visibility of all layers is set to the specified value; otherwise, it returns `nil`.

Example

```
leSetAllLayerVisible( t myTechFile )
```

Sets all layers in library `myTechFile` to be visible and returns `t`.

leSetEntryLayer

```
leSetEntryLayer(
     l_layerPurposePair
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the entry layer for technology file *d_techFileId* to the layer specified by *l_layerPurposePair*, which can be either a list containing a layer name and a layer purpose, or just a layer name (the layer purpose then defaults to `drawing`). If *d_techFileId* is not specified, the current technology file is used.

Arguments

l_layerPurposePair	Specifies the layer to use as the entry layer for library *d_techFileId*. Valid Values: any valid layer name and layer purpose
d_techFileId	Database ID for the technology file specified.

Value Returned

`t/nil`	Returns `t` if the entry layer is set; otherwise, it generates a warning message and returns `nil`.

Example

```
leSetEntryLayer( list("poly1" "drawing") myTechFile )
```

Makes `poly1` with the purpose `drawing` the entry layer for the technology file `myTechFile` and returns `t`.

leSetFormSnapMode

```
leSetFormSnapMode(
     t_SnapMode
     )
     => t_SnapMode
```

Description

Sets the cursor snap mode.

Arguments

t_SnapMode	Specifies the cursor snap mode. Valid Values: `diagonal`, `orthogonal`, `L90XFirst`, `L90YFirst`, `anyAngle`

Value Returned

t_SnapMode	Returns the cursor snap mode.

Example

```
leSetFormSnapMode("diagonal")
```

Sets the cursor snap mode and returns the snap mode set `"diagonal"`.

leSetInstSelectable

```
leSetInstSelectable(
     g_isSelectable
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the selectability of instances in the technology file specified by *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used.

Arguments

g_isSelectable	Specifies if the instances are selectable. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file.

Value Returned

`t/nil`	Returns t if the selectability of the instances is set to the specified value; otherwise, it returns `nil`.

Example

```
leSetInstSelectable( t myTechFile )
```

Sets instances in `myTechFile` to be selectable and returns `t`.

leSetLayerAttributes

```
leSetLayerAttributes(
     d_techFileId
     g_layerPurposePair
     l_attributeList
     )
     => t/nil
```

Description

Sets the attributes of the specified layer in the technology file *d_techFileId* as specified by *g_layerPurposePair* and *l_attributeList*.

Arguments

d_techFileId	Database ID of the technology file containing the layer.
g_layerPurposePair	Specifies the layer in the technology file whose attributes you change.
l_attributeList	Specifies attributes of the layer in the technology file *d_techFileId*. These attributes occur in the following order: *g_isValid*, *g_isSelectable*, *g_isVisible*. Only valid layers are displayed in the Layer Selection Window. A selectable layer must also be valid and visible. Valid Values: `t` or `nil`

Value Returned

`t/nil`	Returns `t` if the attributes of the layer are set to the specified values; otherwise, it returns `nil`.

Example

```
leSetLayerAttributes(techFileId list('("metal1Res" "drawing") t nil t))
```

Sets layer `metal1Res drawing` in `techFileId` to valid, not selectable, and visible.

leSetLayerSelectable

```
leSetLayerSelectable(
     l_layerPurposePair
     g_isSelectable
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the selectability of shapes on layer *l_layerPurposePair* in technology file *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used. For shapes on a layer to be selectable, the layer must be valid, visible, and selectable.

Arguments

l_layerPurposePair	Layer-purpose pair whose selectability is set.
g_isSelectable	Specifies if the layer *l_layerPurposePair* is selectable. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file.

Value Returned

`t/nil`	Returns `t` if selectability of shapes is set; otherwise, it returns `nil`.

Examples

```
leSetLayerSelectable( list("poly1" "drawing") t )
leSetLayerselectable( list("poly1") t )
leSetLayerSelectable( "poly1" t )
```

Any of these examples sets layer `poly1` purpose `drawing` to be selectable and returns `t`.

leSetLayerValid

```
leSetLayerValid(
     l_layerPurposePair
     g_isValid
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the validity of layer *l_layerPurposePair* as specified by *g_isValid*. If *d_techFileId* is not specified, the current technology file is used. Only valid layers are displayed in the Layer Selection Window.

Arguments

l_layerPurposePair	Layer-purpose pair whose validity you want to set.
g_isValid	Specifies whether the layer *l_layerPurposePair* is valid. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file containing the layer-purpose pair.

Value Returned

`t/nil`	Returns `t` if the validity of the layer-purpose pair is set; otherwise, it returns `nil`.

Example

```
leSetLayerValid( list("metal1" "drawing") nil myTechFile )
```

Sets the layer-purpose pair `metal1 drawing` in technology file `myTechFile` to be invalid.

leSetLayerVisible

```
leSetLayerVisible(
     l_layerPurposePair
     g_isVisible
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the visibility of the layer specified by the layer-purpose pair as specified by *g_isVisible*. If *d_techFileId* is not specified, the current technology file Layer Selection Window (LSW) is used.

Arguments

l_layerPurposePair	Layer-purpose pair to set visible.
g_isVisible	Specifies whether the layer is visible. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file.

Value Returned

`t/nil`	Returns `t` if the visibility of the layer is set to the specified value; otherwise, it returns `nil`.

Examples

```
leSetLayerVisible( list("poly1" "drawing") t )
leSetLayerVisible( list("poly1") t )
leSetLayerVisible( "poly1" t )
```

Any of these examples sets layer `poly1` purpose `drawing` to be visible and returns `t`.

leSetLSWBBox

```
leSetLSWBBox(
     l_bBox
     [g_includesBorder]
     )
     => t/nil
```

Description

Sets the Layer Selection Window's (LSW) bounding box to *l_bBox*. The bounding box specifies the lower left and upper right corners of the LSW window.

If *g_includesBorder* is `t`, the bounding box includes the border and title added by the window manager. If *g_includesBorder* is `nil`, the bounding box excludes the border and window title.

If the specified top left corner is off screen, then the top left corner is adjusted so that it is on screen.

If the specified width or height is greater than the screen's width or height, then the width or height is reset to the screen width or height.

If the specified width or height is less than the minimum width or height, then the width or height is reset to the minimum width or height. The minimum size of the LSW is 120 pixels wide by 175 pixels high.

To verify the resized bounding box of the LSW, use `leGetLSWBBox().`

Arguments

l_bBox	List of two window coordinates specifying the lower left corner and the upper right corner of the LSW.
[g_includesBorder]	If set to `nil`, the bounding box excludes the border and window title. If set to `t`, the bounding box includes the border and window title.

Value Returned

`t/nil`	Returns `t` if the LSW's bounding box has changed; otherwise, it returns `nil`.

Example 1

```
leSetLSWBBox( list( 20:20 220:820) )
leGetLSWBBox()
((20 20)
     (220 820))
leGetLSWBBox(t)
((15 15)
     (225 845))
```

`leSetLSWBBox` moves the lower left corner of the LSW to 20:20 and resizes the upper right corner to 220:820. The resulting LSW, excluding the border and window title, is 200 pixels wide and 800 pixels high.

`leGetLSWBBox()` returns the bounding box of the resized LSW excluding the border and window title.

`leGetLSWBBox(t)` returns the bounding box of the resized LSW including the border and window title. Border and window title size may be different depending on the window manager and screen size. These examples are done in OpenWindows.

Example 2

```
leSetLSWBBox( list( 30:30 230:1000) t)
leGetLSWBBox(t)
((30 0)
     (230 900))
```

There are two problems with the `leSetLSWBBox` specifications given in this example:

1. The top of the LSW would be off screen.
2. The LSW would be taller than the 900-pixel-high screen.

Because these specifications are not valid, they are reset to be valid. In this case, the LSW's upper left corner, including the window title, is moved to the top of the screen, and the height is resized by moving the bottom of the LSW up to the bottom of the screen. The resulting LSW bounding box is `30:0 230:900`, the size is 200 x 900 pixels.

`leGetLSWBBox(t)` returns the bounding box of the LSW including the border and window title. The lower left corner of the LSW, including the window border and title, is at `30:0`, the size is 200 x 900 pixels.

leSetPinSelectable

```
leSetPinSelectable(
     g_isSelectable
     [d_techFileId]
     )
     => t/nil
```

Description

Sets the selectability of pins in the technology file specified by *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used.

Arguments

g_isSelectable	Specifies if the pins in library *d_techFileId* are selectable. Valid Values: `t` or `nil`
d_techFileId	Database ID of the technology file.

Value Returned

`t/nil`	Returns `t` if the selectability of the pins is set to the specified value; otherwise, it returns `nil`.

Example

```
leSetPinSelectable( t myTechFile )
```

Sets pins in `myTechFile` to be selectable and returns `t`.

leUnmapLSW

```
leUnmapLSW()
     => t/nil
```

Description

Unmaps (makes invisible) the Layer Selection Window (LSW).

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the LSW becomes invisible; otherwise, it returns `nil`.

leZoomToPoint

```
leZoomToPoint(
     w_windowId
     l_points
     )
     => t/nil
```

Description

Places the cursor in the window, *w_windowId,* at the coordinates specified in *l_points*.

Arguments

w_windowId	Window ID of the window specified.
l_points	List of two XY coordinates.

Value Returned

t/nil	Returns t if the cursor is placed at the coordinates in the window specified; otherwise, it returns nil.

Example

```
leZoomToPoint(hiGetCurrentWindow(), 100:0)
```

leZoomToSelSet

```
leZoomToSelSet(
     [w_windowId]
     )
     => t/nil
```

Description

Zooms to fit the selected objects in the specified window. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window containing the selected objects.

Value Returned

t/nil	Returns t if the system successfully zooms the display to selected objects in the specified window; otherwise, it returns nil.

Reference Point Functions

A reference point is a location set by the user and maintained by the editor. Reference points are used to measure relative distances, such as the distance from the reference point to the cursor.

Each cellview can have only one reference point. The reference point can be active or inactive. You can display a star-marker at the location of the reference point by setting the `displayRefPoint` global environment variable. Reference points are displayed on the `hilight drawing3` layer.

Reference Point Display

Reference Point Setting

You can set the reference point coincident with a particular edge or vertex using `gravityOn`, then measure the distance from the reference point to the cursor as a means of separating objects by design rule spacings. You can also set the reference point automatically whenever you enter a point by setting the `autoSetRefPoint` global environment variable. The distance from the cursor to the reference point is displayed in the status banner of the window.

Moving the Cursor Using the Keyboard

You can move the cursor using bindkeys rather than the mouse. You can move the cursor to the reference point, then “bump” the cursor by fixed distances (for example, 0.5 microns, 1.0 microns) and enter a point using a bindkey. In this way, very accurate distances can be measured and created at “high altitude.”

Refer to the *Virtuoso Layout Editor Help* for more information about moving the cursor with the keyboard.

Procedural Access

The following procedures manipulate reference points and the cursor. Their functionality does not appear on the banner menus.

leGetRefPoint

```
leGetRefPoint(
     d_cellViewId
     )
     => l_point/nil
```

Description

Returns the reference point of cellview *d_cellViewId*. The reference point must be active.

Arguments

d_cellViewId	Database ID of the cellview containing the reference point.

Value Returned

l_point/nil	Returns *l_point* if leGetRefPoint gets the reference point; otherwise, returns nil.

Example

```
leGetRefPoint( cell2 )
```

Returns the coordinates of the reference point in the cellview cell2.

leIsRefPointActive

```
leIsRefPointActive(
     d_cellViewId
     )
     => t/nil
```

Description

Checks whether the cellview *d_cellViewId* contains an active reference point.

Arguments

d_cellViewId	Database ID of the cellview containing the reference point.

Value Returned

t/nil	Returns t if the reference point is set active for cellview *d_cellViewId*; otherwise, it returns nil.

Example

```
leIsRefPointActive( cell2 )
```

Returns t if the reference point is active in the cellview cell2.

leMoveCursor

```
leMoveCursor(
     f_dX
     f_dY
     )
     => t/nil
```

Description

Moves the cursor in the current window a relative distance specified by *f_dX* and *f_dY*.

Arguments

f_dX	Distance to move the cursor in the X direction.
f_dY	Distance to move the cursor in the Y direction.

Value Returned

`t/nil`	Returns `t` if the cursor moves; otherwise, it returns `nil`.

Example

```
leMoveCursor( 5.5 10.5 )
```

Moves the cursor 5.5 in the X direction and 10.5 in the Y direction and returns `t`.

leMoveCursorToRefPoint

```
leMoveCursorToRefPoint(
     )
     => t/nil
```

Description

Moves the cursor to the reference point of the cellview contained in the current window.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the reference point is active. If the reference point of the cellview is not active, it generates an error dialog box and returns `nil`.

Example

```
leMoveCursorToRefPoint()
```

Moves the cursor to the location of the reference point and returns `t`.

leSetRefPoint

```
leSetRefPoint(
     d_cellViewId
     l_point
     )
     => t/nil
```

Description

Sets the reference point of cellview *d_cellViewId* to the point *l_point*. The reference point is set to be active if it was inactive.

Arguments

d_cellViewId	Database ID of the cellview containing the reference point.
l_point	Coordinate specified as the reference point.

Value Returned

t/nil	Returns t if the reference point is set; otherwise, a warning message is generated and nil is returned.

Example

```
leSetRefPoint( cell2 25:25 )
```

Locates the reference point of the cellview cell2 at the coordinate 25:25 and returns t.

leSetRefPointInactive

```
leSetRefPointInactive(
     d_cellViewId
     )
     => t/nil
```

Description

Marks the reference point for cellview *d_cellViewId* as inactive. To make a reference point active again, use `leSetRefPoint`.

Arguments

d_cellViewId	Database ID of the cellview containing the reference point.

Value Returned

`t/nil`	Returns `t` if the reference point is set inactive; otherwise, it returns `nil`.

Example

```
leSetRefPointInactive( cell2 )
```

Makes the reference point in the cellview `cell2` inactive and returns `t`.

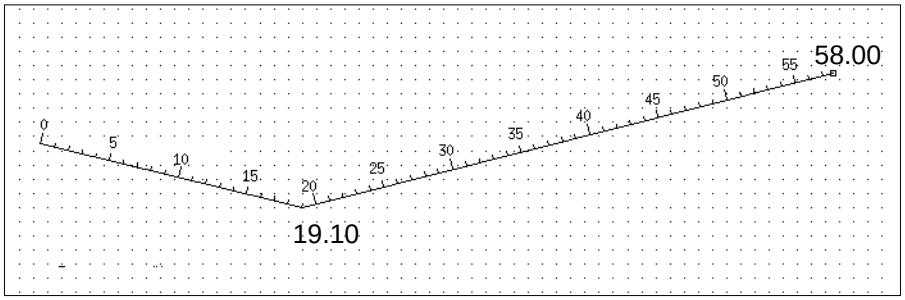
Ruler Functions

You can use a ruler to measure sizes of objects, distances between objects, and so forth. A ruler is not a database object; it is kept as a temporary data structure and is displayed in the window on top of the cellview where it is attached. This means that rulers cannot be stored with the cellview. Creating a ruler does not modify the database.

Rulers are represented by their centerline point-list, which can be non-orthogonal if desired. Ruler measurements are based on user units. User units are displayed as minor and major grid points, with a major grid point for every fifth minor grid point. The distance between grids scales on the screen based upon the window zoom factor of the window in which the ruler is being displayed.

As an option, rulers can be entered to give a cumulative measurement. The cumulative length of the ruler is displayed at each vertex. The total distance is displayed at the last point entered.

Typical Cumulative Ruler Display



leClearAllRuler

```
leClearAllRuler(
     d_cellViewId
     )
     => t/nil
```

Description

This procedure removes all rulers currently attached to cellview *d_cellViewId*.

Arguments

d_cellViewId	Database ID of the cellview containing the rulers.

Value Returned

t/nil	Returns t if the rulers are cleared; otherwise, it returns nil.

Example

```
leClearAllRuler( cell2 )
```

Clears all rulers in the cellview cell2 and returns t.

leCreateRuler

```
leCreateRuler(
     d_cellViewId
     l_points
     )
     => t/nil
```

Description

Creates a ruler in cellview *d_cellViewId* with vertices at the coordinates listed in *l_points*.

Arguments

d_cellViewId	Database ID of the cellview containing the ruler.
l_points	List of coordinates entered to create the ruler.

Value Returned

t/nil	Returns t if the ruler is created; otherwise, it returns nil.

Example

```
leCreateRuler( cell2 list(0:0 0:10 10:10) )
```

Returns t and displays a cumulative ruler in the cellview cell2. The measurement of the segment from 0:0 to 0:10 is displayed at 0:10, and the total length of the ruler is displayed at 10:10.

Interactive Function

```
leHiCreateRuler( w_windowId ) => t/nil
```

Type this function without arguments to create a ruler in the current cellview. The system prompts you to enter coordinates to place the ruler.

Search and Replace Functions

This section describes the functions used to locate specific types of objects in a design. These objects can be replaced with objects of a different type or with different criteria.

leReplace

```
leReplace(
     d_cellViewId
     l_objects
     l_replaceValues
     )
     => t/nil
```

Description

Replaces the specified values on the specified list of objects in the cellview *d_cellViewId*.

Arguments

d_cellViewId	Database ID for the cellview where the objects are located.
l_objects	List of objects to be modified.
l_replaceValues	List of properties or attributes whose values are replaced. The list format is the same as *l_criteriaList* for leSearchHierarchy except the operator (==, >=, <=) is ignored.

Value Returned

t/nil	Returns t if the values are replaced; otherwise, it returns nil.

Example

```
leReplace( cell2 list(path1 path2) list(list("layer" nil list("metal1" "drawing"))
list("path width" == 5.0)) )
```

Changes the layers of path1 and path2 to metal1 drawing and changes their widths to 5.0.

leReplaceAnyInstMaster

```
leReplaceAnyInstMaster(
     d_instId
     t_libName/nil
     t_cellName/nil
     t_viewName/nil
     )
     =>t/nil
```

Description

Changes the master library, cell, or view name for the instance. If you do not want to change any of these names, you must type `nil` in its place.

Arguments

d_instId	Database ID of the instance.
t_libName	New master library name for the instance.
t_cellName	New master cell name for the instance.
t_viewName	New master view name for the instance.

Value Returned

`t/nil`	Returns `t` if the master library, cell, or view name for the instance is changed; `nil` if it is not changed.

Example

```
foreach(fig geGetSelSet()
     if(fig~>objType == "inst"
          leReplaceAnyInstMaster(fig nil "Inv" nil)

          if(fig~>objType == "mosaic" && fig~>mosaicType == "simple"
               leReplaceAnyInstMaster(car(fig~>instanceList) nil "Inv" nil)
          )
     )
)
```

Changes the master cell name the instance to `Inv`.

leSearchHierarchy

```
leSearchHierarchy(
     d_cellViewId
     l_bBox
     x_stopLevel
     t_objectType
     l_criteriaList
     )
     => l_objects/nil
```

Description

Searches the cellview *d_cellViewId* for objects that match the type and criteria specified. The search is bounded by the specified *l_bBox* and restricted to the specified *x_stopLevel*. The search assumes all figures are valid if the Layer Selection Window (LSW) has not been initialized.

Arguments

d_cellViewId	Database ID for the cellview to search.
l_bBox	Bounding box within the cellview to search.
x_stopLevel	Specifies how deep in the hierarchy to search. Valid Values: any integer from `0` to `32`
t_objectType	Type of object for which to search. Valid Values: `inst`, `array`, `label`, `path`, `polygon`, `rectangle`, `contact`, `ellipse`, `donut`, `trl`, `bend`, `taper`, `any shape`, `any conic`, or `any microstrip`
l_criteriaList	List of criteria for which you want to search. You can search for one criterion or lists of several criteria, as shown in the example. Valid Values: a list containing *t_criteriaType, t_operator,* and *g_value*, as described below *t_criteriaType* Criteria for which you want to search. Valid Values: any valid attribute associated with the specific object, as shown in the table below, or any additional property you created for the object. For details about any object attributes, see the description of the SKILL command used to create that

object (for example, `leCreatePath`).

`t_operator`
Boolean operator you want to use in the search. It must be appropriate for the criteria type. For example, <= is appropriate in a search for a numeric value, but not appropriate in a search for a string.
Valid Values: ==, <, <=, >=, >, `!=`, `EXIST` (property exists), `!EXIST` (property does not exist)

`g_value`
Value of the criterion you want to find.
Valid Values: any value appropriate to this criteria type, as shown in the table below.

Object Type	Criteria Types
any conic	property, layer, any radius
any microstrip	property, layer
array	rows, columns, deltaX, deltaY, array name, library name
bend	property, layer, bend style
contact	property, contact type
donut	property, layer, inner radius, outer radius, any radius
ellipse	property, layer, radiusX, radiusY, any radius
instance	property, inst name, lib name, cell name, view name, mag, orient
label	property, layer, text, font, height, orient
path	property, layer, path style, path width
polygon	property, layer
rectangle	property, layer
taper	property, layer, taper style
trl [transmission line]	property, layer, bend style, chamfer fac, radius fac, trl width

Value Returned

l_objects/nil	Returns *l_objects*, a list of all objects that meet the object type and search criteria, if any are found; otherwise, it returns nil.

Example

```
leSearchHierarchy( cell2 list( 0:0 300:300 ) 0 "path"
list( list( "layer" "==" list( "poly1" "drawing" ) )
list( "path width" ">=" 3.0 ) ) )
```

Searches all of cell2 at the top-level for paths on poly1 drawing layer with path width >= 3.0.

Interactive Function

```
leHiSearch( [w_windowId] ) => t/nil
```

Enter this function with only the window ID argument; the system prompts you to specify the object type, criteria type, and where to search. If you do not specify *w_windowId*, the layout editor uses the current window.

Contact Functions

This section describes the functions used to query and change contacts and contact parameters.

leGetContactDefaultParam

```
leGetContactDefaultParam(
     d_cellViewId
     )
     => l_params/nil
```

Description

Returns a list of default parameters associated with a contact as defined in the technology file using `symContactDevice`. The list contains the following items: `width`, `length`, `rows`, `columns`, `xPitch`, `yPitch`, `xBias`, and `yBias`.

Arguments

d_cellViewId	Database ID for the supermaster of the contact.

Value Returned

l_params/nil	Returns a list of contact parameters if they are found; otherwise, it returns `nil`.

Example

```
leGetContactDefaultParam( cell2 )
```

Returns the list of default parameters for `cell2`.

leGetContactNameArray

```
leGetContactNameArray(
     d_techFileId
     )
     => l_contactNames/nil
```

Description

Returns a list of names of all defined contacts in the specified technology file.

Arguments

d_techFileId	Database ID of the technology file to search for defined contacts.

Value Returned

l_contactNames/nil	Returns a list of contact names if defined; otherwise, it returns nil.

Example

```
leGetContactNameArray( myTechFile )
```

Returns a list of names of all defined contacts in myTechFile.

leGetContactParam

```
leGetContactParam(
     d_instId
     )
     => l_params/nil
```

Description

Returns a list of the parameters associated with a contact instance. The list contains the following items: `width`, `length`, `rows`, `columns`, `xPitch`, `yPitch`, `xBias`, and `yBias`.

Arguments

`d_instId`	Database ID for the instance of the contact.

Value Returned

`l_params``/nil`	Returns a list of contact parameters if they are found; otherwise, it returns `nil`.

Example

```
leGetContactParam( cell2 )
```

Returns a list of the parameters associated with `cell2`.

leGetContactRule

```
leGetContactRule(
     d_cellViewId
     )
     => l_contactRules/nil
```

Description

Returns a list of the contact rules in the supermaster cellview of the contact. The list contains the following items: `viaLayer`, `layer1`, `layer2`, `encByLayer1`, `encByLayer2`, `layer1ImpLayer`, `layerImpEnc`, `layer2ImpLayer`, and `layer2ImpEnc`.

Arguments

d_cellViewId	Database ID for the supermaster of the contact.

Value Returned

l_contactRules/nil	Returns `t` if the contact rules in the supermaster are found; otherwise, it returns `nil`.

Example

```
leGetContactRule( cell2 )
```

Returns a list of the contact rules in `cell2`.

leIsAnyContact

```
leIsAnyContact(
     d_instId
     )
     => t/nil
```

Description

Tests if the specified instance is a contact.

Arguments

d_instId	The instance to be tested.

Value Returned

t/nil	Returns t if the specified instance is a contact; otherwise, it returns nil.

Example

```
leIsAnyContact(instId)
```

Checks if instId is a contact and returns either t or nil.

leIsAnyContactMaster

```
leIsAnyContactMaster(
     d_cellViewId
     )
     => t/nil
```

Description

Tests if the specified cellview is a contact.

Arguments

d_cellViewId	The cellview to be tested.

Value Returned

t/nil	Returns t if the specified cellview is a contact; otherwise, it returns nil.

Example

```
leIsAnyContactMaster(cellView)
```

Checks if cellView is a contact and returns either t or nil.

leIsContact

```
leIsContact(
     d_instId
     )
     => t/nil
```

Description

Checks if the specified instance is a contact.

Arguments

d_instId	Database ID for the instance to check.

Value Returned

t/nil	Returns t if the specified instance is a contact; otherwise, it returns nil.

Example

```
leIsContact( inst2 )
```

Returns t if inst2 is a contact.

leIsContactMaster

```
leIsContactMaster(
     d_cellViewId
     )
     => t/nil
```

Description

Checks if *d_cellViewId* is a supermaster of a contact.

Arguments

d_cellViewId	Database ID for the contact to be checked.

Value Returned

t/nil	Returns t if *d_cellviewId* is a contact supermaster; otherwise, it returns nil.

Example

```
leIsContactMaster( cell2 )
```

Returns t if cell2 is a contact supermaster.

leIsContactName

```
leIsContactName(
     d_techFileId
     t_contactName
     )
     => t/nil
```

Description

Checks if the name specified by *t_contactName* is a defined contact in the specified technology file.

Arguments

d_techFileId	Database ID for the technology file to search.
t_contactName	Contact name for which to search.

Value Returned

t/nil	Returns t if *t_contactName* is defined as a contact in the technology file; otherwise, it returns nil.

Example

```
leIsContactName( myTechFile "M2_M1" )
```

Returns t if M2_M1 is defined as a contact in myTechFile.

leSetContactParam

```
leSetContactParam(
     d_instId
     n_width
     n_length
     x_rows
     x_columns
     n_xPitch
     n_yPitch
     t_xBias
     t_yBias
     )
     => t/nil
```

Description

Updates all the pcell parameters of a particular contact instance.

Arguments

d_instId	Database ID of the contact instance.
n_width	Width of the contact. Valid Values: any number greater than zero
n_length	Length of the contact. Valid Values: any number greater than zero
x_rows	Number of rows in the contact array. Valid Values: any integer greater than zero
x_columns	Number of columns in the contact array. Valid Values: any integer greater than zero
n_xPitch	Centerpoint-to-centerpoint spacing in the X direction.
n_yPitch	Centerpoint-to-centerpoint spacing in the Y direction.
t_xBias	Specifies the location of the contact origin on the X axis. Valid Values: `left`, `center`, or `right`
t_yBias	Specifies the location of the contact origin on the X axis. Valid Values: `top`, `center`, or `bottom`

Value Returned

`t/nil`	Returns `t` if the parameters are updated; otherwise, it returns `nil`.

Example

```
leSetContactParam( inst2 2.0 2.0 4 4 4.0 4.0 "center" "center" )
```

Sets the parameters of the contact `inst2` with a width and length of two user units, four rows and columns, vertical and horizontal centerpoint to centerpoint spacing of four user units, and an origin at the center of the instance.

Hierarchy Traversal Functions

This section describes the functions that traverse the hierarchy to edit the master of an instance.

leDescend

```
leDescend(
     w_windowId
     d_instId
     [x_row]
     [x_column]
     )
     => w_windowId/nil
```

Description

Descends through the hierarchy to open the master of instance *d_instId*. leDescend attempts to open the cellview in the same mode as the current cellview, but if an editable version is not available, leDescend opens the master in read-only mode.

Arguments

w_windowId	Window ID of the window containing the instance.
d_instId	Database ID of the instance to descend into.
x_row	Row position if the instance is part of a mosaic.
x_column	Column position if the instance is part of a mosaic.

Value Returned

w_windowId/nil	Returns the window ID of the window containing the instance if the master instance cellview opens; otherwise, it returns nil. (The value for *w_windowId* is the same value provided in the argument *w_windowId* if leDescend is successful.)

Example

```
leDescend(hiGetCurrentWindow() myInst)
```

Descends into the master instance of myInst using the current layout editor editing window. The returned value is the window ID.

leEditInPlace

```
leEditInPlace(
     w_windowId
     l_hierlist
     )
     => t/nil
```

Description

Descends through the hierarchy to edit a cell in place.

Arguments

w_windowId	Window ID of the window to use.
l_hierlist	List describing a path for the instance to edit in place. This argument takes the form `list((`*instId memInst row col*`) (`*instId memInst row col*`) ...)` The *row* and *col* elements represent the row and column positions of the instance if it is part of a mosaic. *memInst* is an integer that is not used in `leEditInPlace`.

Value Returned

`t/nil`	Returns `t` if the cell is made editable in the specified window; otherwise, it returns `nil`.

Example

```
leEditInPlace (hiGetCurrentWindow() ((db:12082048 0 0 0) (db:13240512 0 1 2)
(db:13933796 0 0 0)))
```

Descends through an instance (`db:12082048`) at level one and an element of a mosaic (`db:13240512`) at level two to edit an instance (`db:13933796`) at level three.

Binary and Unary Functions

This section describes the functions used to create objects from the union, intersection, or nonintersection of existing objects.

leLayerAnd

```
leLayerAnd(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes/nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that correspond to the intersections of all shapes on layers *g_lpp1* and *g_lpp2*. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	Database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
g_lpp3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes/nil	Returns the IDs of the new shapes if new shapes are created; otherwise, it returns nil.

Example

```
leLayerAnd( cell1 list("poly1" "drawing") list("pdiff" "drawing")
list("activegate" "drawing") )
```

Creates shapes in cell1 on layer activegate from the intersection of all shapes on the poly1 and pdiff layers. Returns the shape IDs of the shapes.

leLayerAndNot

```
leLayerAndNot(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes/nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that correspond to the shapes on layer *g_lpp1* minus the intersections with the shapes on layer *g_lpp2*. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	Database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
g_lpp3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes/nil	Returns the new shapes if they are created; otherwise, it returns nil.

Example

```
leLayerAndNot( cell3 list("via" "drawing") list("metal2" "drawing")
list("opencontacts" "drawing") )
```

Creates shapes in cell3 on the layer opencontacts from shapes on the layer via minus the intersections with shapes on layer metal2. Returns the shape IDs of the shapes.

leLayerOr

```
leLayerOr(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes/nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that correspond to the union of shapes on layers *g_lpp1* and *g_lpp2*. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	Database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
g_lpp3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes/nil	Returns the IDs of the new shapes if they are created; otherwise, it returns `nil`.

Example

```
leLayerOr( (cell3 list("metal1" "drawing") list("metal2" "drawing")
list("allmetal" "drawing") )
```

Creates shapes in `cell3` on the layer `allmetal` from the union of the shapes on the `metal1` and `metal2` layers. Returns the shape IDs of the shapes.

leLayerSize

```
leLayerSize(
     d_cellViewId
     g_lpp1
     n_sizeAmount
     g_lpp2
     )
     => l_shapes/nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp2* by oversizing the shapes on layer *g_lpp1* by *n_sizeAmount* number of units, which can be either positive (oversize) or negative (undersize). A layer-purpose pair is either a layer name or a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	Database ID of the cellview containing the layer-purpose pairs.
g_lpp1	Layer from which the selected shapes are sized.
n_sizeAmount	Number of user units used to oversize or undersize the shapes.
g_lpp2	Layer on which new shapes are created.

Value Returned

l_shapes/nil	Returns the new shapes if they are created; otherwise, it returns nil.

Example

```
leLayerSize( cell2 list("gate" "drawing") 1.5 list("implant" "drawing") )
```

Creates shapes in cell2 on layer implant by oversizing the shapes on layer gate by 1.5 user units. Returns the shape IDs of the shapes.

leLayerXor

```
leLayerXor(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes/nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that are the result of the union of the shapes on layers *g_lpp1* and *g_lpp2* minus their intersection. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellviewId	Database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
g_lpp3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes/nil	Returns the new shapes if they are created; otherwise, it returns nil.

Example

```
leLayerXor( cell2 list("pdiff" "drawing") list("nwell" "drawing")
list("missingwell" "drawing") )
```

Creates shapes in cell2 on the layer missingwell. The new shapes are the result of the union of the shapes on the pdiff or nwell layers minus their intersections. Returns the shape IDs of the shapes.

Layer Purpose Icon List Function

This section describes a function used to create an icon list of layers for forms or menus created for layout editor.

hiMakeLPChoiceList

```
hiMakeLPChoiceList(
     d_techFileID
     l_layerName&Purpose
     )
     => icon list/nil
```

Description

Creates a layer purpose icon list for use in a form field or menu.

Arguments

d_techFileID	Technology File ID in which the layers are defined.
l_layerName&Purpose	List of lists identifying layers and their purposes.

Value Returned

`icon list/nil`	Returns `icon list` if successful; otherwise, it returns `nil`.

Example

```
iconList = hiMakeLPChoiceList(techFileId list(
          list("metal1" "drawing")
          list("metal2" "drawing")
          )
     )
```

Creates a layer purpose icon list with two layers, a metal1 drawing layer and a metal2 drawing layer.

```
iconList = hiMakeLPChoiceList(techFileId list())
```

Creates a layer purpose icon list containing all the layers of the specified technology file.

Bindkey Functions

This section describes functions used to customize bindkeys in layout editor. For complete information about bindkey functions, see *Cadence User Interface SKILL Functions Reference*.

cmdCtrlOption

```
cmdCtrlOption()
     => t/nil
```

Description

Implements a command option for layout editor creating and editing commands when you press the `Control` key and press the bindkey. The command options are embedded in the layout editor software and cannot be changed. The default bindkey for this function is the right mouse button. You can change the default to another key.

Commands and Command Options

Create Path - While creating a path, press `Control` and click the bindkey to change the layer on which the path is being created. You must have visible, valid, and selectable layers set and you must have selected a valid entry level for a path. Very helpful during path stitching.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the command executed successfully and `nil` if an error occurred.

Example

```
hiSetBindKey("Layout" "Ctrl<Key>a EF" "cmdCtrlOption()")
```

Sets the *Change Path Layer* command option for the *Create Path* command to the a key.

cmdOption

```
cmdOption()
     => t/nil
```

Description

Implements a command option for layout editor creating and editing commands when you press the bindkey. The command options are embedded in the layout editor software and cannot be changed. The default bindkey for this function is the right mouse button. You can change the default to another key.

Commands and Command Options

Chop - When you are in polygon or line mode of the *Chop* command, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

Create Path - When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

Create Pin - When creating a polygonal shape pin, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

Create Polygon - When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

Reshape - When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*. When you have finished entering the points, the bindkey toggles between the two shapes you can select.

Split - When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*. When you have finished entering the points, the bindkey toggles between the two shapes you can select.

Trl - When you are entering points of a transmission line, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

Yank - When you are in polygon mode, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the command executed successfully and `nil` if an error occurred.

Example

```
hiSetBindKey("Layout" "Ctrl<Key>a EF" "cmdOption()")
```

Sets the command options for layout editor creating and editing commands to the a key.

cmdShiftOption

```
cmdShiftOption()
     => t/nil
```

Description

Implements a command option for layout editor creating and editing commands when you press the `Shift` key and press the bindkey. The command options are embedded in the layout editor software and cannot be changed. The default bindkey for this function is the right mouse button. You can change the default to another key.

Commands and Command Options

Copy - When you are copying an item or items, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

Move - When you are moving an item or items, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

Create Instance - When you are creating an instance, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

Create Label - When you are creating a label, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis). The letters will display right side up and will not display backwards.

Create Contact - When you are creating polygonal shaped contacts or multiple rows and columns of contacts, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

Paste - When you are pasting an item or items, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

Pick from schematic - (Virtuoso XL only) When you are using the *Pick from schematic* command, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the command executed successfully and `nil` if an error occurred.

Example

```
hiSetBindKey("Layout" "Shift<Key>a EF" "cmdShiftOption()")
```

Sets the command options for layout editor creating and editing commands to the a key.

leSelBoxOrStretch

```
leSelBoxOrStretch(
     [w_windowId]
     )
     => t
```

Description

The layout editor binds this function to the mouse button DrawThru1 action. The default behavior for mouse button DrawThru1 is select by bBox. When you select using a bBox, rather than a point, the software checks what is under the selected area. If something under the initial point of the bBox is already selected, the software automatically calls the stretch command. If nothing under the cursor is selected the software selects the bBox you have drawn. If *w_windowId* is not specified, the current window is used. This function should not be changed.

Interactive Functions

This section describes the functions that use forms or menu commands. These are the interactive functions that are associated with the layout editor procedural functions.

hiLayerDispMainForm

```
hiLayerDispMainForm()
     => t
```

Description

Opens the Layer Purpose Pair Editor form. In IC 5.0 this form is replaced by two forms; the Display Resource Editor form and the Set Valid Layer form, accessed from the Edit menu on the LSW.

leCloseWindow

```
leCloseWindow()
     => t/nil
```

Description

Closes the current layout window. Similar to the `hiCloseWindow` function. To customize a bindkey you must use `leCloseWindow` because it is the function used in the callback of the *Window – Close* menu command.

leEditDesignProperties

```
leEditDesignProperties(
     [d_cellviewId]
     )
     => t/nil
```

Description

Opens the Edit Cellview Properties form, which lets you change the design properties in the cellview *d_cellViewId*. If *d_cellViewId* is not specified, the cellview in the current window is used.

You can view the properties and values listed under *Attribute*, but you cannot change them. You can add, modify, delete or change the properties and values listed under *Property*.

leHiAbout

```
leHiAbout()
     => t/nil
```

Description

Displays a window which shows the Cadence logo, Virtuoso trademark information, version number and copyright information.

leHiAddShapeToNet

```
leHiAddShapeToNet(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Add Shape To Net form, which lets you associate a shape with a net. If *w_windowId* is not specified, the current window is used.

You can add selected shapes to a specified net or based on the pins overlapping the selected shapes.

leHiAttach

```
leHiAttach(
     [w_windowId]
     )
     => t/nil
```

Description

Attaches a selected object in the specified window to another object in the window. The attached object is called the *child* object, and the object it is attached to is called the *parent* object. When you move, copy, or delete the parent, the child is also moved, copied, or deleted. If *w_windowId* is not specified, the current window is used.

leHiChop

```
leHiChop(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Chop form, which lets you remove parts of objects or break objects into multiple objects. If *w_windowId* is not specified, the current window is used.

You are prompted to create a shape in the cellview that the system uses as a cutter to determine where to remove part of the objects or where to break them into multiple objects.

Chop cannot be used on pins. Use *Reshape* or *Stretch* to change pin shapes.

leHiClearRuler

```
leHiClearRuler(
     [w_windowId]
     )
     => t/nil
```

Description

Clears the rulers that stay in the window when the *Keep Ruler* option is selected using `leHiCreateRuler`. If *`w_windowId`* is not specified, the current window is used.

leHiConvertShapeToPolygon

```
leHiConvertShapeToPolygon(
     [w_windowId]
     )
     => t/nil
```

Description

Converts a selected object in the specified window to a polygon. Converts rectangles, paths, donuts, circles, and ellipses using the number of conic sides specified using `leHiEditEditorOptions`. If *`w_windowId`* is not specified, the current window is used.

leHiCopy

```
leHiCopy(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Copy form, which lets you copy selected objects in the specified window. If *w_windowId* is not specified, the current window is used.

You are prompted to point to a reference point for the copied objects. You can change the layer, rotation, and snap mode when you copy the object. You can also create an array of the copied objects.

leHiCreateBend

```
leHiCreateBend(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Transmission Line Bend form, which lets you create a transmission line bend in the specified window. If *w_windowId* is not specified, the current window is used.

You can set the bend style (bend, chamfer, or radial), the factors for the bend style, and the resolution for the bend. You can also set the snap mode.

leHiCreateChoiceOfPin

```
leHiCreateChoiceOfPin(
     [w_windowId]
     )
     => t/nil
```

Description

Opens either the Create Symbolic Pin form or the Create Shape Pin form in *w_windowId*. The pin form is symbolic or shape depending upon the environment variable `pinsAreSymbolic`. If `pinsAreSymbolic` is set to `t`, `leHiCreateSymPin` is called; otherwise `leHiCreatePin` is called. If *w_windowId* is not specified, the current window is used.

leHiCreateCircle

```
leHiCreateCircle(
     [w_windowId]
     )
     => t/nil
```

Description

Creates a circle in the specified window. You are prompted to point to a location for the center of the circle and the outer edge of the circle. If *w_windowId* is not specified, the current window is used.

leHiCreateContact

```
leHiCreateContact(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Contact form, which lets you create a contact in the specified window. If *w_windowId* is not specified, the current window is used.

You are prompted to choose *Contact Type* and *Justification* and specify the size of the contact. You can also rotate the contact, mirror the contact over the X and Y axis, or create an array of contacts.

leHiCreateDonut

```
leHiCreateDonut(
     [w_windowId]
     )
     => t/nil
```

Description

Creates a donut in the specified window. You are prompted to point to a location for the center of the donut, the inner edge of the donut, and the outer edge of the donut. If *w_windowId* is not specified, the current window is used.

leHiCreateEllipse

```
leHiCreateEllipse(
     [w_windowId]
     )
     => t/nil
```

Description

Creates an ellipse in the specified window. You are prompted to point to the corners of the bounding box for the ellipse. If *w_windowId* is not specified, the current window is used.

leHiCreateInst

```
leHiCreateInst(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Instance form, which lets you create an instance in the specified window. If *w_windowId* is not specified, the current window is used.

You are prompted to specify the name of a library, a cell, and a view to use as the master cell for the instance. You can change the rotation of the instance or mirror the instance over the X or Y axis. You can also create an array of the instance. If the chosen cell is a pcell, its parameters are added to the form.

leHiCreateLabel

```
leHiCreateLabel(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Label form, which lets you create a label in the specified window. If *w_windowId* is not specified, the current window is used.

You can specify the text for the label, the height of the label, the font style, and the justification. You can choose options for *Drafting* and *Attach*. You can also rotate the label or mirror the label over the X or Y axis.

leHiCreatePath

```
leHiCreatePath(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Path form, which lets you create a path in the specified window. If *w_windowId* is not specified, the current window is used.

In manual mode you can set the width, offset, and justification of the path. You can set the snap mode, contact justification, end type, and begin and end extensions. You can also change the layer for the path.

When the layer is changed, an appropriate contact is placed at the intersection of the layers. If *Fixed Width* is on, the width remains the same. Otherwise, the `minWidth` property of the new layer determines the new width.

leHiCreatePin

```
leHiCreatePin(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Shape Pin form, which lets you create a geometric pin in the specified window. If *w_windowId* is not specified, the current window is used.

You can enter a name for the terminal, choose the *Mode*, choose the *I/O Type* and *Access Direction*, and set *Snap Mode*.

leHiCreatePinsFromLabels

```
leHiCreatePinsFromLabels(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Pins From Labels form, which lets you create pins from text labels in your layout cellview or selected instances. This command creates pins with terminal names matching labels on a specified text layer with pin dimensions that you specify, centered on the origin of your text label. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiCreatePolygon

```
leHiCreatePolygon(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Polygon form, which lets you create a polygon in the specified window. If *w_windowId* is not specified, the current window is used.

You are prompted to point to the coordinates for the vertices of the polygon. You can set the snap mode for the polygon and you can create polygons that contain arc segments.

leHiCreateRect

```
leHiCreateRect(
     [w_windowId]
     )
     => t/nil
```

Description

Creates a rectangle in the specified window. You are prompted to point to two coordinates for the opposite corners of the rectangle. If *w_windowId* is not specified, the current window is used.

leHiCreateRuler

```
leHiCreateRuler(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Ruler form, which lets you create a ruler in the specified window. If *w_windowId* is not specified, the current window is used.

You can choose options to keep the ruler in the window after you enter the last point and create a multisegment ruler that displays cumulative distance. You can also set the snap mode for the ruler.

leHiCreateSymDev

```
leHiCreateSymDev(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Device form, which lets you create a symbolic device in the specified window. The symbolic devices are defined in the technology file. If *w_windowId* is not specified, the current window is used.

You specify the class, type, and name to create a symbolic device. You can change the rotation of the instance or mirror the instance over the X or Y axis. You can also create an array of the devices.

leHiCreateSymPin

```
leHiCreateSymPin(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Symbolic Pin form, which lets you create a symbolic pin as in the specified window. The symbolic devices are defined in the technology file. If *w_windowId* is not specified, the current window is used.

You can enter a name for the terminal; choose the *Mode*, the *I/O Type*, *Pin Type*, *Pin Width*, and *Access Direction*; and set *Snap Mode*.

leHiCreateTaper

```
leHiCreateTaper(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Transmission Line Taper form, which lets you create a transmission line taper in the specified window. If *w_windowId* is not specified, the current window is used.

You can specify the *Taper Style*, *Resolution*, *Side Widths*, and *Snap Mode*.

leHiCreateTrl

```
leHiCreateTrl(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Create Transmission Line form, which lets you create a transmission line in the specified window. If *w_windowId* is not specified, the current window is used.

You can set the bend style (bend, chamfer, or radial), the factors for the bend style, the resolution for the bend, and the width of the transmission line. You can also set the snap mode.

leHiDelete

```
leHiDelete(
     [w_windowId]
     )
     => t/nil
```

Description

Deletes selected objects in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiDeleteAllAreaViewLevel

```
leHiDeleteAllAreaViewLevel(
     [w_windowId]
     )
     => t/nil
```

Description

Removes all special display areas in window *w_windowId*. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiDeleteAreaViewLevel

```
leHiDeleteAreaViewLevel(
     [w_windowId]
     )
     => t/nil
```

Description

Displays all currently defined area view levels using the layer-purpose pair `hilight drawing1`, and prompts you to point at the area to be removed. If you do not specify *`w_windowId`*, the layout editor uses the current window.

leHiDeleteShapeFromNet

```
leHiDeleteShapeFromNet(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Delete Shape From Net form, which lets you remove the selected shape from the net. If *w_windowId* is not specified, the current window is used.

You can delete selected shapes from the displayed net.

leHiDescend

```
leHiDescend(
     [w_windowId]
     )
```

Description

Lets you push down one level into the design hierarchy. You can preselect the instance to descend into. If no instances are selected, you are prompted to select an instance. If you do not specify *w_windowId*, the layout editor uses the current window.

If the master cell of the selected instance has more than one cellview available, the function must determine, which cellview to descend into. If the *Prompt For View Name* field is not set, or if there is only one view available, you descend into the cellview that has been instantiated. If more than one view is available, and the *Prompt For View Name* field is set, a form is displayed containing a cyclic field with all the possible view names listed. You can then select the cellview to descend into.

Automatically selects the edit mode to use when descending into the hierarchy. If the current cellview is opened in edit mode, the new cellview is opened in edit mode; if the new cellview cannot be opened in edit mode, it is opened in read mode. If the current cellview is opened in read mode, the new cellview will be opened in read mode.

leHiEditDisplayOptions

```
leHiEditDisplayOptions(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Display Options form, which lets you change the display options in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You can control the display of objects and the grid, how the cursor snaps to the grid, and the default snap mode settings for the create and edit forms.

These options correspond to window environment variables that can be initialized in the `.cdsenv` file or saved to the current cellview.

leHiEditEditorOptions

```
leHiEditEditorOptions(
     )
     => t/nil
```

Description

Opens the Layout Editor Options form, which lets you change the layout editor environment. You can also set these variables in your `.cdsenv` file, so the layout editor defaults to these options on start up.

You can control the behavior of various editing functions, the number of sides for conics, and the type of object the cursor snaps to. You can also set how close the cursor must be to an object before the cursor snaps (aperture), how many levels down in the hierarchy the cursor snaps (depth), and an offset distance for the cursor snap (bounce).

leHiEditInPlace

```
leHiEditInPlace(
     [w_windowId]
     )
     => t/nil
```

Description

Lets you edit an instance's master while viewing it in the context of the instance placement. If you do not specify *w_windowId*, the layout editor uses the current window. You can preselect the instance to be edited in place. If no instance is selected, you are prompted to point at a shape contained in the instance to be edited in place. If you point at several shapes at different levels of the hierarchy, the shape in the higher-level instance is the one that is used.

You can change the window to any level in the hierarchy. For example, if you are already using `leHiEditInPlace` to edit a cellview two levels deep in your hierarchy, you can use `leHiEditInPlace` again to move up or down the hierarchy.

leHiEditProp

```
leHiEditProp(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Edit Properties form, which lets you edit the properties set in the specified window. You are prompted to select objects whose properties you want to edit. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiFlatten

```
leHiFlatten(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Flatten form, which lets you explode a cell instance or array, moving the contents of the cell or array up one or more levels into the current level of the hierarchy. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiLayerGen

```
leHiLayerGen(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Layer Generation form, which lets you create new shapes by performing logical operations on selected objects in the specified layers. If you do not specify *w_windowId*, the layout editor uses the current window.

You choose the first layer of the operation, the type of operation (AND, OR, AND NOT, XOR, or GROW BY), the second layer of the operation, and the layer on which the new shapes are created.

leHiLayerTap

```
leHiLayerTap(
     [w_windowId]
     )
     => t/nil
```

Description

Sets the entry layer to the layer of an indicated shape in the current window. You are prompted to point to the shape to be *tapped*. You can point to shapes anywhere in the hierarchy. If the object you want to select is overlapped by other objects, you can cycle through the layers until the layer you want is displayed. If you do not point to any shapes that are on valid layers, the function calls `hiGetAttention` and prompts you again. If layers overlap, If you do not specify *`w_windowId`*, the layout editor uses the current window.

leHiMakeCell

```
leHiMakeCell(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Make Cell form, which lets you create a cell from the selected objects in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to enter a cell name for the new cell. You can choose to replace the selected objects with the newly created cell.

leHiMarkNet

```
leHiMarkNet(
     [w_windowId]
     )
```

Description

Highlights a net or nets in *w_windowId*. Corresponds to the *Connectivity – Mark Net* command in the cellview window menu. If *w_windowId* is not specified, the current window is used.

After entering the SKILL code, select the net or nets you want to hightlight.

leHiMerge

```
leHiMerge(
     [w_windowId]
     )
     => t/nil
```

Description

Merges selected objects that touch or overlap each other into one object. You can merge shapes of different types that are on the same layer. You cannot merge shapes on different layers. If you do not specify *w_windowId*, the layout editor uses the current window.

Note: Pins cannot be merged.

leHiModifyCorner

```
leHiModifyCorner(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Modify Corner form, which lets you modify corners of rectangles and polygons in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to choose the corners you want to modify. You can specify the type of corner to create, the radius or distance to use, and how many sides to use. The radius or distance used is half the length of the longest line segment adjacent to the corner or the value supplied, whichever is shorter.

leHiMousePopUp

```
leHiMousePopUp(
     )
     => t/nil
```

Description

Opens the layout editor Middle Mouse Button Pop Up form. It is context sensitive. Returns `t` if the current window is a layout window, returns `nil` if the current window is not a layout window.

leHiMove

```
leHiMove(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Move form, which lets you move an object in the specified window to another location or layer in the same window or in a different window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to choose the object you want to move. You can change the layer or snap mode or rotate or mirror the object.

leHiMoveOrigin

```
leHiMoveOrigin(
     [w_windowId]
     )
     => t/nil
```

Description

Enter this function with only the window ID argument; the system prompts you to point to the new origin. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiOptionLayer

```
leHiOptionLayer(
     )
     => t/nil
```

Description

Opens the LSW Option form. Returns `t` when you click OK, returns `nil` when you click Cancel.

leHiPaste

```
leHiPaste(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Paste form, which lets you place the contents of the yank buffer in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to specify where you want the objects placed. You can choose to rotate or mirror the object.

leHiPlotQueueStatus

```
leHiPlotQueueStatus(
     )
     => t/nil
```

Description

Opens the Plot Job Queue form, which shows the status of the plot job queue.

You can view the printer queue and remove jobs from the plot job queue.

leHiPropagateNets

```
leHiPropagateNets(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Propagate Nets form, which lets you promote net information from pins in selected blocks one level down in the hierarchy up to the top level where the router can recognize and route the nets. You use this command when you do not have a schematic for your layout cellview. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiReShape

```
leHiReShape(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Reshape form, which lets you modify the vertices of rectangles, polygons, paths, and transmission lines. You can add vertices, remove vertices, or modify vertex locations. The modified shape is stored in its most efficient form. For example, if a polygon is modified to have a rectangular shape, it is converted into a rectangle. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to choose the object you want to reshape. You can choose to use a rectangle or line as the reshape type, and you can choose the snap mode for creating the reshape type.

leHiRotate

```
leHiRotate(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Rotate form, which lets you change the orientation of geometric objects. If *w_windowId* is not specified, the current window is used.

You choose the snap-to and rotation angle. You can also rotate objects 90, 180, or 270 degrees by using the buttons.

leHiSearch

```
leHiSearch(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Search form, which lets you search the specified window for objects that match user-specified query requests. If you do not specify *w_windowId*, the layout editor uses the current window.

You select the type of object to search for, how deep in the hierarchy to search, and the criteria types to search. For more information about using this form, see the *Virtuoso Layout Editor Help.*

leHiSetAreaViewLevel

```
leHiSetAreaViewLevel(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Set Area View Level form, which lets you create a special display area in the window. This lets you override the window's display start and stop levels for a specified area. If you do not specify *w_windowId*, the layout editor uses the current window.

The system prompts you to create the coordinates for the bounding box and type in the start and stop levels.

leHiSetRefPoint

```
leHiSetRefPoint(
     [w_windowId]
     )
     => t/nil
```

Description

Sets a reference point in the specified window. You are prompted to point to location for the reference point. If you do not specify *w_windowId*, the layout editor uses the current window.

leHiSetValidLayer

```
leHiSetValidLayer(
     )
     => t/nil
```

Description

Opens the Set Valid Layer form, which lets you set which layers in your library are displayed in the Layer Selection Window.

leHiShowSelSet

```
leHiShowSelSet(
     [w_windowId]
     )
     => t/nil
```

Description

Displays information about the selected set in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

The Show Selected Set list displays information about all objects in the selected set: the type of shape, the layer and purpose, and the coordinates used to create the object. If instances or contacts are selected, the list displays the master name, instance name and type, orientation, and origin.

leHiSize

```
leHiSize(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Size form, which lets you enlarge or reduce a shape in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to enter a value for the size. A positive number enlarges an object, and a negative number reduces an object.

leHiSplit

```
leHiSplit(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Split form, which lets you reshape an object by stretching part of it relative to a split line in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to create the split line and then stretch the newly created edges.

leHiStretch

```
leHiStretch(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Stretch form, which lets you stretch objects in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to select an object to stretch. You can change the snap mode or lock angles when you stretch an object.

leHiSubmitPlot

```
leHiSubmitPlot(
     )
     => t/nil
```

Description

Opens the Submit Plot form, which lets you submit plots to a plotter.

You can submit plots to a plotter or file and specify the area to plot, when to plot, how big to make the plot, and how objects appear in the plot.

leHiSummary

```
leHiSummary(
     [w_windowId]
     )
     => t/nil
```

Description

Displays information about the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

The summary lists information about the environment; the number and type of objects created in the cellview; and the number, name, and type of instances placed. If the cellview contains contacts, the summary lists the number, name, and type of contacts.

leHiTree

```
leHiTree(
     [w_windowId]
     )
     => t/nil
```

Description

Displays information about the design hierarchy in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You can choose the order of the display (top to bottom, current to bottom, or top to current) from the Tree form. The default is *Current to bottom*. The design hierarchy shows all placed instances in the window, listing the library, cell, and view names and the number of occurrences of each cell.

leHiYank

```
leHiYank(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Yank form, which lets you copy a set of objects in a cellview into the yank buffer for later pasting. A yank shape defines the area enclosing the objects to copy. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to enter the coordinates of the yank shape.

leiDiscardEdits

```
leiDiscardEdits()
     => t/nil
```

Description

Discards edits. Corresponds to the cellview window *Design – Discard Edits* command.

A dialog box will appear asking if you really want to discard edits.

2

Parameterized Cell Functions

This chapter provides information about creating parameterized cells (pcells) using Cadence® SKILL language instead of using the graphical user interface. It includes safety rules for creating SKILL pcells, information about pcell master cells and submaster cells, and descriptions of supported pc functions.

Most supported pc SKILL functions correspond to the commands available through the graphical user interface; you can use them to create pcells for the graphical pcell environment. However, there are only four pcell functions that you can use within the body of SKILL pcell code.

Note: We recommend using Virtuoso® relative object design (ROD) functions in your pcell code. ROD is a high-level language for easily defining layout connectivity, geometries, and the relationships between them. For information about ROD, see the *Virtuoso Relative Object Design User Guide*.

This section contains the following information:

Safety Rules for Creating SKILL Pcells on page 278

- Calling SKILL Procedures from within Pcells on page 279
- Recommended, Supported SKILL Functions for Pcells on page 279
- Finding Supported SKILL Functions on page 280
- Using print Functions in a Pcell on page 281
- Enclosing the Body of Code in a let or prog for Local Variables on page 282
- What to Avoid When Creating Pcells on page 282

About Pcell Super and Submaster Cells on page 283

- Using the SKILL Operator ~> with Pcells on page 284

pc Functions for SKILL Pcell Code on page 285

- pcExprToString on page 285

pcFix on page 286

pcRound on page 288

pcTechFile on page 289

Parameterized Cell SKILL Cross-Reference Table on page 290

Graphic Parameterized Cell SKILL Functions on page 294

pcDefineCondition on page 295

pcDefineInheritParam on page 297

pcDefineParamLabel on page 298

pcDefineParamLayer on page 299

pcDefineParamPath on page 301

pcDefineParamPolygon on page 303

pcDefineParamProp on page 305

pcDefineParamRect on page 306

pcDefineParamRefPointObject on page 307

pcDefinePathRefPointObject on page 308

pcDefinePCell on page 309

pcDefineRepeat on page 312

pcDefineSteppedObject on page 315

pcDefineStretchLine on page 317

pcDeleteCondition on page 319

pcDeleteParamLayer on page 320

pcDeleteParamProp on page 321

pcDeleteParamShape on page 322

pcDeleteRefPoint on page 323

pcDeleteRepeat on page 324

pcDeleteSteppedObject on page 325

pcGetConditions on page 326

pcGetInheritParamDefn on page 327

pcGetInheritParams on page 328

pcGetOffsetPath on page 329

pcGetOffsetPolygon on page 331

pcGetParameters on page 333

pcGetParamLabelDefn on page 334

pcGetParamLabels on page 335

pcGetParamLayers on page 336

pcGetParamLayerDefn on page 337

pcGetParamProps on page 338

pcGetParamShapeDefn on page 339

pcGetParamShapes on page 340

pcGetRefPointDefn on page 341

pcGetRefPoints on page 342

pcGetRepeatDefn on page 343

pcGetRepeats on page 344

pcGetSteppedObjectDefn on page 345

pcGetSteppedObjects on page 346

pcGetStretchDefn on page 347

pcGetStretches on page 348

pcGetStretchSummary on page 349

pcModifyParam on page 391

pcRedefineStretchLine on page 392

pcRestrictStretchToObjects on page 394

pcSkillGen on page 395

auHiUltraPCell on page 397

Pcell Compiler Customization SKILL Functions on page 399

pcUserAdjustParameters on page 400

pcUserGenerateArray on page 401

pcUserGenerateInstance on page 402

pcUserGenerateInstancesOfMaster on page 403

pcUserGenerateLPP on page 405

pcUserGeneratePin on page 406

pcUserGenerateProperty on page 407

pcUserGenerateShape on page 408

pcUserGenerateTerminal on page 409

pcUserInitRepeat on page 410

pcUserPostProcessCellView on page 412

pcUserPostProcessObject on page 413

pcUserPreProcessCellView on page 414

pcUserSetTermNetName on page 415

auHiUltraPCell on page 397

Safety Rules for Creating SKILL Pcells

This section provides important rules for creating SKILL pcells, including which functions are safe to use and what to avoid.

Caution

If you use SKILL functions that are unsupported and/or not intended for use in pcells, your pcell code will probably fail when you try to translate your design to a format other than Design Framework II or when you use the pcell in a different Cadence application.

The purpose for creating a pcell is to *automate the creation of data*. Pcells should be designed as standalone entities, independent of the environment in which they are created and independent of the variety of environments in which you or someone else might want to

use them. An environment can react to a pcell, but pcell code should not react to, interact with, or be dependent on an environment.

Although it is possible to create pcells dependent on something in your current or local environment, and/or using unsupported or unrecommended functions, your pcell code is likely to fail when you try to translate it for a different environment. And although it is possible to load, read, and write to files in the UNIX file system from within a pcell, do not do so. You cannot control the file permission settings in other locations, so reading or writing from a pcell can cause the pcell to fail in other directories, other environments, and during evaluation by translators.

Functions that are not supported for use by customers within SKILL pcells usually belong to specific applications (tools); they are unknown to other environments, to other tools, and to data translators. For example, if you create a pcell in the `icfb` environment and include place-and-route functions, the pcell will fail in the layout environment. Also, application-specific functions that are not supported for customer use can disappear or change, without notice.

Create pcells using only the recommended, supported functions. Generally, you can identify them by their prefixes. However, you can also use all of the basic SKILL language functions defined in the *SKILL Language Reference Manual*; these functions do not have prefixes. To use only recommended, supported, documented functions, follow the rules in this section.

Calling SKILL Procedures from within Pcells

You can call your own SKILL procedures from within pcells. However, the code in procedures called by pcells must follow all of the safety rules described in this section. The safety rules apply to all code that is evaluated when the system evaluates a pcell, and that includes SKILL procedures called by the pcell.

Called procedures are not compiled with the pcell, so you must make sure they are loaded before the system evaluates the pcell. You can do this by attaching the code files to your library; then the system automatically loads the procedures the first time it accesses the library.

For information about how to attach code files to a library, see “Calling Your Own Procedures Within Pcells” in the *Virtuoso Parameterized Cell Reference*.

Recommended, Supported SKILL Functions for Pcells

When you create SKILL routines within pcells, use only the following functions:

- The SKILL functions documented in the *SKILL Language Reference Manual*; for example, `car`, `if`, `foreach`, `sprintf`, `while`.
- SKILL functions from the following families:

  `db` `dd` `cdf`
  `rod` `tech`

- The following four `pc` SKILL functions, which are documented in this chapter:

  `pcExprToString` `pcFix`

  `pcRound` `pcTechFile`

  You can use only these four `pc` SKILL functions because

  - Most supported `pc` functions correspond to the graphical user interface commands. You can use them to create pcells in the graphical pcell environment, but you cannot use them in the body of SKILL pcell code.
  - Both the Pcell graphical user interface and the Pcell compiler are coded using `pc` SKILL functions. You cannot use the graphical user interface or compiler functions at all.

  For a description of the four `pc` SKILL functions you can use in the body of pcell code, see "pc Functions for SKILL Pcell Code" on page 285.

Finding Supported SKILL Functions

You can use either the `listFunctions` command or the Cadence Finder quick reference tool to list supported functions. But remember, you cannot use most supported functions within your pcell code.

Note: In releases prior to 4.4.3, the `listFunctions` command displayed a list of all SKILL functions. Starting with release 4.4.3, the `listFunctions` command displays only public SKILL functions. *Public SKILL functions* are supported by Cadence for use by customers. If you are using a software release prior to 4.4.3 and use the `listFunctions` command to display pcell functions, the list returned is misleading because it includes `pc` functions used to code the graphical user interface and compiler.

You can use the Finder tool to quickly check which SKILL functions are documented and supported for the `db`, `dd`, `cdf`, `rod`, and `tech` SKILL families.

Using the Finder Tool

To use the Finder tool, follow the steps below.

1. Start the Finder by doing one of the following:
   - From within the Cadence Design Framework II product (DFII), select *Tools–SKILL Development* and click on *Finder*.
   - In a UNIX window, type `cdsFinder`.
2. To display the supported SKILL functions,
   - For *Searching*, select *All Available Finder Data*.
   - For *Search String*, click on *at beginning*.
   - In the data entry area below *Search String*, type the function prefix. For example, type `tech`.
   - Click on *Search*.

     All of the supported, documented SKILL functions for the prefix you entered appear in alphabetical order in the *Matches* field.
3. To see the arguments and a short description for a function that is listed in the *Matches* field, click on the function name.

   The syntax and a brief description of the selected function appear in the Descriptions window. You might need to scroll up to see the syntax.
4. If necessary, scroll up to see the syntax.

   For functions that have keyword-value pairs for arguments, each argument probably wraps to the next line.
5. If the arguments are wrapping, make the Finder window wider by stretching the lower-left corner.
6. Before selecting another function, click on the *Clear* button.

Using print Functions in a Pcell

Generally, you cannot print or generate output from a pcell, as it causes the pcell evaluation to fail. However, you can bypass the system processing for print functions by using the `fprintf` or `println` function to print directly to `stdout` (standard output) in the format shown below:

```
fprintf( stdout ... )
```

```
println( variable stdout )
```

where *variable* is your variable name. For example

```
fprintf( stdout "myVariable = %L \n" myVariable )
```

Using printf or println in a Pcell

If you use the `printf` or `println` function in a pcell, the output is treated as an error. In a cellview, errors are indicated by flashing markers. You can query the error in the cellview, change your pcell code to fix the error, and recompile your pcell.

To query markers, do the following:

1. Choose Verify – Markers – Explain.
2. Click on a flashing marker.

   A dialog box appears, showing the text printed from the pcell by the `printf` or `println` function.

Enclosing the Body of Code in a let or prog for Local Variables

When use local variables in the body of SKILL pcell code in a `pcDefinePCell` statement, be sure to enclose the pcell code in a `let` or `prog` statement, and define all variables used in the pcell code at the beginning of the `let` or `prog` statement. Defining variables as part of a `let` or `prog` prevents conflicts with variables used by the Pcell compiler. Using `let` gives faster performance than `prog`; `prog` allows multiple exits while `let` exits only at its end.

What to Avoid When Creating Pcells

When creating a pcell, do not use functions with prefixes other than `db`, `dd`, `cdf`, `rod`, and `tech`. Although it is possible to call procedures with other prefixes from within a pcell, you will not be able to view the pcell in a different environment or translate it into the format required by another database. Most translators, including the physical design translators, can translate only basic SKILL functions; the four `pc` functions; and functions in the SKILL families `db`, `dd`, `cdf`, `rod`, and `tech`.

Specifically, **do not** use functions with application-specific prefixes such as `ael`, `abs`, `de`, `ge`, `hi`, `las`, `le`, `pr`, and `sch`. Procedures in any application-specific family are at a higher level of functionality than can be used in a pcell safely. For example, if you create a pcell using `le` functions, the pcell works only in environments that include the Virtuoso® layout editor. You cannot use a translator to export your design from the DFII database.

When a translator cannot evaluate a function, one of the following happens:

- The translator fails and issues an `undefined function` error message.
- The translator continues, issues a warning message, and translates the data incorrectly.

To make your design translate successfully, you must resolve undefined functions in pcells. You can do this by flattening the pcells, which results in a loss of hierarchy and parameterization, or by rewriting the pcell code to eliminate the undefined functions.

Important

Here are more important rules for creating pcells. In your pcell code,

- Do not prompt the user for input.
- Do not generate messages; message output is interpreted as an error.
- Do not load, read, or write to files in the UNIX file system.
- Do not run any external program that starts another process.
- If you need to drive external programs to calculate cell shapes, you can do this in SKILL. Use CDF callback procedures to save the resulting list of coordinate pairs in a string, and then pass the string as input to a SKILL pcell.

  This method has the advantage that the external program needs to be called only once per instance, not each time the design is opened. For more information about callback procedures, refer to "Using the Component Description Format" in the *Virtuoso Parameterized Cell Reference*.
- Generally, do not generate output with `print`functions, except as described in "Using print Functions in a Pcell" on page 281.

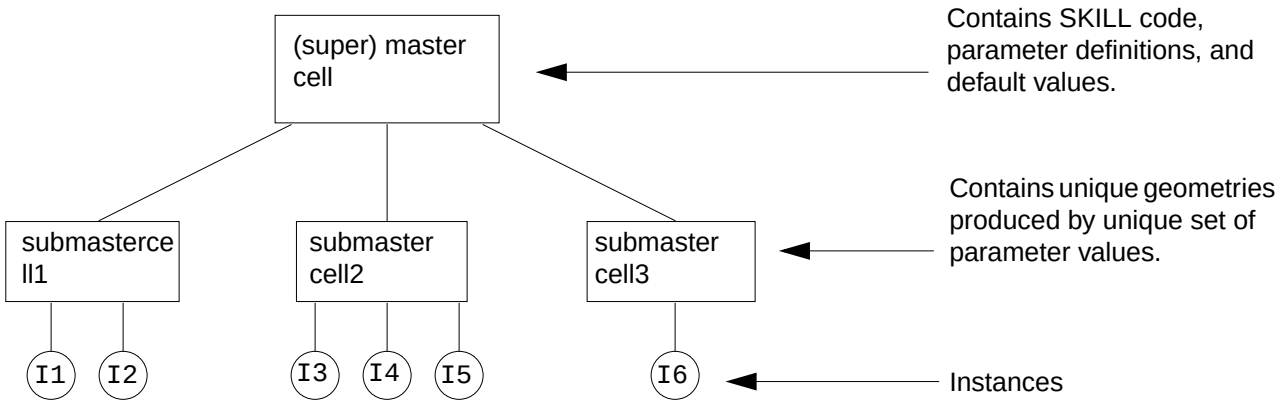
For more information about creating SKILL pcells, also see

- The "Creating Pcells Using SKILL" section in the *Virtuoso Parameterized Cell Reference*
- How to use relative object design functions (with the prefix `rod`) to create geometry for pcells in the *Virtuoso Relative Object Design User Guide*

About Pcell Super and Submaster Cells

The master cell of a pcell has two levels: a (super) master cell in the database and one or more temporary submaster cells in virtual memory. The Parameterized Cell software creates

one submaster cell in virtual memory for each unique set of parameter values assigned to an instance of the (super) master cell.



A submaster cell resides in virtual memory for the duration of your editing session and is accessible by all cellviews. When you create an instance in a cellview, and a submaster with the same parameter values already exists in virtual memory, no new submaster is generated; the new instance references the existing submaster cell.

When you close all cellviews that reference a particular submaster cell, the database software automatically purges (removes) the referenced submaster cell from virtual memory but does not remove the master pcell from your disk.

When you edit an editing session by exiting the Cadence software, the database software automatically purges (removes) all submaster cells from virtual memory. Purging does not remove master pcells from your disk.

Using the SKILL Operator ~> with Pcells

When you use the SKILL operator ~> to access information about the master cell of a pcell instance, you must look two levels above the instance. If you look at only one level above, you access information about the pcell submaster.

For example, for the instance `I1` of the pcell master `mux2`, you retrieve the database identity (`dbId`) of the `mux2` pcell master with the following statement:

```
I1~>master~>superMaster
```

But if you use the statement

```
I1~>master
```

you retrieve the *`dbId`* for the pcell submaster.

pc Functions for SKILL Pcell Code

When you create SKILL routines within pcells, use only the following pc functions:

pcExprToString	pcFix
pcRound	pcTechFile

pcExprToString

```
pcExprToString( g_ilExpr )
     => t_string
```

Description

Converts a SKILL expression to a string. The Pcell compiler uses this function to create labels that display the value of an expression as a string enclosed in quotes.

Arguments

g_ilExpr	SKILL expression you want to convert to a string.

Value Returned

t_string	The string equivalent of the SKILL expression, enclosed in quotes. If the value of the expression is `nil` or an error occurred, an empty string is returned.

Example

For the following expression, the system retrieves the value of the minimum width rule for the `metal1` layer from the technology file and then converts that value to a string enclosed in quotes.

```
pcExprToString( techGetSpacingRule( tfId "minWidth" "metal1" ) )
```

When the minimum width for `metal1` is equal to 0.6, the `pcExprToString` function returns the following string:

```
"0.6"
```

pcFix

```
pcFix( n_num )
     => x_result
```

Description

Converts a number to a floating-point number in the format *number*.0. When *n_num* is *very close* to a whole number, the system keeps the integer part of the number and adds a single decimal place equal to zero. *Very close* means the value of the number is within the range of plus or minus 0.001 of the integer part of the number specified by *n_num*. When the value is not within this range, the function allows the system to use the value in the first decimal place to round the *n_num* to an integer in the format *number*.0; the system ignores all other decimal places. This function is useful for correcting the round-off approximation that can occur with floating-point numbers that are stored in 32 or 64 bits.

For example, if the condition you want to test is `x = 15.0`, use the `pcFix` function to ensure that results in the range from 14.999 to 15.999 are converted to 15.0. This implies the following:

When *x* equals	`pcFix( x )` returns
14.998	14.0
14.999	15.0
15.0	15.0
15	15.0
15.998	15.0
15.999	16.0

Arguments

n_num	Any number you want to convert.

Value Returned

x_result	A number with one decimal place that is equal to zero. If *n_num* does not contain a number, an error occurs; look in the CIW or at the `CDS.log` file for a message about the error.

Example

If you want to test for the condition *x equals 6.0*, you need to use `pcFix` to correct a possible rounding problem that could occur with a floating-point number. Otherwise, your condition might never be satisfied, because x might never quite equal 6.0; instead, it might be a very close value such as 5.999999.

The following code shows how to use `pcFix` to correct floating-point rounding errors:

```
if( pcFix( x ) == 6.0
    ; perform conditional operations here
) ;endif
```

If you used the code shown below instead, your condition might never be satisfied:

```
x = 6.0
if ( x == 6.0
    ; perform conditional operations here
) ;endif
```

pcRound

```
pcRound( n_num )
     => x_result
```

Description

Lets you round a number to the closest integer, using the value of the first decimal place; additional decimal places are ignored. If the value of the first decimal place is less than 5, the system drops it; if the value of the first decimal place is greater than or equal to 5, the system drops it and adds one to the integer.

Arguments

n_num	Any number you want to round.

Value Returned

x_result	An integer. If *n_num* does not contain a number, an error occurs; look in the CIW or at the `CDS.log` file for a message about the error.

Example

The examples below show numbers rounded using `pcRound`.

```
pcRound( 1.49 )  results in  1
pcRound( -1.49)  results in  -1
pcRound( 1.59 )  results in  2.0
pcRound( -1.59 ) results in  2.0
```

pcTechFile

```
pcTechFile( g_expression )
     => g_result
```

Description

Evaluates an expression contained in a string. The Pcell compiler uses this function as an envelope around stretch expressions that access information from a technology file. This function prevents any symbols used in the technology file access expression from being defined as parameters of the pcell.

Arguments

g_expression	The expression you want the system to evaluate when the pcell containing the expression is evaluated.

Value Returned

g_result	The value that resulted from evaluating the expression. If an error occurs, look in the CIW or at the `CDS.log` file for a message about the error.

Example

In the following statement, the symbol `minWidth` is used only during evaluation of the expression; it is not treated as a pcell parameter:

```
pcTechFile( llp~>minWidth )
```

where `llp` is an internal variable, ~> is an operator, and `minWidth` is recognized by the system as a symbol.

If you did not use the `pcTechFile` function, the system would assume that `minWidth` is a pcell parameter.

Parameterized Cell SKILL Cross-Reference Table

This table summarizes the procedural and interactive pc SKILL functions associated with Parameterized Cell Compiler (Pcell) menu commands. A complete listing of the syntax, descriptions, and examples for the Pcell SKILL functions follows this table.

Interactive SKILL Function	Procedural SKILL Function	Menu Command
pcHIDefineParamCell	pcDefinePCell	*Compile – To Pcell*
pcHICompileToSkill	pcSkillGen	*Compile – To SKILL File*
pcHIDefineCondition	pcDefineCondition pcGetConditions pcGetParameters	*Conditional Inclusion – Define*
pcHIDeleteCondition	pcDeleteCondition pcDefineCondition pcGetConditions	*Conditional Inclusion – Delete*
pcHIModifyCondition	No procedural function	*Conditional Inclusion – Modify*
pcHIDisplayCondition	No procedural function	*Conditional Inclusion – Show*
pcHIDefineInheritedParameter	pcDefineInheritParam pcGetInheritParamDefn pcGetParameters	*Inherited Parameters – Define/Modify*
pcHIDisplayInheritedParameter	No procedural SKILL function	*Inherited Parameters – Show*
pcHIDefineLabel	pcDefineParamLabel pcGetParameters pcGetParamLabelDefn pcGetParamLabels	*Parameterized Label – Define*
pcHIModifyLabel	No procedural function	*Parameterized Label – Modify*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
pcHIDefineLayer	pcDefineParamLayer pcGetParameters pcGetParamLayers pcGetParamLayerDefn	*Parameterized Layer – Define*
pcHIDeleteLayer	pcDeleteParamLayer pcDefineParamLayer	*Parameterized Layer – Delete*
pcHIModifyLayer	No procedural function	*Parameterized Layer – Modify*
pcHIDisplayLayer	No procedural function	*Parameterized Layer – Show*
pcHIDefineProp	pcDefineParamProp pcGetParameters pcGetParamProps	*Parameterized Property – Define/Modify*
pcHIDeleteProp	pcDeleteParamProp	*Parameterized Property – Delete*
pcHIDisplayProp	No procedural function	*Parameterized Property – Show*
pcHIDefineParameterizedShape	pcDefineParamPolygon pcDefineParamPath pcDefineParamRect pcGetParameters pcGetParamShapeDefn pcGetParamShapes	*Parameterized Shapes – Define/Modify*
pcHIDeleteParameterizedShape	pcDeleteParamShape	*Parameterized Shapes – Delete*
pcHIDisplayParameterizedShape	No procedural function	*Parameterized Shapes – Show*
pcHIEditParameters	pcModifyParam	*Parameters – Edit Parameters*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
pcHIModifyParams	pcModifyParam	*Parameters – Edit Parameters*
pcHIDisplayParams	No procedural function	*Parameters – Show*
pcHISummarizeParams	No procedural function	Parameters – *Summarize*
pcHIDefineParamRefPointObject	pcDefineParamRefPointObject pcGetParameters pcGetRefPointDefn pcGetRefPoints	*Reference Point – Define by Parameter*
pcHIDefinePathRefPointObject	pcDefinePathRefPointObject pcGetParameters pcGetRefPointDefn pcGetRefPoints	*Reference Point – Define by Path Endpoint*
pcHIDeleteRefPointObject	pcDeleteRefPoint pcDefinePathRefPointObject pcDefineParamRefPointObject	*Reference Point – Delete*
pcHIModifyRefPointObject	No procedural function	*Reference Point – Modify*
pcHIDisplayRefPointObject	No procedural function	*Reference Point – Show*
pcHIDeleteRepeat	pcDeleteRepeat pcDefineRepeat pcGetRepeats	*Repetition – Delete*
pcHIModifyRepeat	No procedural function	*Repetition – Modify*
pcHIDefineRepeat	pcDefineRepeat pcGetParameters pcGetRepeatDefn pcGetRepeats	*Repetition* *– Repeat in X* *– Repeat in Y* *– Repeat in X and Y*
pcHIDisplayRepeat	No procedural function	*Repetition – Show*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
pcHIDefineSteppedObject	pcDefineSteppedObject pcGetParameters pcGetSteppedObjectDefn pcGetSteppedObjects	*Repetition Along Shape – Define*
pcHIDeleteSteppedObject	pcDeleteSteppedObject pcDefineSteppedObject	*Repetition Along Shape – Delete*
pcHIModifySteppedObject	No procedural function	*Repetition Along Shape – Modify*
pcHIDisplaySteppedObject	No procedural function	*Repetition Along Shape – Show*
pcHIModifyStretchLine	pcRedefineStretchLine	*Stretch – Modify*
pcHIQualifyStretchLine	pcRestrictStretchToObjects	*Stretch – Qualify*
pcHIRedefineStretchLine	No procedural function	*Stretch – Redefine*
	pcGetStretchSummary	*Parameters – Summarize*
pcHIDefineStretch	pcDefineStretchLine pcGetParameters pcGetStretchDefn pcGetStretches pcRedefineStretchLine pcGetOffsetPath pcGetOffsetPolygon	*Stretch* *– Stretch in X* *– Stretch in Y*
—	—	*Connectivity – Add Shape from Net*
—	—	*Connectivity – Delete Shape from Net*
auHiUltraPCell	No procedural function	*Ultra Pcell*

Graphic Parameterized Cell SKILL Functions

The following section describes each of the supported pc functions you can use to create pcells in the graphical pcell environment. Do not use the functions in this section within the body of SKILL pcell code.

pcDefineCondition

```
pcDefineCondition(
     d_cvId
     l_figs
     l_namelist
     g_condition
     g_stretch
     f_adjust
     )
     => d_condId/nil
```

Description

Specifies that the conditional inclusion of a list of objects is controlled by a given SKILL expression. Also specifies the inclusion of a dependent stretch control line. If the SKILL expression evaluates to a value other than `nil`, the objects are included.

Arguments

d_cvId	Database ID of the cellview in which the conditional inclusion is defined.
l_figs	List of objects controlled by the same conditional expression.
l_namelist	List of symbols referenced in the conditional expression. These become parameters of the pcell.
g_condition	SKILL expression controlling the inclusion of the selected objects in the instance.
g_stretch	SKILL symbol specifying the name of a dependent stretch control line associated with this conditional inclusion. Use `nil` if there is no dependent stretch control line.
f_adjust	Amount by which a dependent stretch is adjusted from its reference dimension if *g_condition* evaluates to `nil`. This argument is ignored if *g_stretch* is `nil`.

Value Returned

d_condId	Group ID used to store the details of the conditional inclusion.

`nil`	Returned if *`d_cvId`* does not identify a cellview or *`l_figs`* does not contain any objects.

Example

```
pcDefineCondition(cv figs list('Q_bar 'Reset) "Q_bar && Reset" "condStretch" 12.25)
```

Defines a conditional expression `Q_bar && Reset` in cellview *`d_cvId`* that controls the inclusion of the objects in *`l_figs`*. It declares `Q_bar` and `Reset` as symbols used as parameters for the pcell. It also defines the stretch control line `condStretch` as dependent on this conditional inclusion, with the stretch value decreased 12.25 from the reference dimension if `Q_bar && Reset` evaluates to `nil`.

pcDefineInheritParam

```
pcDefineInheritParam(
     d_instId
     s_parameter
     g_value
     l_namelist
     )
     => d_inheritGroup/nil
```

Description

Specifies that a parameter of an instance of a pcell takes its value from a parameter definition of the enclosing cellview. The selection filter allows you to select only instances.

Arguments

d_instId	Database ID of an instance of a pcell.
s_parameter	Name of a parameter in the master cellview of *d_instID*.
g_value	SKILL expression for the value of *s_parameter* in *d_instID*.
l_namelist	List of symbols used in *g_value*. These become parameters of the pcell.

Value Returned

d_inheritGroup	Group ID used to record the details of the inherited parameter.
nil	Returned if *d_instID* does not identify an instance or if there are no inherited parameters defined in it.

Example

```
nfetWidthInheritance = pcDefineInheritParam(nfet1 'w 'nfetWidth list('nfetWidth))
```

Causes parameter w on instance nfet1 to take its value from the nfetWidth parameter of the enclosing cellview. It declares nfetWidth as a parameter of the enclosing cellview.

pcDefineParamLabel

```
pcDefineParamLabel(
     d_labelId
     S_height
     )
     => t/nil
```

Description

Defines a parameterized label. A parameterized label is not an instance name, but a label displaying values within a pcell, such as width and height of gates. This function should be used only when creating graphical pcells.

Arguments

d_labelId	Database ID for the label you want to parameterize. The label is a SKILL expression evaluated in the instance.
S_height	Height of the text in user units. Valid Values: any string or symbol

Value Returned

`t`	Returned if the label is converted to a parameterized label.
`nil`	Returned if the label is not converted to a parameterized label.

Example

```
labelId=dbCreateLabel(geGetEditCellView() "poly1"
20:10 "width*height" "lowerLeft" "R0" "stick" 2)
pcDefineParamLabel(labelId "2")
```

The stretch parameters `width` and `height` are defined before using the example. Creates a label using the values of `width` times `height`. In the example above the dbId produced by the dbCreateLabel function is assigned to a variable named `labelId`. Defines *d_labelId* as a parameterized label whose text is evaluated when you place the pcell. The height of the label is 2 microns.

pcDefineParamLayer

```
pcDefineParamLayer(
     d_cvId
     l_shapes
     g_layerExpr
     l_namelist
     [g_purposeExpr]
     )
     => d_paramLayerId
```

Description

Specifies that a set of shapes has its layer and purpose determined by a parameter definition in the cellview. The selection filter prevents you from selecting instances or stretch control lines.

Arguments

d_cvId	Database ID of the cellview in which the parameter applies.
l_shapes	List of shapes in *d_cvId* that take their layer from a parameter of the cell.
g_layerExpr	SKILL expression for the layer to which the specified shapes belong. This expression evaluates to a string that must be a valid layer name.
l_namelist	List of symbols referenced in *g_layerExpr*. These become parameters of the cell.
g_purposeExpr	SKILL expression for the layer purpose to which the specified shapes belong. This expression evaluates to a string that must be a valid layer purpose.

Value Returned

d_paramLayerId	Group ID used to record the details of the layer parameter.

Example

```
diffLayerParam =
pcDefineParamLayer(geGetEditCellView(w=getCurrentWindow( ))
geGetSelSet(w) 'diffLayer list("nDiff") list('diffLayer))
```

Causes the layer of the set of selected shapes to be determined by the value of the parameter `diffLayer`. It specifies `nDiff` as the default value for `diffLayer` and declares `diffLayer` as a parameter of the cell.

pcDefineParamPath

```
pcDefineParamPath(
     d_pathId
     S_param
     g_margin
     g_width
     n_defaultWidth
     t_snap
     l_namelist
     )
     => d_paramShapeId
```

Description

Defines a path that has its vertices determined by a parameter of the cell. When you place the pcell, you enter a coordinate string that is used as the vertices of the parameterized path. You can use this function more than once in a cellview if all paths have the same vertices.

Arguments

d_pathId	Database ID of a path that has its vertices determined by a parameter of the cell.
S_param	Name of the parameter controlling the vertices of the path. By default, this parameter is `coords`.
g_margin	SKILL expression for the margin by which the path is offset from the coordinate list defined by parameter.
g_width	SKILL expression for the width of the path.
n_defaultWidth	Value to use for the width if *width* evaluates to a nonnumerical value.
t_snap	Snap used by the editor when digitizing path vertices to place an instance of this cell. Valid Values: `orthogonal`, `anyAngle`, `diagonal`, `L90`
l_namelist	List of symbols used in the *g_width* SKILL expression. These become parameters of the pcell.

Value Returned

d_paramShapeId Group ID used to record details of the parameterized paths.

Example

```
pcDefineParamPath(path 'coords 2 'gateWidth 1-5 "orthogonal" list('gateWidth))
```

Defines `path` as a parameterized path that takes its vertices from the value of the parameter `coords` offset by `2,` with a width equal to the value of the `gateWidth` parameter or 1–5 if `gateWidth` evaluates to a nonnumerical value. It tells the editor to use `orthogonal` snap when entering vertices for the path and declares `gateWidth` as a parameter of the cell.

pcDefineParamPolygon

```
pcDefineParamPolygon(
     d_polygonId
     S_param
     g_margin
     t_snap
     l_namelist
     )
     => d_paramShapeId
```

Description

Defines a polygon that has its vertices determined by a parameter of the cell. When you place the pcell, you enter a coordinate string that is used as the vertices of the parameterized polygon. You can use this function more than once in a cellview.

Arguments

d_polygonId	Database ID of the polygon that has its vertices determined by a parameter of the cell.
S_param	Name of the parameter controlling the vertices of the polygon. By default, this parameter is `coords`.
g_margin	SKILL expression for the amount to enlarge or shrink the vertices of the polygon as compared to the coordinate list in the parameter.
t_snap	Snap used by the editor when digitizing polygon vertices to place an instance of this cell. Valid Values: `orthogonal`, `anyAngle`, `diagonal`, `L90`
l_namelist	List of symbols used in the *g_margin* SKILL expression. These become parameters of the pcell.

Value Returned

d_paramShapeId	Group ID used to record details of the parameterized polygons.

Example

```
pcDefineParamPolygon(polygon 'coords '(-margin) "orthogonal" list('margin))
```

Defines *d_polygonId* as a parameterized polygon that takes its vertices from the value of the parameter coords, shrunk by the value of the *g_margin* parameter. It tells the editor to use orthogonal snap when entering vertices for the polygon and declares margin as a parameter of the cell.

pcDefineParamProp

```
pcDefineParamProp(
     d_cvId
     t_name
     g_expr
     )
     => t/nil
```

Description

Defines a parameterized property that can be accessed using a SKILL procedure. You can use this property to store any value within the pcell.

Arguments

d_cvId	Database ID of the cellview in which the parameter applies.
t_name	Name of the parameterized property.
g_expr	SKILL expression for the property value.

Value Returned

`t`	Returned if the parameterized property is created.
`nil`	Returned if the parameterized property is not created.

Example

```
pcDefineParamProp(cv "myProp" 'length)
```

Defines a property name `myProp` to be a parameterized property of the cellview `cv`. The property `myProp` takes its value from the parameter `length` and is evaluated when the instance is created.

pcDefineParamRect

```
pcDefineParamRect(
     d_rectangleId
     S_param
     g_margin
     l_namelist
     )
     => d_paramShapeId
```

Description

Defines a rectangle that has its vertices determined by a parameter of the cell. When you place the pcell, you enter two coordinates that are used as the vertices of the parameterized rectangle. You can use this function more than once in a cellview.

Arguments

d_rectangleId	Database ID of the rectangle that has its vertices determined by a parameter of the cell.
S_param	Name of the parameter controlling the vertices of the rectangle. By default, this parameter is `coords`.
g_margin	SKILL expression for the amount to enlarge or shrink the vertices of the rectangle as compared to the coordinate list in the parameter.
l_namelist	List of symbols used in the *g_margin* SKILL expression. These become parameters of the pcell.

Value Returned

d_paramShapeId	Group ID used to record details of the parameterized rectangles.

Example

```
pcDefineParamRect(rect 'coords '(-margin) list('margin))
```

Defines `rect` as a parameterized rectangle that takes its vertices from the value of the parameter `coords`, reduced by the value of the `margin` parameter. It also declares `margin` as a parameter of the cell.

pcDefineParamRefPointObject

```
pcDefineParamRefPointObject(
     g_objects
     S_param
     l_refpoint
     )
     => d_refPointId
```

Description

Specifies that the location of an object or group of objects in the instance is determined by the location of a reference point that is a parameter of the cell. The objects in the instance have the same relationship to the reference point parameter as the objects in the master cellview have to the corresponding reference point in the master cellview.

Arguments

g_objects	List of objects whose locations are to be determined by the parameterized reference point.
S_param	Name of the parameter for the parameterized reference point. Valid Values: any string or symbol
l_refpoint	Coordinates of the reference point in the master cellview.

Value Returned

d_refPointId	Group ID used to record the details of the parameterized reference point.

Example

```
pcDefineParamRefPointObject(list(pin) 'pinPoint centerBox (pin~>bBox))
```

Causes the object `pin` to be located relative to the coordinate parameter `pinPoint`. The reference point for `pinPoint` in the master cellview is the center point of the `pin` object.

pcDefinePathRefPointObject

```
pcDefinePathRefPointObject(
     l_objects
     S_param
     t_endpoint
     )
     => d_refPointId
```

Description

Specifies that the location of an object or group of objects in the instance be determined by the location of the endpoint of a parameterized path. The objects in the instance have the same relationship to the endpoint of the digitized path in the instance as the objects in the master cellview have to the corresponding endpoint of the parameterized path in the master cellview.

Arguments

l_objects	List of objects whose locations are to be determined by the endpoint of a parameterized path.
S_param	Name of the parameter controlling the parameterized path whose endpoint determines the location of the objects. By default, this parameter is `coords`.
t_endpoint	Endpoint of the path that determines the location of the objects. Valid Values: `first, last`

Value Returned

d_refPointId	Group ID used to store the details of the parameterized reference point.

Example

```
pcDefinePathRefPointObject(list(via) 'coords "first")
```

Causes the object `via` to be located relative to the first vertex of the `coords` parameter. The `coords` parameter controls the vertices of a parameterized path.

pcDefinePCell

```
pcDefinePCell(
     l_cellIdentifier
     l_formalArgs
     body of code
     )
     => d_cellViewId/nil
```

Description

Creates a parameterized (super) master cellview. This function is generated when you compile a pcell.

Arguments

l_cellIdentifier	List containing the library database ID, cell name, view name, and view type. View type is optional and defaults to `maskLayout`. Valid Values: `maskLayout`, `schematic`, and `schematicSymbol`
l_formalArgs	The parameter declaration section, containing a list of input parameters and their default values, all enclosed in parentheses. Optionally, you can also include the data type. Each element of the list is itself a list containing a parameter symbol, its (optional) data type, and a default value. Do not use the `list` keyword, but enclose the parameters in parentheses instead. For example, you can specify *l_formalArgs* as (*parameter_name default_value*) or (*parameter_name data_type default_value*) For Boolean values, specify the data type or the system might interpret a default value of `nil` incorrectly.
body of code	SKILL code for creating geometries and pins. You must provide code here. Enclose the code in a `let` or `prog` statement. If you use variables in the code, define them at the beginning of the `let` or `prog` statement. Using `let` gives faster performance than `prog`; `prog` allows multiple exits while `let` exits only at its

end. Optionally, from within this code, you can call SKILL functions, application functions, and your own user-defined functions.

Calls to functions from within the *body of code* section are part of the pcell, but the functions themselves are **not** part of the pcell. These functions must be available each time the pcell is evaluated. If a called function is not available when a pcell is evaluated, the pcell fails.

If you want the system to automatically load functions when it opens your library, include the functions in your `libInit.il` file. **Do not** load a file from within a pcell. For more information about what to do and what to avoid when writing pcell code, see "Safety Rules for Creating SKILL Pcells" on page 278.

Value Returned

d_cellViewId	The cellview ID of the parameterized (super) master is returned if the (super) master is created.
`nil`	Returned if the parameterized (super) master cellview is not created.

Example

```
pcDefinePCell(
 list( ddGetObj("tutorial") "ask" "layout")
        (
        (layer  "metal1")
        (width  3.0)
        (length 6.0)
         ) ; end or pcell parameters
    let(
        () ; no local variables in this example
        rodCreateRect (
                             ?name       "minMetal"
                             ?width      width
                             ?layer      layer
                             ?length     length
        ) ; end of rodCreateRect
    ) ; end of let
 ) ; end pcDefinePCell
```

This example creates a rectangle named `minMetal` on the `metal1` layer, with a width equal to 3 and a length equal to six.

For more information about using this function, see "pcDefinePCell Function" in the *Virtuoso Parameterized Cell Reference*.

pcDefineRepeat

```
pcDefineRepeat(
     d_cvId
     l_shapes
     l_namelist
     g_stepX
     g_stepY
     g_repeatX
     g_repeatY
     g_stretchX
     g_stretchY
     g_adjustX
     g_adjustY
     t_direction
     )
     => d_repeatId
```

Description

Defines a repetition parameter to be applied to specified objects. Objects can be repeated in the X direction, Y direction, or both. If the value for the repetition direction, *t_direction*, is `horizontal and vertical`, the `pcDefineRepeat` function creates a two-dimensional array.

Arguments

d_cvId	Database ID of the cellview in which the parameter applies.
l_shapes	List of the shapes replicated.
l_namelist	List of the symbols referenced in the step or repeat expressions. These become parameters of the pcell.
g_stepX	SKILL expression specifying stepping distance in the horizontal direction.
g_stepY	SKILL expression specifying stepping distance in the vertical direction.
g_repeatX	SKILL expression specifying the number of times objects are repeated in the horizontal direction.

g_repeatY	SKILL expression specifying the number of times objects are repeated in the vertical direction.
g_stretchX	Name of the vertical, dependent stretch control line associated with this repetition.
g_stretchY	Name of the horizontal, dependent stretch control line associated with this repetition.
g_adjustX	SKILL expression specifying the adjustment from the reference dimension to be applied to the vertical, dependent stretch control line.
g_adjustY	SKILL expression specifying adjustment from the reference dimension to be applied to the horizontal, dependent stretch control line.
t_direction	Direction of repetition. Valid Values: `horizontal`, `vertical`, `horizontal`, and `vertical`

Value Returned

d_repeatId	Group ID used to store details of the repetition.

Example

```
pcDefineRepeat(
     cv   ; cv is defines as cellview ID
     figs ; list of figures to repeat
     list('numGates 'cellPitch)
     'cellPitch
     nil
     '(numGates-1)
     nil
     'gateStretch
     nil
     '(pcFix(pcRepeat - 1) * pcStep)
     nil
     'horizontal
) ; close for pcDefineRepeat
```


`numGates` and `cellPitch` are parameters of the pcell

Defines a horizontal repetition in cellview `cv` affecting the objects listed in `figs`. The horizontal stepping distance is `cellPitch`. The number of horizontal repetitions is `numGates-1`.

This example declares `numGates` and `cellPitch` as parameters of the pcell. It also names `gateStretch` as a dependent stretch control line for the repetition, with an adjustment from the reference dimension defined by the expression `pcFix(pcRepeat - 1) * pcStep`.

pcDefineSteppedObject

```
pcDefineSteppedObject(
     g_objects
     S_param
     g_step
     g_startOffset
     g_endOffset
     l_namelist
     )
     => d_stepObjectId
```

Description

Defines an object or group of objects to be repeated along the length or perimeter of a parameterized shape that has already been defined in the pcell. The selection filter prevents you from selecting stretch control lines, parameterized shapes, or objects in other repeat-along-shape groups.

Arguments

g_objects	List of objects repeated along the length or perimeter of a parameterized shape.
S_param	Name of the parameter controlling the parameterized shape along which objects are repeated. By default, this parameter is `coords`.
g_step	SKILL expression for the stepping distance between repetitions.
g_startOffset	SKILL expression for the space left between the first vertex of the parameterized shape and the first repeated object.
g_endOffset	SKILL expression for the space left between the last vertex of the parameterized shape and the last repeated object.
l_namelist	List of symbols used in *g_step*, *g_startOffset*, or *g_endOffset* expressions. These become parameters of the pcell.

Value Returned

d_stepObjectId	Group ID used to record the details of the repetition along the parameterized shape.

Example

```
pcDefineSteppedObject(list(via1 via2) 'coords 'viaPitch 1.25 1.25 list('viaPitch))
```

Specifies that objects `via1` and `via2` are to be repeated along the parameterized shape controlled by `coords`, using a stepping distance determined by the parameter `viaPitch`, with a gap of `1.25` between the start of the vertices and the first repetition and between the last repetition and final vertex of parameterized shape. It also declares `viaPitch` as a parameter of the pcell.

pcDefineStretchLine

```
pcDefineStretchLine(
     d_lineId
     g_paramExpr
     t_direction
     f_defval
     f_minval
     f_maxval
     g_stretchRepeated
     )
     => d_StretchId
```

Description

Defines a stretch control line used to control stretching in the X direction or Y direction. Objects repeated in the direction parallel to the stretch direction can be set to stretch.

Arguments

d_lineId	Database ID of the line used as the stretch control line. By default, this line is drawn on the layer `stretch`.
g_paramExpr	SKILL expression or symbol controlling the stretch.
t_direction	Direction of the stretch. Valid Values: `right`, `left`, `rightAndLeft`, `up`, `down`, `upAndDown`
f_defval	Default value (reference dimension) for the stretch.
f_minval	Minimum value for the stretch.
f_maxval	Maximum value for the stretch. Use `nil` if no maximum is specified.
g_stretchRepeated	Boolean expression indicating whether to stretch shapes that are repeated in the direction parallel to the stretch.

Value Returned

d_StretchId	Group ID used to store the details of the stretch parameter.

Example

```
pcDefineStretchLine(stretchLine 'nfetWidth, "right" 1.25 1.25 25 nil)
```

Defines `stretchLine` as a stretch control line controlled by the parameter `nfetWidth`. The stretch direction is `right`. The default and the minimum values are `1.25`, and the maximum value is `25`. Horizontally repeated shapes are not stretched.

pcDeleteCondition

```
pcDeleteCondition(
     d_groupId
     )
     => t/nil
```

Description

Deletes a previously defined conditional inclusion parameter.

Prerequisites

You must have defined the conditional inclusion parameter.

Arguments

d_groupId	Group ID of the conditionally included objects.

Value Returned

t	Returned if the parameter is deleted.
nil	Returned if *d_groupId* is not a conditional inclusion parameter.

Example

```
foreach(cond pcGetCondition(cv) pcDeleteCondition(cond))
```

Deletes all conditional inclusion parameters in the cellview cv.

pcDeleteParamLayer

```
pcDeleteParamLayer(
     d_groupId
     )
     => t/nil
```

Description

Deletes a parameter associating a set of shapes with a layer parameter.

Prerequisites

You must have defined the layer parameter.

Arguments

d_groupId	Group ID of the shapes assigned the layer parameter you want to delete.

Value Returned

t	Returned if the parameter is deleted.
nil	Returned if *d_groupId* is not a valid identifier for a layer parameter.

Example

```
pcDeleteParamLayer(diffLayerParam)
```

Deletes the layer parameter identified by `diffLayerParam`.

pcDeleteParamProp

```
pcDeleteParamProp(
     d_cvId
     t_propname
     )
     => t/nil
```

Description

Deletes a parameterized property in the cellview *d_cvId*.

Arguments

d_cvId	Database ID of the cellview in which the parameterized property should be deleted.
t_propname	Name of the parameterized property.

Value Returned

t	Returned if the parameterized property is deleted.
nil	Returned if no parameterized property is defined with the given name or the parameterized property cannot be deleted.

Example

```
pcDeleteParamProp(cv "myProp")
```

Deletes the parameterized property named myProp.

pcDeleteParamShape

```
pcDeleteParamShape(
     d_memberId
     )
     => t/nil
```

Description

Deletes a parameterized shape directive, causing the specified shape to revert back to a regular (nonparameterized) shape.

Prerequisites

You must have defined the shape as a parameterized path, polygon, or rectangle.

Arguments

d_memberId	Database ID of the shape whose parameter you want to delete.

Value Returned

`t`	Returned if the parameter is deleted.
`nil`	Returned if the shape is not defined as a parameterized path, polygon, or rectangle.

Example

```
foreach(shape allMyParamShapes pcDeleteParamShape(shape))
```

Deletes all shape parameters in the list `allMyParamShapes`.

pcDeleteRefPoint

```
pcDeleteRefPoint(
     d_groupId
     )
     => t/nil
```

Description

Deletes a reference point parameter. You can use this command to delete either a reference point defined relative to a parameter of the cell or a reference point defined relative to a parameterized path endpoint.

Prerequisites

You must have defined the reference point parameters.

Arguments

d_groupId	Database ID of the parameterized reference point you want to delete.

Value Returned

t	Returned if the reference point parameter is deleted.
nil	Returned if *d_groupId* is not a valid identifier for a parameterized reference point.

Example

```
pcDeleteRefPoint(gateContactRefPt)
```

Deletes the parameterized reference point defined by the symbol `gateContactRefPt`.

pcDeleteRepeat

```
pcDeleteRepeat(
     d_groupId
     )
     => t/nil
```

Description

Deletes the repetition parameter *d_groupId*.

Arguments

d_groupId	Database ID of the repetition parameter you want to delete.

Value Returned

t	Returned if the parameter is deleted.
nil	Returned if *d_groupId* is not a repeat identifier.

Example

```
foreach(repeat pcGetRepeats(cv) pcDeleteRepeat(repeat))
```

Deletes all repetition parameters in the cellview cv.

pcDeleteSteppedObject

```
pcDeleteSteppedObject(
     d_groupId
     )
     => t/nil
```

Description

Deletes the repetition-along-shape parameter *d_groupId*.

Arguments

d_groupId	Database ID of the repetition along a shape as returned by `pcDefineSteppedObject`.

Value Returned

`t`	Returned if the parameter is deleted.
`nil`	Returned if *d_groupId* is not a repetition-along-shape identifier.

Example

```
foreach(obj allMySteppedObjs pcDeleteSteppedObject(obj))
```

Deletes all repetition-along-shape parameters in the list `allMySteppedObjs`.

pcGetConditions

```
pcGetConditions(
     d_cvId
     )
     => l_condlist/nil
```

Description

Returns a list of identifiers for conditional inclusion parameters in the specified cellview.

Prerequisites

You must have defined the conditional inclusion parameters.

Arguments

d_cvId	Database ID of the cellview containing conditional inclusion parameters.

Value Returned

l_condlist	List of the object IDs used to store the details of the conditional inclusion parameters.
nil	Returned if *d_cvId* does not identify a cellview or if no conditional inclusion parameters are defined in the cellview.

Example

```
condlist = pcGetConditions(cv)
```

Lists all conditional inclusion parameters defined in cv.

pcGetInheritParamDefn

```
pcGetInheritParamDefn(
     d_instId
     )
     => l_inherit/nil
```

Description

Returns an identifier for an inherited parameter in the specified cellview.

Prerequisites

You must have defined the inherited parameter.

Arguments

d_instId	Database ID of the instance of the pcell whose inherited parameter is to be retrieved.

Value Returned

l_inherit	Group ID used to record details of the inherited parameter.
nil	Returned if *d_instId* does not identify an instance as a pcell or if there is no inherited parameter defined in it.

Example

```
inherit = pcGetInheritParamDefn(inst)
```

Retrieves the inherited parameter defined in *d_instId*.

pcGetInheritParams

```
pcGetInheritParams(
     d_cvId
     )
     => l_inheritlist/nil
```

Description

Returns a list of identifiers for inherited parameters in the specified cellview.

Prerequisites

You must have defined the inherited parameter.

Arguments

d_cvId	Database ID of the cellview whose inherited parameters you want listed.

Value Returned

l_inheritlist	List of object IDs used to record details of the inherited parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no inherited parameters defined in the cellview.

Example

```
inheritlist = pcGetInheritParams(cv)
```

Gets a list of all the inherited parameters defined in cv.

pcGetOffsetPath

```
pcGetOffsetPath(
                    d_cvId
                    l_vertex_points
                    n_offset
    )
    => l_vertex_points/nil
```

Description

Applies the offset to each vertex of the path to create a list of vertices for a longer or shorter version of the path. Returns a list of points for the vertices of the offset version of the path. This function does not change the original path and does not create the offset version of the path

Arguments

d_cvId	Database ID of the cellview containing the path.
l_vertex_points	A list of lists identifying a path in the cellview, where each sublist specifies the coordinates for one vertex. Use the following format: `list( '( x y ) '( x y ) ... '( x y ) )`
n_offset	An integer or floating-point number.

Value Returned

l_vertex_points	A list of lists specifying the vertices of the offset version of the path, in the following format: `((x y) (x y) ... (x y))`
`nil`	The function did not execute successfully.

Example

```
pcGetOffsetPath( cv list( '(6.0 6.0) '(10.0 6.0) '(10.0 16.0) '(20.0 16.0) ) 2.0 )
```

Creates and returns the following list of points, which are offset from the specified path by 2.0:

```
((6.0 8.0)
  (8.0 8.0)
```

```
  (8.0 18.0)
  (20.0 18.0)
)
```

pcGetOffsetPolygon

```
pcGetOffsetPolygon(
                    d_cvId
                    l_vertex_points
                    n_offset
    )
    => l_vertex_points/nil
```

Description

Applies the offset to each edge of the polygon to create a list of the vertices for an oversized or undersized version of the polygon. Returns a list of points for the vertices of the offset version of the polygon. This function does not change the original polygon and does not create the offset version of the polygon

Arguments

d_cvId	Database ID of the cellview containing the polygon.
l_vertex_points	A list of lists identifying a polygon in the cellview, where each sublist specifies the coordinates for one vertex. Use the following format: `list( '( x y ) '( x y ) ... '( x y ) )`
n_offset	An integer or floating-point number; a positve number specifies an oversized polygon, a negative number specifies an undersized polygon.

Value Returned

l_vertex_points	A list of lists specifying the vertices of an oversized or undersized version of the polygon after the offset has been applied. The list is in the following format: `((x y) (x y) ... (x y))`
`nil`	The polygon was not found or the offset was not applied successfully.

Example

```
pcGetOffsetPolygon( cv list( '(2.7 2.4) '(8.0 2.4) '(8.0 7.1) '(6.4 7.1)
'(6.4 4.3) '(2.7 4.3) ) 3.0 )
```

Oversizes the polygon specified in the list by adding 3.0 to every edge, and returns the following points for the uncreated oversized polygon:

```
((-0.3 -0.6)
   (11.0 -0.6)
   (11.0 10.1
   (3.4 10.1)
   (3.4 7.3)
   (-0.3 7.3)
)
```

pcGetParameters

```
pcGetParameters(
     d_cvId
     )
     => l_paramlist/nil
```

Description

Returns a list of parameters and their default values defined in the specified cellview.

Prerequisites

You must have defined the parameters in this cellview.

Arguments

d_cvId	Database ID of the cellview containing the parameters.

Value Returned

l_paramlist	List of the parameter names and their default values. The list is in the form (*t_paramName g_defaultValue*).
nil	Returned if d_cvId does not identify a cellview or if there are no parameters defined in the cellview.

Example

```
paramlist = pcGetParameters(cv)
```

Gets list of all parameters defined in cv.

pcGetParamLabelDefn

```
pcGetParamLabelDefn(
     d_labelId
     )
     => d_paramlabelId/nil
```

Description

Returns the parameterized label identifier resulting from a call to `pcDefineParamLabel` on the specified label.

Prerequisites

You must have defined a parameterized label.

Arguments

d_labelId	Database ID of the label whose parameterized label is to be retrieved.

Value Returned

d_paramlabelId	Object ID of the parameterized label.
`nil`	Returned if *d_labelId* does not identify a label or there are no parameterized labels defined in it.

Example

```
paramLabel = pcGetParamLabelDefn(label)
```

Retrieves parameterized labels defined in `label`.

pcGetParamLabels

```
pcGetParamLabels(
     d_cvId
     )
     => l_labellist/nil
```

Description

Returns a list of parameterized labels in the specified cellview.

Prerequisites

You must have defined a parameterized label.

Arguments

d_cvId	Database ID of the cellview containing parameterized labels.

Value Returned

l_labellist	List of object IDs of the parameterized labels.
nil	Returned if *d_cvId* does not identify a cellview or if there are no parameterized labels defined in the cellview.

Example

```
labellist = pcGetParamLabels(cv)
```

Gets a list of all the parameterized labels defined in cv.

pcGetParamLayers

```
pcGetParamLayers(
     d_cvId
     )
     => l_layerlist/nil
```

Description

Returns a list of identifiers for layer parameters in the specified cellview.

Prerequisites

You must have defined layer parameters for this cellview.

Arguments

d_cvId	Database ID of the cellview whose layer parameters you want listed.

Value Returned

l_layerlist	List of object IDs used to record details of layer parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no layer parameters defined in the cellview.

Example

```
layerlist = pcGetParamLayers(cv)
```

Gets a list of all the layer parameters defined in cv.

pcGetParamLayerDefn

```
pcGetParamLayerDefn(
     d_instId
     )
     => l_layerId/nil
```

Description

Returns the identifier for a layer parameter of a specified shape.

Prerequisites

You must have defined a layer parameter for this shape.

Arguments

d_instId	Database ID of the shape whose layer parameter is to be retrieved.

Value Returned

l_layerId	Object ID used to record details of a layer parameter.
nil	Returned if *d_instId* does not identify a shape or there are no layer parameters defined in it.

Example

```
layer = pcGetParamLayerDefn(shape)
```

Retrieves the layer parameters defined in `shape`.

pcGetParamProps

```
pcGetParamProps(
     d_cvId
     )
     => l_proplist/nil
```

Description

Lists parameterized properties in the specified cellview.

Prerequisites

You must have defined the parameterized properties.

Arguments

d_cvId	Database ID of the cellview containing parameterized properties.

Value Returned

l_proplist	List of object IDs of parameterized properties.
nil	Returned if *d_cvId* does not identify a cellview or if there are no parameterized properties defined in the cellview.

Example

```
proplist = pcGetParamProps(cv)
```

Gets a list of all parameterized properties defined in cv.

pcGetParamShapeDefn

```
pcGetParamShapeDefn(
     d_instId
     )
     => d_paramShapeId/nil
```

Description

Returns an identifier for a parameterized shape parameter resulting from a call to `pcDefineParamPath`, `pcDefineParamPolygon`, or `pcDefineParamRect`.

Prerequisites

You must have defined the parameterized shape.

Arguments

d_instId	Database ID of the shape whose parameterized shape parameter is to be retrieved.

Value Returned

d_paramShapeId	Group ID used to record details of the parameter.
`nil`	Returned if *d_instId* does not identify a shape or there are no parameterized shape parameters defined in it.

Example

```
paramshape = pcGetParamShapeDefn(shape)
```

Retrieves parameterized shape parameters defined in `shape`.

pcGetParamShapes

```
pcGetParamShapes(
     d_cvId
     )
     => l_paramshapelist/nil
```

Description

Returns a list of identifiers for the parameterized shape parameters in the specified cellview.

Prerequisites

You must have defined a parameterized shape for this cellview.

Arguments

d_cvId	Database ID of the cellview containing the shape parameters you want listed.

Value Returned

l_paramshapelist	List of the object IDs of the shape parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no parameterized shapes defined in the cellview.

Example

```
paramshapelist = pcGetParamShapes(cv)
```

Lists all shape parameters defined in cv.

pcGetRefPointDefn

```
pcGetRefPointDefn(
     d_objectId
     )
     => l_refPointId/nil
```

Description

Returns an identifier for the reference point parameter defined in the specified cellview.

Prerequisites

You must have defined a reference point parameter for the cellview.

Arguments

d_objectId	Database ID of the object with the reference point parameter.

Value Returned

l_refPointId	Group ID used to record details of the reference point parameter.
nil	Returned if *d_objectId* does not identify an object or there is no reference point parameter defined with it.

Example

```
refpoint = pcGetRefPointDefn(fig)
```

Retrieves a reference point parameter defined in `fig`.

pcGetRefPoints

```
pcGetRefPoints(
     d_cvId
     )
     => l_refpointlist/nil
```

Description

Returns a list of identifiers for all the reference point parameters in a specified cellview.

Prerequisites

You must have defined parameters in this cellview.

Arguments

d_cvId	Database ID of the cellview whose reference point parameters you want listed.

Value Returned

l_refpointlist	List of object IDs used to record details of reference point parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no reference point parameters defined in the cellview.

Example

```
refpointlist = pcGetRefPoints(cv)
```

Gets a list of all the reference point parameters defined in cv.

pcGetRepeatDefn

```
pcGetRepeatDefn(
     d_objectId
     )
     => l_repeatlist/nil
```

Description

Returns a list of identifiers for all repetition parameters assigned to an object in the cellview. A single object can be assigned to more than one repetition group.

Prerequisites

You must have defined repetition parameters for the cellview.

Arguments

d_objectId	Database ID of the object with repetition parameters.

Value Returned

l_repeatlist	Group ID for the internal structure used to record details of repetition parameters. Returns one object ID for each repeat parameter defined on the given object.
nil	Returned if *d_objectId* does not identify an object or there are no repetition parameters defined with it.

Example

```
repeat = pcGetRepeatDefn(fig)
```

Retrieves the repetition parameters defined for `fig`.

pcGetRepeats

```
pcGetRepeats(
     d_cvId
     )
     => l_repeatlist/nil
```

Description

Returns a list of identifiers for repetition parameters in the specified cellview.

Prerequisites

You must have defined repetition parameters for the cellview.

Arguments

d_cvId	Database ID of the cellview with repetition parameters.

Value Returned

l_repeatlist	List of object IDs used to record the details of repetition parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no repetition parameters defined in the cellview.

Example

```
repeatlist = pcGetRepeats(cv)
```

Lists all repetition parameters defined in cv.

pcGetSteppedObjectDefn

```
pcGetSteppedObjectDefn(
     d_objectId
     )
     => l_stepobjId/nil
```

Description

Returns an identifier for repetition-along-shape parameters resulting from calls to `pcDefineSteppedObject`. The object cannot be a parameterized shape.

Prerequisites

You must have defined the repetition-along-shape parameters for this object.

Arguments

d_objectId	Database ID of the object for which repetition-along-shape parameters are retrieved.

Value Returned

l_stepobjId	List of object IDs used to record details of repetition along parameterized shape parameters.
`nil`	Returned if *d_objectId* does not identify an object that is defined as a parameterized shape.

Example

```
stepobj = pcGetSteppedObjectDefn(fig)
```

Gets a list of all the repetition-along-shape parameters defined in cellview.

pcGetSteppedObjects

```
pcGetSteppedObjects(
     d_cvId
     )
     => l_stepobjlist/nil
```

Description

Returns a list of identifiers for the repetition-along-shape parameters in the specified cellview.

Prerequisites

You must have defined repetition-along-shape parameters for this cellview.

Arguments

d_cvId	Database ID of the cellview containing the repetition-along-shape parameters.

Value Returned

l_stepobjlist	List of object IDs used to record details of the repetition-along-shape parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there is no repetition-along-shape parameters defined in the cellview.

Example

```
stepobjlist = pcGetSteppedObjects(cv)
```

Gets a list of all the repetition-along-shape parameters defined in cv.

pcGetStretchDefn

```
pcGetStretchDefn(
     d_objectId
     )
     => d_StretchId/nil
```

Description

Returns the identifier for a stretch parameter.

Prerequisites

You must have defined this line as a stretch control line.

Arguments

d_objectId	Database ID of the line with the stretch parameter.

Value Returned

d_StretchId	Group ID used to record details of the stretch control line.
nil	Returned if a stretch control line is not defined.

Example

```
stretch = pcGetStretchDefn(stretchline)
```

Retrieves a stretch parameter defined with `stretchline`.

pcGetStretches

```
pcGetStretches(
     d_cvId
     )
     => l_stretchlist/nil
```

Description

Returns a list of identifiers for the stretch parameters in the specified cellview.

Prerequisites

You must have defined the stretch control lines.

Arguments

d_cvId	Database ID of the cellview containing the stretch parameters.

Value Returned

l_stretchlist	List of object IDs used to record details of the stretch parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no stretch parameters defined in the cellview.

Example

```
stretchlist = pcGetStretches(cv)
```

Gets a list of all stretch parameters defined in cv.

pcGetStretchSummary

```
pcGetStretchSummary(
                    d_cvId
    )
    => l_stretchLines / nil
```

Description

For a parameterized master cell created using the Pcell graphical user interface tool, returns a list of lists, where each list contains the field names and values from the Stretch in X and Stretch in Y forms for each stretch line that was defined for the parameterized cell (cell).

Arguments

d_cvId	Database ID of the cellview containing a pcell master with stretch lines.

Value Returned

l_stretchLines	A list of lists, where each list contains the field names and values from the Stretch in X and Stretch in Y forms for one stretch line that was defined for the pcell.
nil	The function did not execute successfully.

Example

```
pcStretchSummary( cv )
```

For a pcell master containing two stretch lines, returns a list containing two lists, each of which contain the form field names and values for one stretch line; for example:

```
(("Stretch Type: Horizontal " "Name or Expression for Stretch:  Length" "Stretch
Direction:  left" "Reference Dimension (Default):  1.000000")
   ("Stretch Type: Vertical " "Name or Expression for Stretch:  width" "Stretch
Direction:  up" "Reference Dimension (Default):  1.000000")
)
```

pcHICompileToSkill

```
pcHICompileToSkill()
     => t/nil
```

Description

Displays the *Compile To SKILL* form to let you create a SKILL file from the data in the current cellview. The file can then be edited as any SKILL file.

Prerequisites

Before you compile a cellview to a SKILL file, you must define all parameters for the cellview.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the SKILL file was created; otherwise, returns `nil`.

Example

```
pcHICompileToSkill()
```

pcHIDefineCondition

```
pcHIDefineCondition()
     => t/nil
```

Description

Lets you designate specified objects as conditional by prompting you to select one or more objects in the current cellview. When you complete selecting objects, the system displays the *Conditional Inclusion* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the objects were defined as conditional; otherwise, returns `nil`.

Example

```
pcHIDefineCondition()
```

pcHIDefineInheritedParameter

```
pcHIDefineInheritedParameter()
     => t/nil
```

Description

Lets you designate a pcell instance whose parameters should be inherited from the pcell parent in which the instance is placed by prompting you to select the instance in the current cellview. After you select the instance, the system displays the *Define/Modify Inherited Parameters* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the function was successful; otherwise, returns `nil`.

Example

```
pcHIDefineInheritedParameter()
```

pcHIDefineLabel

```
pcHIDefineLabel()
     => t/nil
```

Description

Displays the *Define Parameterized Label* form to let you create a parameterized label for pcell you are currently editing. After you complete the form, the system prompts you to enter an anchor point for the label.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the label was created; otherwise, returns `nil`.

Example

```
pcHIDefineLabel()
```

pcHIDefineLayer

```
pcHIDefineLayer()
     => t/nil
```

Description

Lets you designate specified shapes to be in a parameterized layer group by prompting you to select one or more shapes in the current cellview. When you complete selecting shapes, the system displays the *Define Parameterized Layer* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the parameterized label group was created; otherwise, returns `nil`.

Example

```
pcHIDefineLayer()
```

pcHIDefineParamCell

```
pcHIDefineParamCell(
     [l_cellIdentifier]
     )
     => t/nil
```

Description

Displays the *Compile To Pcell* form to let you create a graphical pcell (super) master in the database from the design in the current window or from the cellview you specify. If you do not compile a pcell before you place an instance of it in another design, the system interprets the design as a standard fixed cell instead of a pcell. Each time you edit a graphical pcell, you must recompile it so that all placed instances reflect the changes.

Prerequisites

You must have already defined parameters for the cellview.

Arguments

l_cellIdentifier
: List containing the library database ID, cell name, view name, and view type. View type is optional and defaults to `maskLayout`.
Valid Values for view type: `maskLayout`, `schematic`, and `schematicSymbol`

Value Returned

`t/nil`
: Returns `t` if the parameterized (super) master was created; otherwise, returns `nil`.

Example

```
pcHIDefineParamCell(cellview_Id)
```

pcHIDefineParameterizedShape

```
pcHIDefineParameterizedShape()
     => t/nil
```

Description

Lets you assign the vertices of a shape as parameters of the pcell you are currently editing by prompting you to select a shape. You can parameterize paths, polygons, and rectangles. When you place an instance of the pcell, you supply values for the parameters by entering coordinates. After you select a shape, the system displays the appropriate *Define Parameterized Shapes* form, depending on the type of shape you selected.

You can define multiple shapes in the same pcell master as parameterized, as long as they are all of the same shape type. For example, you can define several rectangles as parameterized, but you cannot define one rectangle and one polygon as parameterized in the same pcell master. For more information about parameterized shapes, see "Parameterized Shapes Commands" in the *Virtuoso Parameterized Cell Reference*.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the parameterized shape was created; otherwise, returns `nil`.

Example

```
pcHIDefineParameterizedShape()
```

pcHIDefineParamRefPointObject

```
pcHIDefineParamRefPointObject()
     => t/nil
```

Description

Lets you specify a reference point parameter as the origin point for a selected object or group of objects in the pcell you are currently editing. The system prompts you for the reference point, then prompts you to select the object(s). When you complete selecting objects, the system displays the *Reference Point by Parameter* form. There can be only one reference point parameter defined in a pcell.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the parameterized reference point was created; otherwise, returns `nil`.

Example

```
pcHIDefineParamRefPointObject()
```

pcHIDefinePathRefPointObject

```
pcHIDefinePathRefPointObject()
     => t/nil
```

Description

Lets you specify the end of a parameterized path as the reference point for objects in the pcell you are currently editing by prompting you to select the objects. When you complete selecting objects, the system displays the *Reference Point by Path Endpoint* form. There can be only one reference-point-by-path-endpoint parameter defined in a pcell.

Prerequisites

The parameterized path must exist before you define the reference point.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the reference point was created; otherwise, returns `nil`.

Example

```
pcHIDefinePathRefPointObject()
```

pcHIDefineProp

```
pcHIDefineProp()
     => t/nil
```

Description

Displays the *Parameterized Property* form to let you specify a property for the pcell you are currently editing.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the property was created; otherwise, returns `nil`.

Example

```
pcHIDefineProp()
```

pcHIDefineRepeat

```
pcHIDefineRepeat(
     t_direction
     )
     => d_repeatId
```

Description

Lets you define a repetition parameter for specified objects by prompting you to select one or more objects in the current cellview. When you complete selecting objects, the system displays one of the following forms, depending on the value of the *t_direction* argument: *Repeat in X*, *Repeat in Y*, or *Repeat in X and Y*.

Arguments

t_direction	Direction of repetition. When the value is 2D, the function creates a two-dimensional array. Valid Values: `horizontal`, `vertical`, and `2D`

Value Returned

d_repeatId	Group ID used to store details of the repetition.

Example

```
pcHIDefineRepeat("horizontal")
```

pcHIDefineSteppedObject

```
pcHIDefineSteppedObject()
     => t/nil
```

Description

Lets you specify an object or group of objects to repeat along the coordinate string controlling a parameterized shape by prompting you to select the objects in the pcell you are currently editing. The pcell must already contain a parameterized shape. When you complete selecting objects, the system displays the *Repeat Along Shape* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the repetition group was created; otherwise, returns `nil`.

Example

```
pcHIDefineSteppedObject()
```

pcHIDefineStretch

```
pcHIDefineStretch(
     t_direction
     )
     => d_StretchId
```

Description

Allows you to define a stretch parameter for specified objects by prompting you to select one or more objects in the current cellview. The argument *t_direction* determines which form the system displays when you complete selecting objects: *Stretch in X* or *Stretch in Y*.

Arguments

t_direction	Direction of the stretch. Valid Values: `right`, `left`, `rightAndLeft`, `up`, `down`, `upAndDown`

Value Returned

d_StretchId	Group ID used to store the details of the stretch parameter.

Example

```
pcHIDefineStretch("up")
```

pcHIDeleteCondition

```
pcHIDeleteCondition()
     => t/nil
```

Description

Lets you delete an object from a conditional induction group by prompting you to select the object in the current cellview. When you select the object, the system displays the *Delete Conditional Inclusion* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the object was removed from the conditional inclusion group; otherwise, returns `nil`.

Example

```
pcHIDeleteCondition()
```

pcHIDeleteLayer

```
pcHIDeleteLayer()
     => t/nil
```

Description

Lets you remove a parameterized layer group by prompting you to select a shape in the group. When you select a shape, the system highlights all objects in the parameterized layer group and displays the *Delete Parameterized Layer* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the parameterized layer group was deleted; otherwise, returns `nil`.

Example

```
pcHIDeleteLayer()
```

pcHIDeleteParameterizedShape

```
pcHIDeleteParameterizedShape()
     => t/nil
```

Description

Lets you delete a parameterized shape from the pcell you are currently editing by prompting you to select the parameterized shape. When you delete a parameterized shape, the coordinates of the shape are no longer parameters of the pcell. After you select a shape, the system displays the appropriate *Delete Parameterized Shape* form, depending on the type of shape you selected.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the parameterized shape was deleted; otherwise, returns `nil`.

Example

```
pcHIDeleteParameterizedShape()
```

pcHIDeleteProp

```
pcHIDeleteProp()
     => t/nil
```

Description

Displays the *Delete Parameterized Property* form to let you specify a property to delete from the pcell you are currently editing. When there is more than one parameterized property defined for the pcell, click *Next* to view another property.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the property was deleted; otherwise, returns `nil`.

Example

```
pcHIDeleteProp()
```

pcHIDeleteRefPointObject

```
pcHIDeleteRefPointObject()
     => t/nil
```

Description

Lets you delete either a reference point defined as a parameter of the cellview or a reference point defined relative to the endpoint of a parameterized path by prompting you to select any object in the reference point group. After you select an object, the system highlights all objects in the group and opens either the *Delete Reference Point* form or the *Delete Reference Point By Path* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the reference point group was deleted; otherwise, returns `nil`.

Example

```
pcHIDeleteRefPointObject()
```

pcHIDeleteRepeat

```
pcHIDeleteRepeat()
     => t/nil
```

Description

Lets you delete a repeat group from the pcell you are currently editing by prompting you to select a shape in the repeat group you want to delete. After you select a shape, the system highlights all shapes in the group and displays one of the following forms, depending on the type of repeat group you selected: *Delete Repeat in X,* *Delete Repeat in Y,* or *Delete Repeat in X and Y.*

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the repeat group was deleted; otherwise, returns `nil`.

Example

```
pcHIDeleteRepeat()
```

pcHIDeleteSteppedObject

```
pcHIDeleteSteppedObject()
     => t/nil
```

Description

Lets you delete a repetition along shape group from the pcell you are currently editing by prompting you to select a member (shape) of the repetition along shape group. After you select a member, the system highlights all shapes in the group and displays the *Delete Repeat Along Shape*..

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the repetition along shape group was deleted; otherwise, returns `nil`.

Example

```
pcHIDeleteSteppedObject()
```

pcHIDisplayCondition

```
pcHIDisplayCondition()
     => t/nil
```

Description

Highlights a conditional inclusion group in the current cellview window and displays the *Show Conditional Inclusion* text window with information about the highlighted group. When there is more than one conditional inclusion group, click *OK* in the text window to view the next one.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if there are no inclusion groups or if the Show Conditional Inclusion text window does not display; otherwise, returns `nil`.

Example

```
pcHIDisplayCondition()
```

pcHIDisplayInheritedParameter

```
pcHIDisplayInheritedParameter()
     => t/nil
```

Description

Highlights a pcell instance whose parameters are inherited from the pcell parent in which the instance is placed and displays the *Show Inherited Parameters* text window with information about the highlighted instance. When there is more than one pcell instance with inherited parameters, click *OK* in the text window to view the next one.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if there are no instances with inherited parameters or if the Show Inherited Parameters text window does not display; otherwise, returns `nil`.

Example

```
pcHIDisplayInheritedParameter()
```

pcHIDisplayLayer

```
pcHIDisplayLayer()
     => t/nil
```

Description

Highlights a parameterized layer group in the current cellview window and displays a *Show Parameterized Layer* text window with information about the highlighted group. When there is more than one parameterized layer group, click *OK* in the text window to view the next one.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if there are no parameterized layer groups or if the Show Parameterized Layer text window does not display; otherwise, returns `nil`.

Example

```
pcHIDisplayLayer()
```

pcHIDisplayParameterizedShape

```
pcHIDisplayParameterizedShape()
     => t/nil
```

Highlights a parameterized shape in the current cellview window and displays the *Show Parameterized Shape* text window with information about the highlighted group. When there is more than one parameterized shape, click *OK* in the text window to view the next one.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if there are no parameterized shapes or if the Show Parameterized Shape text window does not display; otherwise, returns `nil`.

Example

```
pcHIDisplayParameterizedShape()
```

pcHIDisplayParams

```
pcHIDisplayParams()
     => t/nil
```

Description

Although the `pcHIDisplayParams` function still displays the *Show Parameters* text window, the information contained in the window might not be complete. The *Show Parameters* command has been replaced by the *Edit Parameters* command. To display information about pcell parameters, use either the `pcHIEditParameters` function or the `pcHIParamsSummarize` function.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if there are no parameters defined for the pcell or if the *Show Parameters* text window does not display; otherwise, returns `nil`.

Example

```
pcHIDisplayParams()
```

pcHIDisplayProp

```
pcHIDisplayProp()
     => t/nil
```

Description

Displays a *Show Parameterized Property* text window with information about all parameterized properties defined for the pcell you are currently editing.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the text window was displayed; otherwise, returns `nil`.

Example

```
pcHIDisplayProp()
```

pcHIDisplayRefPointObject

```
pcHIDisplayRefPointObject()
     => t/nil
```

Description

Highlights a reference point group in the current cellview window and opens the *Show Reference Point* text window with information about the highlighted group. When there is more than one reference point group, click *OK* in the text window to view the next one.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if there are no reference point groups or if the Show Reference Point text window does not display; otherwise, returns `nil`.

Example

```
pcHIDisplayRefPointObject()
```

pcHIDisplayRepeat

```
pcHIDisplayRepeat()
     => t/nil
```

Highlights a repeat group in the current cellview window and displays one of the following text windows, depending on the type of group you selected: *Show Repeat in X*, *Show Repeat in Y*, or *Show Repeat in X and Y* with information about the highlighted group. When there is more than one repeat group, click *OK* in the text window to view the next one.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if a repeat group exists and the text window was displayed; otherwise, returns `nil`.

Example

```
pcHIDisplayRepeat()
```

pcHIDisplaySteppedObject

```
pcHIDisplaySteppedObject()
     => t/nil
```

Description

Highlights the objects in a repetition along shape group in the current cellview window and opens the *Show Repetition Along Shape* text window with information about the highlighted group. When there is more than one repetition along shape group, click *OK* in the text window to view the next one.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if there are no repetition along shape groups or if the Show Repetition Along Shape text window does not display; otherwise, returns `nil`.

Example

```
pcHIDisplaySteppedObject()
```

pcHIEditParameters

```
pcHIEditParameters()
     => t
```

Description

Lets you change the data type and/or value for parameters already defined for the pcell by displaying the *Edit Parameters* form.

Arguments

None.

Value Returned

`t`	Always returns `t`.

Example

```
pcHIEditParameters()
```

pcHIModifyCondition

```
pcHIModifyCondition()
     => t/nil
```

Description

Lets you add objects to a conditional inclusion group by prompting you to select an object in the inclusion group you want to modify. The system highlights all objects in the selected group and prompts you to select shapes to be added to the group. When you complete selecting the objects, the system displays the *Modify Conditional Inclusion* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the selected shapes were added to the conditional inclusion group; otherwise, returns `nil`.

Example

```
pcHIModifyCondition()
```

pcHIModifyLabel

```
pcHIModifyLabel()
     => t/nil
```

Description

Prompts you to select the parameterized label you want to modify. After you select the label, displays the *Modify Parameterized Label* form to let you change the values for the selected label.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the parameterized label was modified; otherwise, returns `nil`.

Example

```
pcHIModifyLabel()
```

pcHIModifyLayer

```
pcHIModifyLayer()
     => t/nil
```

Description

Lets you add shapes to a parameterized layer group by prompting you to select an object in the layer group you want to modify. The system highlights all objects in the selected group and prompts you to select shapes to be added to the group. When you complete selecting the shapes, the system displays the *Modify Parameterized Layer* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the selected shapes were added to the parameterized layer group; otherwise, returns `nil`.

Example

```
pcHIModifyLayer()
```

pcHIModifyParams

```
pcHIModifyParams()
     => t
```

Description

Lets you change the data type and/or value for parameters already defined for the pcell by displaying the *Edit Parameters* form. This function is equivalent to the `pcHIEditParameters` function.

Arguments

None.

Value Returned

`t`	Always returns `t`.

Example

```
pcHIModifyParams()
```

pcHIModifyRefPointObject

```
pcHIModifyRefPointObject()
     => t/nil
```

Description

Lets you add objects to a reference group by prompting you to select an object in the group you want to modify. The system highlights all objects in the selected group and prompts you to select shapes to be added to the group. When you complete selecting the shapes, the system lets you change the reference point by displaying either the *Reference Point by Parameter* form or the *Reference Point by Path Endpoint* form, depending on the type of group you selected. There can be only one reference point parameter and only one reference point by path endpoint defined for a pcell.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the reference group was modified; otherwise, returns `nil`.

Example

```
pcHIModifyRefPointObject()
```

pcHIModifyRepeat

```
pcHIModifyRepeat()
     => t/nil
```

Description

Lets you add shapes to a repeat group by prompting you to select a shape in the repeat group you want to modify. After you select an object, the system highlights all shapes in the group and prompts you to select shapes to be added to the group. When you complete selecting shapes, the system displays one of the following forms, depending on the type of repeat group you selected: *Modify Repeat in X*, *Modify Repeat in Y*, or *Modify Repeat in X and Y.*

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the repeat group was modified; otherwise, returns `nil`.

Example

```
pcHIModifyRepeat()
```

pcHIModifySteppedObject

```
pcHIModifySteppedObject()
     => t/nil
```

Description

Lets you add shapes to a repetition along shape group by prompting you to select a shape in the group you want to modify. After you select a shape, the system highlights all shapes in the group and displays the *Modify Repetition Along Shape* form.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the repetition along shape group was modified; otherwise, returns `nil`.

Example

```
pcHIModifySteppedObject()
```

pcHIModifyStretchLine

```
pcHIModifyStretchLine()
     => t/nil
```

Description

Prompts you to select the stretch line you want to modify. After you select a stretch line, displays the *Stretch in X* or *Stretch in Y* form, depending on whether you are modifying an X stretch line or a Y stretch line, to let you change the values for the selected stretch line.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the stretch line was modified; otherwise, returns `nil`.

Example

```
pcHIModifyStretchLine()
```

pcHIQualifyStretchLine

```
pcHIQualifyStretchLine()
     => t/nil
```

Description

Lets you add shapes to be affected by a stretch line by prompting you to select the stretch line. After you select a stretch line, prompts you to select the shapes to be affected. No form is displayed.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the stretch line was qualified; otherwise, returns `nil`.

Example

```
pcHIQualifyStretchLine()
```

pcHIRedefineStretchLine

```
pcHIRedefineStretchLine()
     => t/nil
```

Description

Lets you redefine a previously defined stretch control line or change the parameters assigned to a stretch control line by prompting you to select the stretch line. After you select a stretch line, the system prompts you to draw a stretch line to replace the selected stretch line. After you draw the line, the system displays the *Stretch in X* or *Stretch in Y* form, depending on whether you are redefining an X stretch line or a Y stretch line, to let you change the values for the redefined stretch line.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the stretch line was redefined; otherwise, returns `nil`.

Example

```
pcHIRedefineStretchLine()
```

pcHISummarizeParams

```
pcHISummarizeParams()
     => t/nil
```

Description

Displays the *Pcell Parameter Summary* text window with information about the parameters defined for the pcell.

Arguments

None.

Value Returned

`t/nil`	Returns `t` if the *Pcell Parameter Summary* text box was displayed; otherwise, returns `nil`.

Example

```
pcHISummarizeParams()
```

pcModifyParam

```
pcModifyParam(
     d_cvId
     S_param
     t_type
     g_value
     )
     => d_paramId/nil
```

Description

Lets you modify the parameter type and default value for parameters assigned to a compiled pcell.

Arguments

d_cvId	Database ID of the specified cellview.
S_param	Name of the parameter. Valid Values: a string or symbol
t_type	Type of the parameter. Valid Values: `int`, `float`, `Boolean`, `string`, `ILList`
g_value	Default value of the parameter. Valid Value: any value consistent with the value type specified

Value Returned

d_paramId	Group ID of the property that stores the parameter.
`nil`	Returned if the function does not execute.

Example

```
pcModifyParam(supermaster "gateWidth" "float" 0.625)
```

The above example modifies the parameter `gateWidth` to have `float` as its type and `0.625` as its default value.

pcRedefineStretchLine

```
pcRedefineStretchLine(
     d_lineId
     g_paramExpr
     t_direction
     f_defval
     f_minval
     f_maxval
     g_stretchRepeated
     )
     => d_StretchId/nil
```

Description

Redefines the attributes of an existing stretch control line. You can also specify a new location for the stretch control line with this command.

Prerequisites

You must have defined the stretch control line.

Arguments

d_lineId	Database ID of the stretch control line.
g_paramExpr	SKILL expression or symbol controlling the stretch.
t_direction	Direction of the stretch. Valid Values: `right`, `left`, `rightAndLeft`, `up`, `down`, `upAndDown`
f_defval	Default value (reference dimension) for the stretch.
f_minval	Minimum value for the stretch.
f_maxval	Maximum value for the stretch. Use `nil` if no maximum is specified.
g_stretchRepeated	Boolean expression indicating whether to stretch shapes repeated in the direction parallel to the stretch.

Value Returned

d_StretchId	Group ID used to store stretch parameter details.
nil	Returned if *d_lineId* is not defined as a stretch control line.

Example

```
pcRedefineStretchLine(stretchLine 'nfetWidth "rightAndLeft" 1.00 1.00 25 nil)
```

Redefines the stretch control line `stretchLine` as controlled by the parameter `nfetWidth`. The stretch direction is `rightAndLeft`. The default and minimum values are `1.00`, and the maximum value is `25`. Horizontally repeated shapes are not affected.

pcRestrictStretchToObjects

```
pcRestrictStretchToObjects(
     d_stretchId
     l_objlist
     )
     => d_stretchId/nil
```

Description

Lets you specify the objects affected by a particular stretch control line. Objects not specified are not moved or stretched by this stretch control line.

Arguments

d_stretchId	Database ID of the stretch control line.
l_objlist	List of objects whose location can be affected by this stretch control line. If `nil`, the stretch applies to all objects.

Value Returned

d_stretchId	Group ID used to store the stretch details.
`nil`	Returned if *d_stretchId* is not defined or if there are no objects in the list of objects.

Example

```
pcRestrictStretchToObjects(stretchLine
geGetSelSet(getCurrentWindow( )))
```

Restricts the effect of the stretch control line to only the selected objects in the cellview in the current window.

pcSkillGen

```
pcSkillGen(
     d_cellViewId
     t_outputFile
     g_isSkillFile
     )
     => t/nil
```

Description

Converts a specified cellview into a SKILL file. A SKILL file can be edited and loaded back to a cellview after modification. Loading a SKILL file generates a SKILL master; however, the cellview contains only a label with the text: *Warning: The master is defined by the SKILL procedure associated with the cellview.*

Prerequisites

You must have defined the parameters in this cellview.

Arguments

d_cellViewId	Cellview to be converted to SKILL.
t_outputFile	The destination text file.
g_isSkillFile	If `t`, generates a `pcGenCell` procedure. If `nil`, generates a SKILL function.

Value Returned

`t/nil`	Returns `t` if the SKILL file is created; otherwise, returns `nil`.

Example

```
pcSkillGen(cv ~/mySkillCell t)
```

Puts the SKILL code that creates the cellview `cv` in the `~/mySkillFile` file. For example:

```
let( (  pcMember pcStretchGroup stretchOffsetX stretchOffsetY pcLib
pcMaster pcInst pcTerm pcPin pcPinName
        pcNet pcTermNet pcNetName pcTermNetName pcMosaicInst
        tpcParameters pcParamProp pcStep pcStepX pcStepY
        pcRepeat pcRepeatX pcRepeatY pcIndexX pcIndexY
```

```
        pcLayer pcPurpose pcLabelText pcLabelHeight pcPropText
        pcParamText pcCoords pcPathWidth pcPolygonMargin )
pcLib = pcCellView~>lib
pcParameters = pcCellView~>parameters~>value
; generate all the cv's properties
; --------------------------------


; generate all the primitive shapes in layer "pwell (drawing)"
; -------------------------------------------------------------
pcLayer = 6
pcPurpose = "drawing"
pcInst = dbCreateRect(pcCellView list(pcLayer pcPurpose) list(2:0.5 6.5:7.5))
t

)
pcSkillGen(cv ~/mySkillCell nil)
```

Puts the SKILL code that creates the cellview `cv` inside a `pcGenCell` procedure and writes it to the `~/mySkillFile` file. For example:

```
procedure( pcGenCell( pcCellView "d")
let( (  pcMember pcStretchGroup stretchOffsetX stretchOffsetY pcLib
        pcMaster pcInst pcTerm pcPin pcPinName
        pcNet pcTermNet pcNetName pcTermNetName pcMosaicInst
        tpcParameters pcParamProp pcStep pcStepX pcStepY
        pcRepeat pcRepeatX pcRepeatY pcIndexX pcIndexY
        pcLayer pcPurpose pcLabelText pcLabelHeight pcPropText
        pcParamText pcCoords pcPathWidth pcPolygonMargin )
pcLib = pcCellView~>lib
pcParameters = pcCellView~>parameters~>value

; generate all the cv's properties
; --------------------------------

; generate all the primitive shapes in layer "pwell (drawing)"
; -------------------------------------------------------------
pcLayer = 6
pcPurpose = "drawing"
pcInst = dbCreateRect(pcCellView list(pcLayer pcPurpose) list(2:0.5 6.5:7.5))
t
)
)
```

auHiUltraPCell

```
auHiUltraPCell(
     [t_filename]
     )
     => t/nil
```

Description

Displays the *Ultra Pcell* form to let you create an Ultra pcell by compiling multiple pcells into one cell. Optionally, you can save the Ultra pcell SKILL code in a file by specifying the *t_filename* argument. For a description of Ultra pcells, see the Make Ultra Pcell Command chapter in the *Virtuoso Parameterized Cell Reference*.

Prerequisites

Ultra pcells have the following requirements:

- Multiple pcell layouts need not be in the same cell, although they must be in the same library as the completed ultra pcell.
- Each pcell must compile successfully.
- Each pcell must have identical parameters and default values. No pcell can have more or fewer or different parameters from the other pcells. When the Ultra pcell is compiled, it has all of the parameters, which work as designed in each pcell.
- The selector parameter must be unique. It cannot match any other parameter.
- *type*, *objType*, *name*, *cell*, and *status* are reserved words and cannot be used as selector parameters.

Arguments

t_filename	Filename (or path and filename) in which you want to save the SKILL code output by the *Make Ultra Pcell* command.

Value Returned

t/nil	Returns t if the form opened, and optionally, the SKILL file was created; otherwise, returns nil.

Example

```
auHiUltraPCell( "myUltraPcell.il" )
```

Pcell Compiler Customization SKILL Functions

This section provides syntax, descriptions, and examples for the SKILL functions associated with customizing the Pcell compiler. Refer to "Customizing the Pcell Compiler" in the *Parameterized Cell Reference Guide* for more information.

pcUserAdjustParameters

```
pcUserAdjustParameters(
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any objects. The procedure is normally used to generate code to transform user-specified parameter values, such as to snap them to an even value. Parameters can then be referenced as variables in the SKILL code that is generated.

Arguments

p_port	Port to which the output code is generated.

Value Returned

t/nil	Return value is not relevant.

Example

```
procedure( pcUserAdjustParameters( port )
; stretch implemented by 2 stretch control lines ⇒
divide parameter value by 2
fprintf(port "ch_width = ch_width/2\n")
 )
```

Generates a call to a SKILL procedure to divide the ch_width parameter value by 2.

pcUserGenerateArray

```
pcUserGenerateArray(
     d_mosaic
     t_masterTag
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any simple arrays (mosaics) in a master pcell. The procedure is normally used to suppress array generation or to modify arrays.

Arguments

d_mosaic	Database ID of array.
t_masterTag	Name for the master pcell of the array that can be used in the generated SKILL code.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce the array in the submaster pcell.
nil	Compiler generates the code to reproduce the array in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserGenerateArray(mosaic master port )
if( mosaic~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for the array named `userSpecial`.

pcUserGenerateInstance

```
pcUserGenerateInstance(
     d_inst
     t_masterTag
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any instances in a master pcell. The procedure is normally used to suppress instance generation or to modify instances.

Arguments

d_inst	Database ID of instance.
t_masterTag	Name for the master pcell that can be used in the generated SKILL code.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce the instance in the submaster pcell.
nil	Compiler generates the code to reproduce the instance in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserGenerateInstance(inst master port )
  if( inst~>name == \"userSpecial\" t nil)
 )
```

Suppresses code generation for an instance named `userSpecial`.

pcUserGenerateInstancesOfMaster

```
pcUserGenerateInstancesOfMaster(
     d_masterCV
     l_instanceList
     t_tag
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler for every master pcell in a master pcell. The compiler calls the procedure before it generates code for instances (but not arrays) for the master. The procedure is normally used to generate code to switch masters.

Arguments

d_masterCV	Database ID of master pcell.
l_instanceList	List of database IDs for all instances of *d_masterCV* in the master pcell.
t_tag	Name of the master that can be used in SKILL code for placing instances of the master.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce the instances of this master pcell in the submaster pcell.
nil	Compiler generates code to reproduce the instances of this master pcell in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserGenerateInstancesOfMaster(master instances tag port )
if( master~>cellName == \"userSpecialNand\" then
; switch master to generic one
```

```
fprintf(port %s = dbOpenCellViewByType(pcLib \"nand\" \"%s\")\n"
tag master~>viewName)
)
; always want code for instances to be generated by
; compiler
nil
)
```

Replaces instances of `userSpecialNand` cell with instances of `nand` cell.

pcUserGenerateLPP

```
pcUserGenerateLPP(
     d_lpp
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes shapes belonging to layer-purpose pairs in the master pcell. The procedure is normally used to suppress shape-set generation.

Arguments

d_lpp	Database ID of layer-purpose pair.
p_port	Port to which the output code is generated.

Value Returned

`t`	Compiler does not generate the code to reproduce any shapes belonging to the layer-purpose pair in the submaster pcell.
`nil`	Compiler generates the code to reproduce shapes belonging to the layer-purpose pair in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserGenerateLPP(lpp port )
if( lpp~>layerName == \"userSpecial\" t nil)
)
```

Suppresses code generation for all shapes on a layer called `userSpecial`.

pcUserGeneratePin

```
pcUserGeneratePin(
     d_pin
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes pins on any terminals in the master pcell. The procedure is normally used to suppress pin generation or to modify pins.

Arguments

d_pin	Database ID of pin on pcell.
p_port	Port to which output code is generated.

Value Returned

`t`	Compiler does not generate code to reproduce the pin in the submaster pcell.
`nil`	Compiler generates code to reproduce the pin in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserGeneratePin(pin port )
     if( pin~>term~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for the pin if the terminal is called `userSpecial`.

pcUserGenerateProperty

```
pcUserGenerateProperty(
     d_object
     d_prop
     t_tag
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes properties on any objects. The procedure is normally used to suppress property generation in the master pcell.

Arguments

d_object	Database ID of the object to which the property is attached.
d_prop	Database ID of the property.
t_tag	Name for the object that can be used in any SKILL code generated.
p_port	Port to which the output code is generated.

Value Returned

`t`	Compiler does not generate code to reproduce the property in the submaster pcell.
`nil`	Compiler generates code to reproduce the property in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserGenerateProperty(obj prop tag port )
if( prop~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for a property called `userSpecial`.

pcUserGenerateShape

```
pcUserGenerateShape(
     d_shape
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any shapes in the master pcell. The procedure is normally used to *suppress* shape generation or to modify shapes.

Arguments

d_shape	Database ID of shape.
p_port	Port to which the output code is generated.

Value Returned

`t`	Compiler does not generate the code to reproduce the shape.
`nil`	Compiler generates the code to reproduce the shape as if the procedure were not called.

Example

```
procedure( pcUserGenerateShape(shape port )
if( cond = shape~>userSpecialProp then
; generate conditional code
fprintf(port "if( %L then\n" cond)
)
; always generate code for this shape
nil
)
```

Looks for the property `userSpecialProp` for the shape. If the property exists, the compiler generates SKILL code to test the results of evaluating the property value when an instance is placed. The code to reproduce the shape is generated by the compiler within a conditional block, so the shape is reproduced in the submaster pcell only if the condition evaluates to a `nil` value.

You need to close the condition properly (generating closing parentheses) in the call to `pcUserPostProcessObject`.

pcUserGenerateTerminal

```
pcUserGenerateTerminal(
     d_terminal
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any terminals in the master pcell. The procedure is normally used to suppress terminal generation or to modify terminals.

Arguments

d_terminal	Database ID of the terminal on the pcell.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate code to reproduce the terminal in the submaster pcell.
nil	Compiler generates code to reproduce the terminal in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserGenerateTerminal(term port )
     if( term~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for a terminal called `userSpecial`.

pcUserInitRepeat

```
pcUserInitRepeat(
     l_stepX
     l_stepY
     l_repeatX
     l_repeatY
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any repetitions. The procedure is normally used to generate code to set the values of variables for repetition parameters.

Arguments

l_stepX	SKILL list for the expression governing the X-stepping distance of the repetition.
l_stepY	SKILL list for the expression governing the Y-stepping distance of the repetition.
l_repeatX	SKILL list for the expression governing the number of repetitions in the X direction.
l_repeatY	SKILL list for the expression governing the number of repetitions in the Y direction.
p_port	Port to which the output code is generated.

Value Returned

t/nil	Return value is not relevant.

Example

```
procedure( pcUserInitRepeat( sX sY rX rY port )
     ; keep record of step distance and repetitions
     fprintf(port "pcUserStep = %L\n" sY)
     fprintf(port "pcUserRepeat = %L\n" rY)
)
```

Generates a call to SKILL procedures to record repetition parameters.

pcUserPostProcessCellView

```
pcUserPostProcessCellView(
     d_cv
     t_tag
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler after it processes any object in a pcell. The procedure is normally used to generate code to process a list of objects that was built during compilation.

Arguments

d_cv	Database ID of the master pcell being processed.
t_tag	Name that can be used to refer to the pcell in the SKILL code output by the procedure.
p_port	Port to which the output code is generated.

Value Returned

t/nil	Return value is not relevant.

Example

```
procedure( pcUserPostProcessCellView( cv tag port )
     ; adjust contacts up by half "slop" amount
fprintf(port "foreach( contact PCUserContacts \n")
fprintf(port "    dbMoveShape(contact pcCellView
list( 0:(width - PCUserRepeat*PCUserStep) / 2
\"R0\"))\n")
fprintf(port ")\n")
)
```

Generates a call to a SKILL procedure to move the list of database objects.

pcUserPostProcessObject

```
pcUserPostProcessObject(
     d_obj
     t_tag
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler after it processes any object (instance, shape, terminal, and so forth) in a master pcell. The procedure is normally used to generate code to modify a generated object.

Arguments

d_obj	Database ID of object.
t_tag	Name for the object in the generated SKILL code.
p_port	Port to which the output code is generated.

Value Returned

t/nil	Return value is not relevant.

Example

```
procedure( pcUserPostProcessObject(obj tag port )
if( obj~>objType == "inst"
&& obj~>userSpecialProp then
     ; generate code to change its magnification
fprintf(port "%s~>mag = pcUserMagScale\n" tag)
)
)
```

Checks to see if the object is an instance and has a property `userSpecialProp`. If so, code is generated to change the magnification of the instance to `pcUserMagScale` (which was defined using `pcUserPreProcessCellView`).

pcUserPreProcessCellView

```
pcUserPreProcessCellView(
     d_cv
     t_tag
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any objects in a pcell. The procedure is normally used to generate code to initialize variables before the compiler processes individual objects.

Arguments

d_cv	Database ID of the master pcell being processed.
t_tag	Name that can be used to refer to the pcell in the SKILL code output by the procedure.
p_port	Port to which the output code is generated.

Value Returned

t/ nil	Return value is not relevant.

Example

```
procedure(pcUserPreProcessCellView( cv tag port )
     ; Initialize list to store the contacts
fprintf(port "PCUserContacts = nil\n")
)
```

Generates a call to a SKILL procedure to initialize a list of database objects.

pcUserSetTermNetName

```
pcUserSetTermNetName(
     d_pinFig
     p_port
     )
     => t/nil
```

Description

A user-defined procedure called by the compiler before it processes any pins on any terminals that are part of a repetition group in the master pcell. The procedure is normally used to customize the connectivity of replicated pins. SKILL code generated by this procedure should assign the net name to the SKILL variable `pcTermNetName`. This is the net name used in the code generated by the compiler to create terminals in the submaster pcell. The SKILL variables `pcIndexX` and `pcIndexY` are available for incorporation into `pcTermNetName` if you need to make different nets for each different repeated pin.

Arguments

d_pinFig	Database ID of the figure associated with the pin.
p_port	Port to which the output code is generated.

Value Returned

`t`	Compiler does not generate code to define the terminal net name in the submaster pcell.
`nil`	Compiler generates code to name the terminal net in the submaster pcell as if the procedure were not called.

Example

```
procedure( pcUserSetTermNetName(fig port )
if( fig~>pin~>term~>name == \"userSpecial\" then
fprintf(port "pcTermNetName = get_string( concat(
\"userSpecial\" pcIndexX)) \n")
t
else ; let compiler generate net name
nil
)
)
```

Generates code to define a net name for the terminal.

3

Compactor Functions

This section provides syntax, descriptions, and examples for the SKILL functions associated with the *Compactor* menu commands.

Introduction on page 421

- syActivateConstraint on page 423
- syCompactView on page 424
- syCreateAlignmentConstraint on page 426
- syCreateFence on page 428
- syCreateJogConstraint on page 430
- syCreateMagneticConstraint on page 431
- syCreatePathWidthConstraint on page 433
- syCreateSeparationConstraint on page 434
- syCreateWireWidthConstraint on page 437
- syDeactivateConstraint on page 438
- syDeleteConstraint on page 439
- syDRC on page 440
- syERC on page 441
- syExtractView on page 442
- syFrameView on page 444
- syGetAlignmentConstraint on page 446
- syGetAutoJog on page 447
- syGetBottomAbutted on page 448

syGetDimensions on page 449

syGetEnv on page 450

syGetFixLowLeftBoundary on page 451

syGetForceChildCellCompact on page 452

syGetGeoOptions on page 453

syGetHardCell on page 455

syGetInitialDirection on page 456

syGetLeftAbutted on page 457

syGetMagneticConstraint on page 458

syGetMaxAutoJogs on page 459

syGetMaxIterations on page 460

syGetPeelbackDistance on page 461

syGetPFrame on page 462

syGetPostProcess on page 463

syGetPreserveCellRow on page 465

syGetPreserveWireWidth on page 466

syGetRightAbutted on page 467

syGetSeparationConstraint on page 468

syGetTopAbutted on page 469

syGetUseChildBdy on page 470

syGetUserConstraint on page 471

syGetWireJogConstraint on page 473

syGetWireWidthConstraint on page 474

syGetXAnchorBoundary on page 475

syGetXPinGrid on page 476

syGetXPinGridOffset on page 477

syGetYAnchorBoundary on page 478

syGetYPinGrid on page 479

syGetYPinGridOffset on page 480

syReadAlignmentConstraint on page 481

syReadMagneticConstraint on page 483

syReadSeparationConstraint on page 485

syReadWireJogConstraint on page 487

syReadWireWidthConstraint on page 488

syRefCellConstraint on page 489

syRefEdgeConstraint on page 490

sySetAutoJog on page 492

sySetAutoJogByLayer on page 493

sySetAutoJogByNet on page 495

sySetAutoJogByWire on page 496

sySetBottomAbutted on page 497

sySetEnv on page 498

sySetFixLowLeftBoundary on page 499

sySetForceChildCellCompact on page 500

sySetGeoOptions on page 501

sySetHardCell on page 503

sySetInitialDirection on page 504

sySetLeftAbutted on page 505

sySetMaxAutoJogs on page 506

sySetMaxIterations on page 507

sySetPeelbackDistance on page 509

sySetPFrame on page 510

sySetPostProcess on page 511

sySetPreserveCellRow on page 513

sySetPreservePathWidth on page 514

sySetPreserveWireWidth on page 515

sySetRightAbutted on page 516

sySetTopAbutted on page 517

sySetUseChildBdy on page 518

sySetXAnchorBoundary on page 519

sySetXPinGrid on page 520

sySetXPinGridOffset on page 521

sySetYAnchorBoundary on page 522

sySetYFirst on page 523

sySetYPinGrid on page 524

sySetYPinGridOffset on page 525

sySymToGeo on page 526

sySymToPolygon on page 527

syUserAllDel on page 528

syUserAllOff on page 529

syUserAllOn on page 530

syUserAllXDel on page 531

syUserAllXOff on page 532

syUserAllXOn on page 533

syUserAllYDel on page 534

syUserAllYOff on page 535

syUserAllYOn on page 536

Introduction

This chapter provides descriptions of SKILL functions associated with the Virtuoso® Compactor menu commands.

The following Virtuoso compactor environment variables can be viewed and set with the `syGetEnv()` and `sySetEnv()` functions.

`cpCpathLPP`
Defines the layer-purpose pair used to display arrow segments in the critical path and overconstraints displays.
Valid Values: a valid layer name, followed by a space, followed by a valid purpose name
Default: `annotate drawing7`

`cpAuxLPP`
Defines the layer-purpose pair used to display the additional line segments that appear in the critical path and overconstraints displays.
Valid Values: either a valid layer name, followed by a space, followed by a valid purpose name, or `none`
Default: `annotate drawing8`

`cpOutlineLPP`
Defines the layer-purpose pair used to outline the objects and groups in the critical path and overconstraint displays.
Valid Values: a valid layer name, followed by a space, followed by a valid purpose name
Default: `annotate drawing9`

`hilite1LPP`
Defines the layer-purpose pair used to highlight the first object or object group selected with the commands on the *Alignment*, *Separation*, and *Magnetic* menus.
Valid Values: a valid layer name, followed by a space, followed by a valid purpose name
Default: `hilite drawing`

`hilite2LPP`
Defines the layer-purpose pair used to highlight the second object selected with the commands on the *Separation* and *Magnetic* menus, or the second object group selected with the commands on the *Alignment* menu.
Valid Values: a valid layer name, followed by a space, followed by a valid purpose name
Default: `marker error`

`hilite3LPP`	Defines the layer-purpose pair used when you select an edge of a flexible object as the first object. A flexible object can be a well, cell boundary, or soft fence. The compactor uses `hilite1LPP` to highlight the edge and `hilite3LPP` to highlight the flexible object. Valid Values: a valid layer name, followed by a space, followed by a valid purpose name Default: `hilite drawing1`
`hilite4LPP`	Defines the layer-purpose pair used when you select an edge of a flexible object as the second object. A flexible object can be a well, cell boundary, or soft fence. The compactor uses `hilite2LPP` to highlight the edge and `hilite4LPP` to highlight the flexible object. Valid Values: a valid layer name, followed by a space, followed by a valid purpose name Default: `hilite drawing2`
`useDBox`	Indicates whether you want information displayed in dialog boxes or in the CIW. A value of `t` indicates that most information will appear in dialog boxes. Valid Values: `t`, `nil` Default: `t`
`arrowScale`	Scale factor used to enlarge or reduce the size of arrowheads used to highlight jog positions, critical paths, and overconstraints. Values larger than 1.0 enlarge the arrowhead, values less than 1.0 reduce the arrowhead. Valid Values: floating-point number between 0.01 and 100 Default: `1.0`
`zoomScale`	Scale factor used to zoom in on an area of the cellview in some constraint editing commands and in the commands in the *Show* menu. Valid Values: any floating-point number greater than or equal to 0 Default: `1.2`

syActivateConstraint

```
syActivateConstraint(
     d_constObjId
     )
     => t/nil
```

Description

Activates the specified alignment, separation, magnetic, wire-jog, or wire-width constraint.

Arguments

d_constObjId	Database object ID of the constraint. Use the ID returned by the function you use to create the constraint.

Value Returned

`t`	The constraint was activated.
`nil`	Function failed.

Example

```
syActivateConstraint( constraintA )
```

Activates `constraintA`.

syCompactView

```
syCompactView(
     d_inputCVId
     l_outputCVName
     [g_hierP]
     [t_logFileName]
     [g_appendMode]
     )
     => d_outputCVId/nil
```

Description

Compacts symbolic objects to the smallest possible area while maintaining user-specified design rules and constraints.

Arguments

d_inputCVId	Database object ID of the cellview to be compacted.
l_outputCVName	Cellname and viewname of the output cellview.
g_hierP	Indicates whether you want single-level or hierarchical compaction. Valid Values: `nil` (invokes a single-level compaction); `t` (invokes a hierarchical compaction) Default: `nil`
t_logFileName	Specifies the name of the compactor log file. Default: `compactor.log`
g_appendMode	Indicates whether you want to append the log output to the log file or overwrite the log file (if it exists) with the log output. Valid Values: `t` (appends the log output to the log file); `nil` (overwrites the log file) Default: `nil`

Value Returned

d_outputCVId	Database object ID of the output cellview.
`nil`	Function failed.

Example

```
outputCellViewId=syCompactView( cellA list("cellB" "compacted") t )
```

Performs hierarchical compaction on the cellview `cellA` and produces a compacted output cellview with the cellname `cellB` and the viewname `compacted`.

syCreateAlignmentConstraint

```
syCreateAlignmentConstraint(
     t_direction
     l_objectLists
     [?unrelaxable]
     )
     => d_constObjId/nil
```

Description

Instructs compactor to align the edges of objects, the centerlines of objects, or layers within objects either horizontally or vertically. Each group of objects with edges to be aligned is called an alignment group.

Arguments

t_direction	Direction of the constraint. Valid Values: X for horizontal alignment, Y for vertical alignment
l_objectLists	List of object lists (necessary because an alignment constraint must have more than one object) in the format where First member is the database object ID of one of the objects to be aligned. Second member specifies the reference line of the object. Third member is an offset from the reference line. If the object is a device, an optional fourth member specifies the layer. If the object is a device but no fourth member is specified, the bounding box is used. Valid Values for Second Member (Rectangular Object): 'l (low), 'c means center, 'h means high. These values apply to both X and Y directions. When *t_direction* is X, 'l, 'c, and 'h represent the left, center, and right lines of the object. When *t_direction* is Y, 'l, 'c, and 'h represent the bottom, center, and top lines of the object.

	Valid Values for Third Member (Polygonal Object): a reference point expressed as `'(xCoord yCoord)` in user units. It should be the midpoint on the relevant line of the object. Valid Values for Fourth Member (layer): a layer number, layer name, or layer-purpose pair
`?unrelaxable`	Indicates that you want the constraint to be unrelaxable. If you omit this argument, compactor relaxes the constraint when an overconstaint is likely to occur. Valid Value: `t` makes the constraint unrelaxable

Value Returned

`d_constObjId`	Database object ID of the constraint.
`nil`	Function failed.

Example

```
syCreateAlignmentConstraint( "X" list(list(objId1 'l 0.0)
     list(objId2 'h 5.0 "metal1")) ?unrelaxable t )
```

Creates an unrelaxable horizontal alignment constraint such that the right edge of the `metal1` layer of object `objId2` is aligned 5 units to the right of the left edge of object `objId1`.

syCreateFence

```
syCreateFence(
     d_inputCVId
     l_layerPurposePair
     l_points
     g_rectOrNot
     )
     => d_fenceObjId/nil
```

Description

Creates a fence. A fence is an area in which movement of objects during compaction is restricted. Objects enclosed by a hard fence are not compacted relative to each other but are compacted as an entity to objects outside the hard fence. Objects enclosed by a soft fence are compacted relative to each other and then are compacted as an entity to objects outside the soft fence.

Note: You cannot activate or deactivate a fence. You can only create or delete a fence. Because `syCreateFence()` returns a database object ID, you must use `dbDeleteObject()` to delete a fence.

Arguments

d_inputCVId	Database object ID of the cellview in which you want the constraint created.
l_layerPurposePair	List of layer-purpose pairs.
l_points	List of points. If *g_rectOrNot* is `t`, this should be only two points.
g_rectOrNot	Specifies whether the fence is a rectangle. Valid Values: `t` for rectangular fence, `nil` for polygonal fence

Value Returned

d_fenceObjId	Database object ID of the fence.
`nil`	Function failed.

Example

```
fenceID=syCreateFence( cellA list("softFence" "drawing")
     '((0.0 0.0) (200.0 100.0)) t )
```

Creates a rectangular soft fence on the layer named `softFence` and with the purpose `drawing` in the cell `cellA`.

syCreateJogConstraint

```
syCreateJogConstraint(
     d_dbWireId
     l_point
     )
     => d_constObjId/nil
```

Description

Creates a point where the compactor can jog the wire during compaction. When you create wire jog points on a path, compactor can insert wire jogs only at the points you specify on the path. The compactor does not insert automatic wire jogs on a path that has wire-jog points.

Arguments

d_dbWireId	Database object ID of the wire.
l_point	X and Y coordinates of the jog location.

Value Returned

d_constObjId	Database object ID of the constraint.
nil	Function failed.

Example

```
syCreateJogConstraint( dbWireId list(10.0 20.0) )
```

Creates a jog point on wire dbWireId at the coordinate (10, 20).

syCreateMagneticConstraint

```
syCreateMagneticConstraint(
     t_direction
     l_object1
     l_object2
     t_magneticType
     )
     => d_constObjId/nil
```

Description

Creates a magnetic constraint between two objects or layers within objects. You can set a magnetic constraint to pull two objects (or layers within objects) together or to push them apart during compaction.

Arguments

t_direction	Direction of the constraint. Valid Values: X pulls the objects together or pushes them apart along the X axis, Y pulls the objects together or pushes them apart along the Y axis
l_object1	List of two or three members as follows: First member is the database object ID of the first object in the constraint. Second member is a Cadence® SKILL language symbol for the reference line of the first object. If the object is a device, an optional third member specifies the layer. If the object is a device but no third member is specified, the bounding box is used. Valid Values for Second Member (Rectangular Object): 'l (low), 'c (center), 'h (high). These values apply to both X and Y directions. When *t_direction* is X, 'l, 'c, and 'h represent the left, center, and right lines of the object. When *t_direction* is Y, 'l, 'c, and 'h represent the bottom, center, and top lines of the object.

	Valid Values for the reference line (Polygonal Object): a reference point expressed as *'(xCoord yCoord)* in user units. It should be the midpoint on the relevant line of the object. Valid Values for Third Member (layer): a layer number, layer name, or layer-purpose pair
l_object2	List of two or three members that specifies the second object in the constraint. The description of this argument is the same as *l_object1*.
t_magneticType	Indicates whether you want the objects pulled together or pushed apart. Valid Values: `attract` pulls the objects together, `repel` pushes the objects apart.

Value Returned

d_constObjId	Database object ID of the constraint.
`nil`	Function failed.

Example

```
syCreateMagneticConstraint( "X" list(object1 'l)
     list(object2 'h) "attract" )
```

Creates a horizontal magnetic constraint between `object1` and `object2`. During compaction, the left edge of `object1` and the right edge of `object2` are pulled together.

syCreatePathWidthConstraint

```
syCreatePathWidthConstraint()
```

Description

Replaced by syCreateWireWidthConstraint.

syCreateSeparationConstraint

```
syCreateSeparationConstraint(
     t_direction
     l_object1
     l_object2
     l_minMaxSeparation
     [?unrelaxable]
     [?ignoreDRC]
     )
     => d_constObjId/nil
```

Description

Separates objects or layers within objects by a specified distance. You can separate two objects, an object and one edge of the cellview boundary, or two edges of the cellview boundary.

Arguments

t_direction	Direction of the constraint. Valid Values: X separates the objects along the X axis; Y separates the objects along the Y axis
l_object1	List of two or three members as follows: First member is the database object ID of the first object to be separated. Second member is a SKILL symbol for the reference line of the first object. If the object is a device, an optional third member specifies the layer. If the object is a device but no third member is specified, the bounding box is used. Valid Values for Second Member (Rectangular Object): 'l (low), 'c (center), 'h (high). These values apply to both X and Y directions. When *t_direction* is X, 'l, 'c, and 'h represent the left, center, and right lines of the object. When *t_direction* is Y, 'l, 'c, and 'h represent the bottom, center, and top lines of the object.

	Valid Values for Second Member (Polygonal Object): a reference point expressed as'*(xCoord yCoord)* in user units. It should be the midpoint on the relevant line of the object. Valid Values for Third Member (layer): a layer number, layer name, or layer-purpose pair
l_object2	List of two or three members that specifies the second object in the constraint. The description of this argument is the same as *l_object1*.
l_minMaxSeparation	List of two members (or `nil` to preserve the current spacing). First number represents the minimum separation distance between the reference lines of the two objects. Second number represents the maximum separation distance between the reference lines of the two objects. To create a fixed-value constraint, specify the same value for both numbers. To create a value constraint within a range, specify a greater value for the second number than the first. To set a minimum distance only, specify the second number as `nil`. To set a maximum distance only, specify the first number as `nil`.
`?unrelaxable`	Indicates that the constraint is unrelaxable. If you omit this argument, compactor can relax the constraint. Valid Value: `t` makes the constraint unrelaxable Default: `nil`
`?ignoreDRC`	Indicates that the constraint is to override the design rule between the object and the boundary (applicable only when the constraint is between the object and the boundary). If you omit this argument, the design rule can override the constraint. Valid Value: `t` makes the constraint override the design rule Default: `nil`

Value Returned

d_constObjId	Database object ID of the constraint.
`nil`	Function failed.

Example

```
syCreateSeparationConstraint( "X" list(object1 'l)
     list(object2 'h) list(5.0 10.0) ?ignoreDRC t )
```

Creates a horizontal separation constraint between `object1` and `object2`, where one of the objects is the cell boundary. After compaction, the left edge of `object1` and the right edge of `object2` are separated by a distance greater than 5 units but less than 10 units. This constraint overrides the design rule between the object and the boundary.

syCreateWireWidthConstraint

```
syCreateWireWidthConstraint(
     d_dbWireId
     n_finalWidth
     )
     => d_constObjId/nil
```

Description

Creates a width constraint on the specified wire (path). The compactor forces the wire to be of this width. If a constraint is already present, this function overrides it. The `syCreateWireWidthConstraint` function is a replacement for the `syCreatePathWidthConstraint` function.

Arguments

d_dbWireId	Database object ID of the wire (path).
n_finalWidth	Width you want applied to the wire.

Value Returned

d_constObjId	Database object ID of the constraint.
`nil`	Function failed.

Example

```
syCreateWireWidthConstraint( dbWireId 5.0 )
```

Creates a width constraint on the wire `dbWireId`. After compaction, the width of the wire is 5.0.

syDeactivateConstraint

```
syDeactivateConstraint(
     d_constObjId
     )
     => t/nil
```

Description

Deactivates the specified alignment, separation, magnetic, wire-jog, or wire-width constraint.

Arguments

d_constObjId	Database object ID of the constraint. Use the ID returned by the function you use to create the constraint.

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
syDeactivateConstraint( constraintA )
```

Deactivates `constraintA`.

syDeleteConstraint

```
syDeleteConstraint(
     d_constObjId
     )
     => t/nil
```

Description

Deletes the specified alignment, separation, magnetic, wire-jog, or wire-width constraint.

Arguments

d_constObjId	Database object ID of the constraint. Use the ID returned by the function you use to create the constraint.

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
syDeleteConstraint( constraintA )
```

Deletes `constraintA`.

syDRC

```
syDRC(
     d_inputCVId
     d_runDir
     )
     => t/nil
```

Description

Checks for layer-to-layer spacing errors in the cellview and stores error data for subsequent error viewing.

Arguments

d_inputCVId	Database object ID of the cellview to be checked.
	
	When executing the function from the command interpreter window, display the database object ID with the following:
	
	`getWindowCellView(`*windowId*`)`
	
	When executing the function from a UNIX window, display the database object ID with the following:
	
	`dbOpenCellView( `*libId* `"`*cellName viewName*`" "r" )`

d_runDir	Directory in which the DRC error data is stored for later interactive viewing.

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
syDRC( dbOpenCellView( libID "cellName" "viewName" "r" ) "/usr/design/drcrun" )
```

Runs Symbolic DRC on the current cellview and stores the error data in `/usr/design/drcrun`.

syERC

```
syERC(
     d_inputCVId
     d_runDir
     )
     => t/nil
```

Description

Checks for electrical rule errors in the cellview and stores error data for subsequent error viewing.

Arguments

d_inputCVId	Database object ID of the cellview to be checked.
	
	When executing the function from the command interpreter window, display the database object ID with the following:
	
	`getWindowCellView(`*windowId*`)`
	
	When executing the function from a UNIX window, display the database object ID with the following:
	
	`dbOpenCellView(` *libId* `"`*cellName viewName*`" "r" )`

d_runDir	Directory in which the ERC error data is stored for later interactive viewing.

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
syERC( dbOpenCellView( libID "cellName" "viewName" "r" ) "/usr/design/ercrun" )
```

Runs Symbolic ERC on the current cellview and stores the error data in `/usr/design/ercrun`.

syExtractView

```
syExtractView(
     d_inputCVId
     l_outputCVName
     [g_hierP]
     [t_logFileName]
     [g_appendMode]
     )
     => d_outputCVId/nil
```

Description

Checks input data and net connectivity of the input cellview and creates an output cellview with the errors (if any) highlighted.

Arguments

d_inputCVId	Database object ID of the input cellview.
l_outputCVName	Cellname and viewname of the output cellview.
g_hierP	Indicates whether you want single-level or hierarchical extraction. Valid Values: `nil` invokes a single-level extraction, `t` invokes a hierarchical extraction Default: `nil`
t_logFileName	Name of the compactor log file. Default: `compactor.log`
g_appendMode	Indicates whether you want to append the log output to the log file or overwrite the log file (if it exists) with the log output. Valid Values: `t` appends the log output to the log file; `nil` overwrites the log file if it exists Default: `nil`

Value Returned

d_outputCVId	Database object ID of the output cell.
`nil`	Function failed.

Example

```
outputCellViewId=syExtractView( cellA, list("cellB" "syExtracted" ) nil )
```

Checks input data and net connectivity of the cellview `cellA`. It then creates another cellview with the cellname `cellB` and the viewname `syExtracted`. This new cellview contains the design data plus the extracted information. The function opens the new cellview with the errors highlighted.

syFrameView

```
syFrameView(
     d_srcCVId
     l_destCellView
     n_mergeFactor
     [g_fillCenter]
     [t_logFileName]
     [g_appendMode]
     )
     => d_outputCVId/nil
```

Description

Creates a protection frame, or abstraction of a cellview, for placement at the next level of hierarchy to improve efficiency of the compactor. The function fills areas between and merges geometries on the same layer to form either a single polygon or multiple polygons, depending on your specifications. In addition, you can specify for the function to fill an area in the center of the cellview that is the maximum design rule from the cell boundary and merge this area with the geometries formed by the first fill-and-merge operation.

Arguments

d_srcCVId	Database object ID of the input cellview.
l_destCellView	Cellname and viewname of the destination cellview.
n_mergeFactor	Maximum distance by which two edges on the same layer must be separated to remain separate during geometry merging. A lower value causes less merging of shapes, leaving more edges on the framed view.
g_fillCenter	Indicates whether to fill an area in the center of the cellview in addition to filling and merging geometries. Valid Values: `t` fills an area in the center of the cellview; `nil` does not Default: `t`
t_logFileName	Name of the compactor log file. Default: `compactor.log`
g_appendMode	Indicates whether you want to append the log output to the log file or overwrite the log file (if it exists) with the log output. Valid Values: `t` appends the log output to the log file; `nil`

overwrites the log file
Default: `nil`

Value Returned

`d_outputCVId`	Database object ID of the output cell.
`nil`	Function failed.

Example

```
framedcVId=syFrameView( cellA list( "childCell" "framed" 10.0 nil )
```

Creates and returns the database object ID of a protection frame created from the cellview `cellA` by merging all geometries on the same layer that are separated by less than 10.0 user units.

syGetAlignmentConstraint

```
syGetAlignmentConstraint(
     d_cvId
     [?object d_objId]
     [?direction s_direction]
     )
     => (d_constraintId ... )/nil
```

Description

Returns a list of database object IDs of alignment constraints in a given direction. If you specify `?object`, the function returns alignment constraints on only that object; otherwise, the function returns all alignment constraints in the cellview. You can use the returned database object IDs as arguments for `syReadAlignmentConstraint()` to get detailed information about a constraint.

Arguments

d_cvId	Database object ID of the cellview, which is ignored if you specific `?object` is also specified.
d_objId	Database object ID of the object.
s_direction	Selects horizontal or vertical constraints, or both. Valid Values: `'X` for horizontal alignment constraints, `'Y` for vertical alignment constraints Default: both horizontal and vertical constraints

Value Returned

d_constraintId	List of constraint database object IDs.
`nil`	Found no constraints in the specified directions.

Example

```
syGetAlignmentConstraint( cv ?object obj ?direction 'X)
     => (db:20106428)
```

Reports that object `obj` has one horizontal alignment constraint.

syGetAutoJog

```
syGetAutoJog(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetAutoJog`.

Arguments

`d_cvId`	Database object ID of the cellview.

Value Returned

`t`	Automatic jog insertion function is enabled.
`nil`	Automatic jog insertion function is disabled.

Example

```
if(syGetAutoJog(cellA)then
     println("Auto jog enabled")
     )
```

Checks the auto jog setting of the cellview `cellA`. If automatic jog insertion is enabled (`t` returned), it displays the message `Auto jog enabled`.

syGetBottomAbutted

```
syGetBottomAbutted(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetBottomAbutted`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Bottom edge of the cell is set for abutment.
`nil`	Bottom edge of the cell is not set for abutment.

Example

```
if (syGetBottomAbutted(cellA) then
     println("Bottom edge to be abutted")
     )
```

Checks the cellview `cellA`. If its bottom edge is to be abutted (`t` returned), it displays the message `Bottom edge to be abutted`.

syGetDimensions

```
syGetDimensions(
     d_inputCVId
     )
     => l_dimensions/nil
```

Description

Returns the width, height, and area of the cellview.

Arguments

d_inputCVId	Database object ID of the input cellview.

Value Returned

l_dimensions	List of the width, height, and area of the cellview.
nil	Function failed.

Example

```
syGetDimensions( cellA )
```

Returns the dimensions of the cellview `cellA`.

syGetEnv

```
syGetEnv(
     t_name
     )
     => g_value/nil
```

Description

Returns the value currently assigned to *t_name*.

For a detailed description of the compactor environment variables, see "Introduction" on page 421.

Arguments

t_name	Name of the compactor environment variable. Valid Values: cpCpathLPP, cpAuxLPP, cpOutlineLPP, hilite1LPP, hilite2LPP, hilite3LPP, hilite4LPP, useDBox, arrowScale, zoomScale

Value Returned

g_value	Value of *t_name*.
nil	Specified variable does not exist.

Example

```
syGetEnv( "cpCpathLPP" )
```

Returns the value of cpCpathLPP.

syGetFixLowLeftBoundary

```
syGetFixLowLeftBoundary(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetFixLowLeftBoundary`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Moves the lower-left corner of the compacted cellview to (0,0).
`nil`	Does not need to move the lower-left corner of the compacted cellview to (0,0).

Example

```
syGetFixLowLeftBoundary( cellA )
     => nil
```

Compactor does not need to move the lower-left corner of the compacted view of `cellA` to (0,0).

syGetForceChildCellCompact

```
syGetForceChildCellCompact(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetForceChildCellCompact`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Compacts all child cells during hierarchical compaction.
`nil`	Compacts only those child cells not previously compacted.

Example

```
syGetForceChildCellCompact( cellA )
     => nil
```

Compactor does not compact previously compacted child cells when performing hierarchical compaction on `cellA`.

syGetGeoOptions

```
syGetGeoOptions(
     d_cvId
     )
     => (x_stopLevel g_smashDevice t_mode)
```

Description

Returns the stop level, smash device flag, and conversion options set by `sySetGeoOptions`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

x_stopLevel	Hierarchical level at which compactor stops converting symbols and cell instances to polygon data. Valid Values: any positive integer indicating a hierarchical level in your design Default: `0` (current cellview)
g_smashDevice	Indicates whether compactor converts symbols and cell instances to polygon data down to the hierarchical level specified by *x_stopLevel*. Valid Values: `t` converts symbols and cell instances to polygons; `nil` does not Default: `t`
t_mode	Conversion options that control the shape of the output polygon. Valid Values: If *g_smashDevice* is `nil`, valid values are `fill notch` or `fill gap`. If *g_smashDevice* is `t`, valid values can be one of the following: `fill notch`, `fill gap`, `merge polygon`, or `no merge`. Default: `no merge` `fill notch` fills any notches smaller than the design rule for the layer or mask layer.

`fill gap` fills any notches smaller than the design rule for the layer or mask layer and fills any gaps smaller than the design rule between two polygons of the same layer or the same mask layer.

`merge polygon` causes abutting or overlapping polygons of the same layer to merge into a single polygon. The output cell produced by this option requires less storage space on a disk.

`no merge` causes abutting or overlapping polygons of the same layer to remain separate.

Example

```
syGetGeoOptions( cellAId )
     => (10 t "fill gap")
```

Returns the stop level, device flag, and conversion values set for `cellAId`.

For `cellAId`, the compactor does the following:

- Converts symbols and cell instances to polygon data to the tenth level of the design hierarchy
- Fills any notches smaller than the design rule for the layer or mask layer
- Fills any gaps smaller than the design rule between two polygons of the same layer or the same mask layer

syGetHardCell

```
syGetHardCell(
     d_inputCVId
     )
     => t/nil
```

Description

Returns the value set by `sySetHardCell`.

Arguments

d_inputCVId	Database object ID of the input cellview.

Value Returned

`t`	Cell is a hard cell and therefore is not compacted.
`nil`	Cell is not a hard cell and therefore is compacted.

Example

```
syGetHardCell( cellA )
     => t
```

`cellA` is a hard cell and therefore is not compacted.

syGetInitialDirection

```
syGetInitialDirection(
     d_cvId
     )
     => X/Y
```

Description

Returns the initial direction of compaction for the specified cellview set by `sySetInitialDirection`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

X	Direction for initial horizontal pass.
Y	Direction for initial vertical pass.

Example

```
syGetInitialDirection( cellA )
     => X
```

The initial direction of compaction for `cellA` is the horizontal direction.

syGetLeftAbutted

```
syGetLeftAbutted(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetLeftAbutted`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Left edge of the cell is set for abutment.
`nil`	Left edge of the cell is not set for abutment.

Example

```
if syGetLeftAbutted(cellA) then
     println("Left edge to be abutted")
```

Checks the cellview `cellA`. If its left edge is to be abutted (if `t` is returned), it displays the message `Left edge to be abutted`.

syGetMagneticConstraint

```
syGetMagneticConstraint(
     d_cvId
     [?object d_objId]
     [?direction s_direction]
     )
     => (d_constraintId ... )/nil
```

Description

Returns a list of database object IDs of magnetic constraints in a given direction. If `?object` is specified, only magnetic constraints on the given object are returned; otherwise, all magnetic constraints in the given cellview are returned. You can use the returned object IDs as arguments for `syReadMagneticConstraint()` to get detailed information about a constraint.

Arguments

d_cvId	Database object ID of the cellview, which is ignored if you specify `?object`.
d_objId	Database object ID of the object.
s_direction	Selects horizontal or vertical constraints, or both. Valid Values: `'X` for horizontal magnetic constraints, `'Y` for vertical magnetic constraints Default: both horizontal and vertical magnetic constraints are returned

Value Returned

d_constraintId	List of constraint database object IDs.
`nil`	No magnetic constraints are found in the directions you specified.

Example

```
syGetMagneticConstraint( cv )
     => (db:20108060 db:20107708 db:20107504)
```

Returns the database object IDs of all magnetic constraints in the cellview `cv`.

syGetMaxAutoJogs

```
syGetMaxAutoJogs(
     d_cvId
     )
     => n_value/nil
```

Description

Returns the value set by `sySetMaxAutoJogs`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

n_value	Maximum number of jogs compactor can add to any one wire.
`nil`	There is no limit to the number of jogs compactor can add to any one wire.

Example

```
syGetMaxAutoJogs( cellA )
     => 5
```

The maximum number of jogs the compactor can add to any wire in `cellA` is 5.

syGetMaxIterations

```
syGetMaxIterations(
     d_cvId
     )
     => g_value
```

Description

Returns the value set by `sySetMaxIterations`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

g_value	Maximum number of iterations. If you specify *n_maxIter* in `sySetMaxIterations()`, a list of two numbers is returned. The first number indicates the maximum number of iterations; the second number indicates the minimum percent of area the cellview must decrease to continue compaction.

Example

```
syGetMaxIterations( cellA )
     => 2.0
```

Performs two iterations when compacting `cellA`.

```
syGetMaxIterations(cellB)
     => (20.0 5.0)
```

Performs a maximum of 20 iterations when compacting `cellB`, but stops compacting when the area decreases by less than 5 percent during the sequence of iterations.

syGetPeelbackDistance

```
syGetPeelbackDistance(
     d_cvId
     )
     => n_distance
```

Description

Returns the value set by `sySetPeelbackDistance`.

Arguments

`d_cvId`	Database object ID of the cellview.

Value Returned

`n_distance`	Wire end adjustment distance.

Example

```
syGetPeelbackDistance( cellA )
     => 2.0
```

Deletes any unconnected wire segment with a length less than or equal to 2 user units when compacting `cellA`.

syGetPFrame

```
syGetPFrame(
     d_cvId
     )
     => t_viewName/nil
```

Description

Returns the value set by `sySetPFrame`.

Arguments

`d_cvId`	Database object ID of the cellview.

Value Returned

`t_viewName`	Name of the frame view to be used for child cells of cellview *`d_cvId`* during compaction.
`nil`	No protection frame viewname is specified for the compaction of cellview *`d_cvId`*.

Example

```
syGetPFrame( cellA )
     => frame
```

If `cellA` is to be compacted with the protection frame option set, `frame` would be used as the frame viewname for all child cells of `cellA`.

syGetPostProcess

```
syGetPostProcess(
     d_cvId
     )
     => t_WlmEnum/nil
```

Description

Returns the process set by `sySetPostProcess`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

t_WlmEnum	Method of wire length minimization. Valid Values: `wwlm`, `min`, `max` Default: `wwlm` `wwlm` minimizes total wire length in the compacted output based on the resistivity factor set for each routing layer in the technology file and then compacts toward the lower-left corner of the cellview. `min` compacts toward the lower-left corner of the cellview without minimizing wire length. `max` compacts toward the upper-right corner of the cellview without minimizing wire length.
`nil`	Process has not been set, therefore compactor minimizes total wire length in the compacted output based on the resistivity factor set for each routing layer in the technology file and then compacts toward the lower-left corner of the cellview.

Example

```
syGetPostProcess( cellA )
     => min
```

Places objects as close as possible to the left or bottom of the compacted cellview when compacting `cellA`.

syGetPreserveCellRow

```
syGetPreserveCellRow(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetPreserveCellRow`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Must preserve cell row structure.
`nil`	Might change cell row structure.

Example

```
syGetPreserveCellRow( cellA )
     => t
```

Preserves cell row structure when compacting `cellA`.

syGetPreserveWireWidth

```
syGetPreserveWireWidth(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetPreserveWireWidth`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Must preserve the drawn wire width.
`nil`	Might change the drawn wire width.

Example

```
syGetPreserveWireWidth( cellA )
     => t
```

Preserves the drawn wire width when compacting `cellA`.

syGetRightAbutted

```
syGetRightAbutted(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetRightAbutted`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Right edge of the cell is set for abutment.
`nil`	Right edge of the cell is not set for abutment.

Example

```
syGetRightAbutted( cellA )
     => t
```

Right edge of `cellA` is set for abutment.

syGetSeparationConstraint

```
syGetSeparationConstraint(
     d_cvId
     [?object d_objId]
     [?direction s_direction]
     )
     => (d_constraintId ... )/nil
```

Description

Returns a list of database object IDs of separation constraints in the given direction. If you specify `?object`, the function returns separation constraints on only that object; otherwise the function returns all separation constraints in the cellview. You can use the returned IDs as arguments for `syReadSeparationConstraint()` to get detailed information about each constraint.

Arguments

d_cvId	Database object ID of the cellview, which is ignored if you specify `?object`.
d_objId	Database object ID of the object.
s_direction	Selects horizontal or vertical constraints, or both. Valid Values: `'X` for horizontal separation constraints, `'Y` for vertical separation constraints Default: both horizontal and vertical constraints

Value Returned

d_constraintId	List of constraint database object IDs.
`nil`	Found no separation constraints in the directions you specified.

Example

```
syGetSeparationConstraint( cv ?direction 'Y )
     => (db:20108580 db:20107320)
```

Returns all of the vertical (Y) separation constraints in cellview `cv`.

syGetTopAbutted

```
syGetTopAbutted(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetTopAbutted`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Top edge of the cell is set for abutment.
`nil`	Top edge of the cell is not set for abutment.

Example

```
syGetTopAbutted( cellA )
     => nil
```

Top edge of `cellA` is not set for abutment.

syGetUseChildBdy

```
syGetUseChildBdy(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetUseChildBdy`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Boundaries (if any) of child cells are used as abstractions during compaction.
`nil`	Boundaries of child cells are not used as abstractions during compaction.

Example

```
syGetUseChildBdy( cellA )
     => t
```

Uses the boundary of `cellA` as an abstraction during compaction.

syGetUserConstraint

```
syGetUserConstraint(
     d_cvId
     [?object d_objId]
     [?direction s_direction]
     )
     => ((s_type (d_constraintId ...)) ... )/nil
```

Description

Returns a list of the database object IDs of all wire-width, wire-jog, alignment, magnetic, and separation constraints in the given direction. If `?object` is specified, only constraints on that object are returned; otherwise, all constraints in the specified cellview are returned. You can use these database object IDs to delete, activate, or deactivate the constraints or to obtain detailed information about the constraints using the appropriate `syReadConstraint()` function for the constraint type.

Arguments

d_cvId	Database object ID of the cellview, which is ignored if you specify `?object`.
d_objId	Database object ID of the object.
s_direction	Selects X or Y constraints, or both. Valid Values: `'X` for X constraints, `'Y` for Y constraints Default: both X and Y constraints are returned

Value Returned

A list of constraint database object IDs grouped by constraint type. You can use the `assoc()` function to extract information from the list. The keywords in the list are as follows:

`'wire`	Wire-width constraint
`'jog`	Wire-jog constraint
`'align`	Alignment constraint
`'mag`	Magnetic constraint
`'`	Separation constraint

`nil` Found no constraints in the directions you specified.

Example

```
syGetUserConstraint( cv ?object obj ?direction 'Y )
     => (('wire (db:4846428)) ('jog (db:4847552))
     (sep (db:4849312)))
```

Checks `obj` for Y constraints. The list returned shows that `obj` has one wire width, one wire jog, and one separation constraint in the Y direction. The object has no alignment or magnetic constraints in the Y direction.

syGetWireJogConstraint

```
syGetWireJogConstraint(
     d_cvId
     [?object d_objId]
     [?direction s_direction]
     )
     => (d_jogId ... )/nil
```

Description

Returns a list of the database object IDs of wire-jog constraints in the direction you specify. If you specify `?object`, the function returns wire-jog constraints on only that object; otherwise, the function returns all wire-jog constraints in the cellview. You can use the returned database object IDs as arguments for `syReadWireJogConstraint()` to get detailed information about the constraints.

Arguments

d_cvId	Database object ID of the cellview which is ignored if you specify `?object`.
d_objId	Database object ID of the object.
s_direction	Selects jogs on vertical or horizontal wires, or both. Valid Values: `'X` for horizontal jogs in vertical wires, `'Y` for vertical jogs in horizontal wires Default: both horizontal and vertical jogs

Value Returned

d_jogId	List of constraint database object IDs. You can use the `assoc()` function to extract information from the list.
`nil`	Wires you specified have no jogs in the directions you specified.

Example

```
syGetWireJogConstraint( cv )
     => (db:20105780 db:20105152 db:20105532)
```

Returns the database object IDs of all three wire jogs in the cellview `cv`.

syGetWireWidthConstraint

```
syGetWireWidthConstraint(
     d_cvId
     [?object d_objId]
     [?direction s_direction]
     )
     => (d_constraintId ... )/nil
```

Description

Returns a list of database object IDs of wire-width constraints in the given direction. If you specify `?object`, the function returns wire-width constraints on only this object; otherwise the function returns all wire-width constraints in the cellview. You can use the returned database object IDs as arguments for `syReadWireWidthConstraint()` to get detailed information about each constraint.

Arguments

d_cvId	Database object ID of the cellview, which is ignored if you specify `?object`.
d_objId	Database object ID of the object.
s_direction	Selects constraints on vertical or horizontal wires, or both. Valid Values: `'X` for vertical wires (X constraints), `'Y` for horizontal wires (Y constraints) Default: both horizontal and vertical[constraints?]

Value Returned

d_constraintId	List of constraint database object IDs. You can use the `assoc()` function to extract information from the list.
`nil`	No constraints in the directions you specified.

Example

```
syGetWireWidthConstraint( cv ?direction 'X )
     => (db:20104028 db:20104904)
```

Returns all wire-width constraints on vertical wires in the cellview `cv`.

syGetXAnchorBoundary

```
syGetXAnchorBoundary(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetXAnchorBoundary`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Cannot move pins on horizontal boundary edges.
`nil`	Can move pins on horizontal boundary edges.

Example

```
syGetXAnchorBoundary( cellA )
     => t
```

Cannot move pins on horizontal boundary edges of `cellA`.

syGetXPinGrid

```
syGetXPinGrid(
     d_cvId
     )
     => f_gridValue/nil
```

Description

Returns the value set by `sySetXPinGrid`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

f_gridValue	Grid value that indicates the X constraint on the pin locations.
`nil`	No value has been set, therefore no restrictions are used.

Example

```
syGetXPinGrid( cellA )
     => 5.0
```

Places pins such that the lower-left corner of each pin region (that is, connection region) has an X coordinate that is an exact multiple of 5.

syGetXPinGridOffset

```
syGetXPinGridOffset(
     d_cvId
     )
     => f_gridOffsetValue/nil
```

Description

Returns the value set by `sySetXPinGridOffset`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

f_gridOffsetValue	Offset of the horizontal pin grid.
`nil`	No value has been set, therefore no offset is used.

Example

```
syGetXPinGridOffset( cellA )
     => 3.0
```

Offset for the horizontal spacing of the grid used for placement of pins in `cellA` is 3 user units.

syGetYAnchorBoundary

```
syGetYAnchorBoundary(
     d_cvId
     )
     => t/nil
```

Description

Returns the value set by `sySetYAnchorBoundary`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

`t`	Cannot move pins on vertical boundary edges.
`nil`	Can move pins on vertical boundary edges.

Example

```
syGetYAnchorBoundary( cellA )
     => nil
```

Can move pins on vertical boundary edges of `cellA`.

syGetYPinGrid

```
syGetYPinGrid(
     d_cvId
     )
     => f_gridValue/nil
```

Description

Returns the value set by `sySetYPinGrid`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

f_gridValue	Grid value indicating the Y constraint on the pin locations.
`nil`	No value has been set, therefore no restrictions are used.

Example

```
syGetYPinGrid( cellA )
     => 5.0
```

Places pins such that the lower-left corner of each pin region (that is, connection region) has a Y coordinate that is an exact multiple of 5.

syGetYPinGridOffset

```
syGetYPinGridOffset(
     d_cvId
     )
     => f_gridOffsetValue/nil
```

Description

Returns the value set by `sySetYPinGridOffset`.

Arguments

d_cvId	Database object ID of the cellview.

Value Returned

f_gridOffsetValue	Offset of the vertical pin grid.
`nil`	No value has been set, therefore no offset is used.

Example

```
syGetYPinGridOffset( cellA )
     => 3.0
```

Offset for the vertical spacing of the grid used for placement of pins in `cellA` is 3 user units.

syReadAlignmentConstraint

```
syReadAlignmentConstraint(
     d_constraint
     )
     => (l_attribList)
```

Description

Returns a list of the attributes of a given alignment constraint. You can extract information from the list with the `assoc()` function.

Arguments

d_constraint	Database object ID of an alignment constraint returned by `syGetAlignmentConstraint()`.

Value Returned

A list containing the following items:

`direction X`	Horizontal direction of the constraint.
`direction Y`	Vertical direction of the constraint.
`objects (`*d_objId x_layer g_edge n_offset ...*`)`	Lists of attributes for each object in the constraint group. *d_objId* is the database object ID in each list. *x_layer* is the layer number relative to which the constraint is measured; it is significant only when the object is a component. (A value of `0` indicates that the object is not a component or that the constraint is measured relative to a component bounding box.) *g_edge* shows which edge is aligned. *n_offset* is the constraint offset distance.
`unrelaxable t`	Compactor cannot relax the constraint.

`unrelaxable nil`	Compactor can relax the constraint.
`(active t)`	Constraint is active.
`(active nil)`	Constraint is inactive.

Example

```
X_ALIGN = syGetAlignmentConstraint( cv ?direction 'X )
=> (db:20106428 db:20106128)

syReadAlignmentConstraint( car(X_ALIGN) )
=> ( (direction X)
     (objects ((db:20007036 5 left 0.0)
          (db:20011884 5 right 10.000000)))
     (unrelaxable nil) (active t))
```

Reports that the first horizontal (X) alignment constraint in the cellview `cv` is an active, relaxable constraint aligning the left edge of layer 5 on the first component to the right edge of layer 5 on the second component, with an offset distance of 10 user units.

syReadMagneticConstraint

```
syReadMagneticConstraint(
     d_constraint
     )
     => (l_attribList)
```

Description

Returns a list of the attributes of a given magnetic constraint. You can extract information from the list with the `assoc()` function.

Arguments

d_constraint	Magnetic constraint database object ID returned by `syGetMagneticConstraint()`.

Value Returned

A list containing the following items:

`direction X` Horizontal direction of the constraint.

`direction Y` Vertical direction of the constraint.

`object1(`*d_objId x_layer g_edge*`)`
Attributes of the first object in the constraint group.

d_objId is the database object ID.

x_layer is the layer number relative to which the constraint is measured; it is significant only when the object is a component. (A value of `0` indicates that the object is not a component or that the constraint is measured relative to a component bounding box.)

g_edge shows which edge is referenced in the constraint. Valid Values: `left`, `centerV`, `right` for X constraints; `bottom`, `centerH`, `top` for Y constraints; or the edge midpoint coordinates if the constraint is measured relative to the edge of a polygon in the object

`object2(`*`d_objId x_layer g_edge`*`)`
Attributes of the second object in the constraint group.

`attract t` Constraint attracts.

`attract nil` Constraint repulses.

`unrelaxable t` Compactor cannot relax the constraint.

`unrelaxable nil` Compactor can relax the constraint.

`active t` Constraint is active.

`active nil` Constraint is inactive.

Example

```
ALL_MAG = syGetMagneticConstraint( cv )
=> (db:20108060 db:20107708 db:20107504)

syReadMagneticConstraint( car(ALL_MAG) )
=> ( (direction Y) (object1 (db:20010332 0 centerH))
     (object2 (db:20107916 0 bottom))
     (attract nil) (unrelaxable nil) (active t))
```

Reports that the first magnetic constraint in the cellview `cv` is an active, relaxable, repulsive magnetic constraint in the Y direction between the horizontal centerline of the first object and the bottom edge of the second object.

syReadSeparationConstraint

```
syReadSeparationConstraint(
     d_constraint
     )
     => (l_attribList)
```

Description

Returns a list of the attributes of a given separation constraint. You can extract information from the list with the `assoc()` function.

Arguments

d_constraint	Separation constraint database object ID returned by `syGetSeparationConstraint()`.

Value Returned

A list containing the following items:

`direction X` Horizontal direction of the constraint.

`direction Y` Vertical direction of the constraint.

`object1(`*d_objId x_layer g_edge*`)`
Attributes of the first object in the constraint group.

d_objId is the database object ID.

x_layer is the layer number relative to which the constraint is measured; it is significant only when the object is a component. (A value of `0` indicates that the object is not a component or that the constraint is measured relative to a component bounding box.)

g_edge shows which edge is used in the constraint.
Valid Values: `left`, `centerV`, `right` for X constraints; `bottom`, `centerH`, `top` for Y constraints; or the edge midpoint coordinates if the constraint is measured relative to the edge of a polygon in the object

`object2(`*`d_objId x_layer g_edge`*`)`
Attributes of the second object in the constraint group.

`separation (`*`n_min n_max`*`)`
Minimum and maximum separation values. These are signed values, not magnitudes. The values are measured from the first object to the second object, so the order in which the objects were selected affects the signs of the values. For examples of how the signs are determined, see the *Virtuoso Compactor Reference Manual*.

`ignoreDRC t`	Compactor can ignore design rules.
`ignoreDRC nil`	Compactor must obey design rules.
`unrelaxable t`	Compactor cannot relax the constraint.
`unrelaxable nil`	Compactor can relax the constraint.
`active t`	Constraint is active.
`active nil`	Constraint is inactive

Example

```
Y_SEP = syGetSeparationConstraint( cv ?direction 'Y )
=> (db:20108580 db:20107320)

syReadSeparationConstraint(car( Y_SEP) )
=> ( (direction Y) (object1 (db:20108292 0 (21.377001
       5.318000)))
     (object2 (db:20011668 0 bottom))
     (separation (0.0 20.000000)) (ignoreDRC nil)
     (unrelaxable t) (active t))
```

Reports that the first vertical (Y) separation constraint in cellview `cv` is an active, unrelaxable constraint between a polygon edge with midpoint (21.377001 5.318000) and the bottom edge of a second object. The separation constraint value is the range 0.0 to 20.0. This constraint does not override the design rules.

syReadWireJogConstraint

```
syReadWireJogConstraint(
     d_constraint
     )
     => (l_attribList)
```

Description

Returns a list of the attributes of a wire-jog constraint. You can extract information from the list with the `assoc()` function.

Arguments

d_constraint	Wire-jog constraint database object ID returned by `syGetWireJogConstraint()`.

Value Returned

`direction X`	Horizontal direction of the constraint.
`direction Y`	Vertical direction of the constraint.
`path` *d_pathId*	Path ID for this wire.
`point` *p_point*	Coordinates of the jog.
`active t`	Constraint is active.
`active nil`	Constraint is inactive.

Example

```
ALL_JOG = syGetWireJogConstraint(cv)
=> (db:20105780 db:20105152 db:20105532)

syReadWireJogConstraint(car(ALL_JOG))
=> ( (direction Y) (path db:20010620) (point (35.611000
     40.100000)) (active t))
```

Reports that the first horizontal (Y) direction wire-jog insertion in the cellview `cv` is an active constraint at point (35.611000 40.100000) on the path db:20010620.

syReadWireWidthConstraint

```
syReadWireWidthConstraint(
     d_constraintId
     )
     => (l_attribList)
```

Description

Returns a list of wire-width constraint attributes. You can extract information from the list with the `assoc()` function.

Arguments

d_constraintId	Wire width constraint ID returned by `syGetWireWidthConstraint()`.

Value Returned

A list containing the following items:

`direction X`	Horizontal direction of the constraint.
`direction Y`	Vertical direction of the constraint.
`path` *d_pathId*	Path ID for this wire.
`width` *n_value*	Wire width constraint value.
`active t`	Constraint is active.
`active nil`	Constraint is inactive

Example

```
X_WIRE = syGetWireWidthConstraint( cv ?direction 'X )
=> (db:20104028 db:20104904)

syReadWireWidthConstraint( car(X_WIRE) )
=> ( (direction X) (path db:20006112) (width 4.500000)
     (active t))
```

Reports that the first X wire width constraint in the cellview `cv` is an active constraint setting the width of path db:20006112 to 4.5 user units.

syRefCellConstraint

```
syRefCellConstraint(
     d_srcCvId
     d_destCvId
     )
     => t/nil
```

Description

Matches pins, wells, and cell heights or widths for two cells by setting constraints on the destination cell. The constraints are set so that the cell dimensions and the pin and well locations on all four edges of the cell boundary match those of the source cell after compaction.

You can use this function to prepare cells for abutment or to compact cells to match certain defined interface requirements. For this function to work properly, the two cells must have the same number and type of pins and wells on corresponding edges. Only wells that touch or extend beyond a cell edge are considered.

Arguments

d_srcCvId	Database ID of the source cell from which the dimensions and pin and well locations are copied.
d_destCvId	Database ID of the destination cell for which constraints are created.

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
syRefCellConstraint( cellA cellB )
```

Creates in `cellB` a set of constraints on all four edges that specify the height, width, and pin and well locations to be achieved after compaction. The constraints are based on corresponding information in `cellA`.

syRefEdgeConstraint

```
syRefEdgeConstraint(
     l_srcEdge
     l_destEdge
     )
     => t/nil
```

Description

Matches pins, well, and cell heights or widths on two cells by setting constraints on one of the cells (*l_destEdge*, the slave cell) so that the spacing between the pins, wells, and both corners of the cell will match that of the other cell (*l_srcEdge*, the master cell) after compaction.

Arguments

l_srcEdge	Edge of the master cell to be used for reference, in the following format: `list(` *d_srcCVId s_side* `)` where *d_srcCVId* is the database object ID of the master cell. *s_side* is the reference edge of the master cell. Valid Values: `'top`, `'bottom`, `'left`, `'right`
l_destEdge	Edge of the slave cell to be constrained to match *l_srcEdge* after compaction. This is given in the same format as *l_srcEdge.*

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
syRefEdgeConstraint( '(object1 'right) '(object2 'left) )
```

Creates a set of constraints on `object2` so that the relative positions of pins and wells on its left edge match those on the right edge of `object1`, and its cell height matches that of `object1`.

sySetAutoJog

```
sySetAutoJog(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Allows or inhibits automatic jog insertion for the specified cellview.

Note: When *g_yesOrNo* is `nil`, the settings of the `sySetAutoJog` function is ignored.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether automatic jog insertion is allowed or inhibited. Valid Values: `t` to allow automatic jog insertion, or `nil` to inhibit automatic jog insertion

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetAutoJog( mycell nil )
```

Inhibits automatic jog insertion for `mycell`.

sySetAutoJogByLayer

```
sySetAutoJogByLayer(
     d_cvId
     g_layer
     g_yesOrNo
     )
     => t/nil
```

Description

Allows or inhibits automatic jog insertion for the wires on the specified layer.

Note: When automatic jog insertion is inhibited globally with `sySetAutoJog()`, the compactor ignores the settings of this function. Use this function to limit the scope of automatic jog insertion when auto jog is enabled globally.

Arguments

d_cvId	Database object ID of the cellview.
g_layer	Layer number, layer name, or layer-purpose pair. When you specify only the layer number or layer name, the purpose value defaults to "`drawing`". To specify a purpose other than "`drawing`", use the layer-purpose format; for example, `list("metal1" "drawing2")`. Valid Values: one of the layer-purpose pairs listed in the symbolic wire declaration portion of the technology file. For more information, see the description of `techSetSymWire` in the *Technology File and Display Resource File SKILL Reference Manual*.
g_yesOrNo	Enables or inhibits automatic jog insertion for the specified layer. Valid Values: `t` to enable automatic jog insertion, or `nil` to inhibit automatic jog insertion

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetAutoJogByLayer( mycell "ndiff" nil )
```

Inhibits automatic jog insertion for wires on the `ndiff` drawing layer in the cellview `mycell`.

sySetAutoJogByNet

```
sySetAutoJogByNet(
     d_netId
     g_yesOrNo
     )
     => t/nil
```

Description

Allows or inhibits automatic jog insertion for the specified net.

Note: When automatic jog insertion is inhibited globally with `sySetAutoJog()`, the compactor ignores the settings of this function. Use this function to limit the scope of automatic jog insertion when auto jog is enabled globally.

Arguments

d_netId	Database object ID of the net.
g_yesOrNo	Allows or inhibits automatic jog insertion for the specified net. Valid Values: `t` to allow automatic job insertion, `nil` to inhibit automatic jog insertion

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetAutoJogByNet( netA nil )
```

Inhibits automatic jog insertion for `netA`.

sySetAutoJogByWire

```
sySetAutoJogByWire(
     d_wireId
     g_yesOrNo
     )
     => t/nil
```

Description

Allows or inhibits automatic jog insertion for the specified wire.

Note: When automatic jog insertion is inhibited globally with `sySetAutoJog()`, the compactor ignores the settings of this function. Use this function to limit the scope of automatic jog insertion when auto jog is enabled globally.

Arguments

d_wireId	Database object ID of the wire.
g_yesOrNo	Allows or inhibits automatic jog insertion for the specified wire. Valid Values: `t` to allow automatic job insertion, `nil` to inhibit automatic jog insertion

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetAutoJogByWire( wireA nil )
```

Inhibits automatic jog insertion for `wire`A.

sySetBottomAbutted

```
sySetBottomAbutted(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Sets the bottom edge of the cell for abutment.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the bottom edge is set for abutment. Valid Values: `t` indicates that the bottom edge is set for abutment, `nil` indicates that the bottom edge is not set for abutment

Value Returned

`t`	The bottom edge was set.
`nil`	Function failed.

Example

```
sySetBottomAbutted( cellA nil )
```

Sets the bottom edge of `cellA` so that it is not an abutment edge.

sySetEnv

```
sySetEnv(
     t_name
     g_value
     )
     => t/nil
```

Description

Sets the compactor environment variable supplied in *t_name* to the value supplied in *g_value*.

For a complete description of the compactor environment variables, see "Introduction" on page 421.

Arguments

t_name	Name of the compactor environment variable you want to set. Valid Values: `cpCpathLPP`, `cpAuxLPP`, `cpOutlineLPP`, `hilite1LPP`, `hilite2LPP`, `hilite3LPP`, `hilite4LPP`, `useDBox`, `arrowScale`, `zoomScale`
g_value	Value of the variable *t_name*. Valid Values: any value that matches the value type defined for the variable

Value Returned

`t`	The variable was set.
`nil`	Function failed.

Example

```
sySetEnv( "zoomScale" 2.0 )
```

Sets the zoom factor to 2 in the current window.

sySetFixLowLeftBoundary

```
sySetFixLowLeftBoundary(
     d_cvId
     g_value
     )
     => t/nil
```

Description

Specifies whether to use the lower-left corner coordinates of the source cellview as the lower-left corner coordinates of the compacted cellview. Otherwise, the compactor moves the lower-left corner of the compacted cellview to (0,0).

Arguments

d_cvId	Database object ID of the cellview.
g_value	Indicates where to place the lower-left corner of the compacted cellview. Valid Values: `t` places the lower-left corner at the source cellview lower-left corner, `nil` moves the lower-left corner to (0,0)

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetFixLowLeftBoundary( cellA nil )
```

Moves the lower-left corner of the compacted `cellA` to the coordinate (0,0).

sySetForceChildCellCompact

```
sySetForceChildCellCompact(
     d_cvId
     g_value
     )
     => t/nil
```

Description

Specifies whether the compactor should compact all child cells during hierarchical compaction. During hierarchical compaction, the compactor normally compacts only those child cells not previously compacted.

Arguments

d_cvId	Database object ID of the cellview.
g_value	Indicates whether the compactor should compact all child cells. Valid Values: `t` compacts all child cells, `nil` does not compact all child cells.

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetForceChildCellCompact( cellA t )
```

Compacts all child cells when performing hierarchical compaction on `cellA`.

sySetGeoOptions

```
sySetGeoOptions(
     d_cvId
     x_stopLevel
     g_smashDevice
     t_mode
     )
     => t/nil
```

Description

Sets the stop level, smash device flag, and conversion options for `sySymToGeo`.

Arguments

d_cvId	Database object ID of the cellview.
x_stopLevel	Hierarchical level at which the compactor stops converting symbols and cell instances to polygon data. Valid Values: any positive integer indicating a hierarchical level in your design
g_smashDevice	Specifies whether the compactor converts symbols and cell instances to polygon data down to the hierarchical level specified by *x_stopLevel*. Valid Values: `t` converts symbols and cell instances to polygons, `nil` does not
t_mode	Conversion options that control the shape of the output polygon. Valid Values: If *g_smashDevice* is `nil`, `fill notch`, `fill gap`. If *g_smashDevice* is `t`: `fill notch`, `fill gap`, `merge polygon`, `no merge`. `fill notch` fills any notches smaller than the design rule for the layer or mask layer. `fill gap` fills any notches smaller than the design rule for the layer or mask layer and fills any gaps smaller than the design rule between two polygons of the same layer or the same mask layer.

`merge polygon` causes abutting or overlapping polygons of the same layer to merge into a single polygon. The output cell produced by this option requires less storage space on a disk.

`no merge` causes abutting or overlapping polygons of the same layer to remain separate.

Value Returned

`t`	The stop level was set.
`nil`	Function failed.

Example

```
sySetGeoOptions( cellAId 10 t "fill gap" )
```

Sets the stop level, device flag, and conversion values for `cellAId`. For `cellAId`, does the following:

- Converts symbols and cell instances to polygon data to the tenth level of the design hierarchy
- Fills any notches smaller than the design rule for the layer or mask layer
- Fills any gaps smaller than the design rule between two polygons of the same layer or the same mask layer

sySetHardCell

```
sySetHardCell(
     d_inputCVId
     g_yesOrNo
     )
     => t/nil
```

Description

Controls whether hierarchical compaction compacts a specified cell and its subcells.

Arguments

d_inputCVId	Database object ID of the cellview to be set to or from a hard cell.
g_yesOrNo	Specifies whether hierarchical compaction compacts the specified cell and its subcells. Valid Values: `t` does not compact the cell and its subcells; `nil` compacts the cell and its subcells during hierarchical compaction

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetHardCell( cellA nil )
```

Compacts `cellA` and its subcells during hierarchical compaction.

sySetInitialDirection

```
sySetInitialDirection(
     d_cvId
     t_XOrY
     )
     => t/nil
```

Description

Sets the initial direction of compaction.

Arguments

d_cvId	Database object ID of the cellview.
t_XOrY	Direction for the initial pass. Valid Values: X (horizontal), Y (vertical)

Value Returned

t	The initial direction was set.
nil	Function failed.

Example

```
sySetInitialDirection( cellA "X" )
```

Sets the initial direction of compaction of cellA to horizontal.

sySetLeftAbutted

```
sySetLeftAbutted(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Sets the left edge of the cell for abutment.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the left edge is set for abutment. Valid Values: `t` indicates that the left edge is set for abutment; `nil` indicates that the left edge is not set for abutment

Value Returned

`t`	The left edge was set.
`nil`	Function failed.

Example

```
sySetLeftAbutted( cellA nil )
```

Sets the left edge of `cellA` so that it is not an abutment edge.

sySetMaxAutoJogs

```
sySetMaxAutoJogs(
     d_cvId
     n_value
     )
     => t/nil
```

Description

Sets the maximum number of jogs that the compactor can add automatically to any one wire.

Arguments

d_cvId	Database object ID of the cellview.
n_value	The maximum number of jogs. Valid Values: any integer

Value Returned

`t`	The maximum number was set.
`nil`	Function failed.

Example

```
sySetMaxAutoJogs( cellA 3 )
```

Allows the compactor to insert up to three jogs automatically in any one wire in `cellA`.

sySetMaxIterations

```
sySetMaxIterations(
     d_cvId
     n_maxIter
     [n_minChange]
     )
     => t/nil
```

Description

Sets the maximum number of iterations executed by the compactor. Optionally sets the percentage of area the cellview must decrease with each iteration to continue compaction. Compaction stops when either termination criterion is met.

One iteration is a compaction pass in both directions. Therefore, to set an odd compaction sequence, set *n_maxIter* to a value ending with .5. For example, for compacting horizontally, then vertically, then horizontally, you would set *n_maxIter* to 1.5.

Compactor checks the termination criteria each time it completes a full iteration. If the area of the design has been reduced by an amount equal to or less than the percentage specified in *n_minChange*, or if compactor has compacted the number of times indicated by *n_maxIter*, compaction stops. If you do not specify *n_minChange*, compactor uses 0 percent as the termination criterion.

The initial direction of compaction is determined by the value set by the `sySetInitialDirection()` function.

Arguments

d_cvId	Database object ID of the cellview.
n_maxIter	Maximum number of iterations. Valid Values: any positive number that is a multiple of 0.5
n_minChange	Percent of area reduction at which compaction stops. Valid Values: any number greater than or equal to 0 and less than 100

Value Returned

`t`	The maximum number was set.

`nil` Function failed.

Example

```
sySetMaxIterations( cellB 10.0 2.5 )
```

Compacts the cellview in both directions until one of the following conditions is satisfied:

- A total of 10 compaction iterations have been carried out
- The decrease in area between iterations is less than or equal to 2.5 percent.

sySetPeelbackDistance

```
sySetPeelbackDistance(
     d_cvId
     n_distance
     )
     => t/nil
```

Description

Sets the maximum length of an unconnected segment you want deleted. Any unconnected wire segment with a length less than or equal to *n_distance* is deleted.

Arguments

d_cvId	Database object ID of the cellview.
n_distance	Wire peelback distance in user units.

Value Returned

t	The maximum length was set.
nil	Function failed.

Example

```
sySetPeelbackDistance( cellA 3.0 )
```

Deletes unconnected wire segments less than or equal to 3 user units in length.

sySetPFrame

```
sySetPFrame(
     d_cvId
     t_viewName
     )
     => t/nil
```

Description

Sets the viewname for the protection frames to be used for all child cells of a cellview to be compacted.

Arguments

d_cvId	Database object ID of the parent cellview for which the protection frame viewname will be used during its compaction.
t_viewName	Viewname for the protection frames to be used.

Value Returned

`t`	The viewname was set.
`nil`	Function failed.

Example

```
sySetPFrame( cellA "frame" )
```

Sets the viewname for the protection frames used for child cells during compaction of `cellA` to `frame`. When you compact `cellA` with the option set to use protection frames, any child cell with a view named `frame` is replaced with the `frame` view.

sySetPostProcess

```
sySetPostProcess(
     d_cvId
     t_WlmEnum
     )
     => t/nil
```

Description

Specifies whether the compactor should minimize total wire length during compaction. If you turn off wire-length minimization, you can change the default compaction direction.

Arguments

d_cvId	Database object ID of the cellview.
t_WlmEnum	Post processing to be used. Valid Values: `wwlm`, `min`, `max` Default: `wwlm` `wwlm` minimizes total wire length in the compacted output based on the resistivity factor set for each routing layer in the technology file and then compacts toward the lower-left corner of the cellview. `min` compacts toward the lower-left corner of the cellview without minimizing wire length. `max` compacts toward the upper-right corner of the cellview without minimizing wire length.

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetPostProcess( cellA "min" )
```

Sets the method of wire length minimization for `cellA` so that compactor places objects as close as possible to the left or bottom of the compacted cellview.

sySetPreserveCellRow

```
sySetPreserveCellRow(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Specifies whether the compactor must preserve the structure of cell rows.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the compactor must preserve cell row structure. Valid Values: `t` preserves the cell row structure, `nil` changes the cell row structure

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetPreserveCellRow( cellA t )
```

Preserves the cell row structure for `cellA`.

sySetPreservePathWidth

```
sySetPreservePathWidth()
```

Description

Replaced by sySetPreserveWireWidth.

sySetPreserveWireWidth

```
sySetPreserveWireWidth(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Specifies whether the compactor must preserve the wire (path) width specified when the wire was drawn. The `sySetPreserveWireWidth()` function is a replacement for the `sySetPreservePathWidth()` function.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the compactor must preserve the wire (path) width specified when the wire was drawn. Valid Values: `t` preserves the wire width, `nil` changes the wire width Default: `nil`

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetPreserveWireWidth( cellA t )
```

Preserves the wire (path) widths specified when the wires were drawn for `cellA`.

sySetRightAbutted

```
sySetRightAbutted(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Sets the right edge of the cell for abutment.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the right edge is set for abutment. Valid Values: `t` indicates that the right edge is set for abutment; `nil` indicates that the right edge is not set for abutment

Value Returned

`t`	The right edge was set.
`nil`	Function failed.

Example

```
sySetRightAbutted( cellA nil )
```

Sets the right edge of `cellA` so that it is not an abutment edge.

sySetTopAbutted

```
sySetTopAbutted(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Sets the top edge of the cell for abutment.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the top edge is set for abutment. Valid Values: `t` indicates that the top edge is set for abutment, `nil` indicates that the top edge is not set for abutment

Value Returned

`t`	The top edge was set.
`nil`	Function failed.

Example

```
sySetTopAbutted( cellA nil )
```

Sets the top edge of `cellA` so that it is not an abutment edge.

sySetUseChildBdy

```
sySetUseChildBdy(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Specifies whether the boundaries of all child cells with boundaries are used as abstractions during compaction.

Arguments

d_cvId	Database object ID of the cellview to be compacted.
g_yesOrNo	Indicates whether the boundaries are used as abstractions. Valid Values: `t` indicates that the boundaries are used as abstractions, `nil` indicates that the boundaries are not used as abstractions

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetUseChildBdy( cellA t )
```

Indicates that the boundaries of all child cells in `cellA` are used as abstractions during compaction.

sySetXAnchorBoundary

```
sySetXAnchorBoundary(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Specifies whether the compactor can move the pin locations on horizontal cell boundary edges.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the compactor can move pins. Valid Values: `t` indicates that the compactor cannot move pins; `nil` indicates that the compactor can move pins

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetXAnchorBoundary( cellA t )
```

Prevents the compactor from changing the location of pins on horizontal edges of the boundary of `cellA`.

sySetXPinGrid

```
sySetXPinGrid(
     d_cvId
     f_gridValue
     )
     => t/nil
```

Description

Sets the X spacing of the grid used for placement of pins. The compactor places pins so that the lower-left corner of each pin region (that is, connection region) has an X coordinate that is an exact multiple of *f_gridValue.*

Arguments

d_cvId	Database object ID of the cellview.
f_gridValue	Grid value that indicates the X constraint on the pin locations.

Value Returned

t	The spacing was set.
nil	Function failed.

Example

```
sySetXPinGrid( cellA 5.0 )
```

Sets spacing of the pin grid in the X direction to 5.0 for cellA.

sySetXPinGridOffset

```
sySetXPinGridOffset(
     d_cvId
     f_gridOffsetValue
     )
     => t/nil
```

Description

Sets the offset for the horizontal spacing of the grid used for placement of pins. To set the subsequent spacing, use `sySetXPinGrid()`.

Arguments

`d_cvId`	Database object ID of the cellview.
`f_gridOffsetValue`	Offset of the horizontal pin grid.

Value Returned

`t`	The offset was set.
`nil`	Function failed.

Example

```
sySetXPinGridOffset( cellA 3.0 )
```

Sets the offset for the horizontal spacing of the grid used for placement of pins to 3 user units.

sySetYAnchorBoundary

```
sySetYAnchorBoundary(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Specifies whether the compactor can move the pin locations on vertical cell boundary edges.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Indicates whether the compactor can move pins. Valid Values: `t` indicates that the compactor cannot move pins; `nil` indicates that the compactor can move pins

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetYAnchorBoundary( cellA t )
```

Prevents the compactor from changing the location of pins on vertical edges of the boundary of `cellA`.

sySetYFirst

```
sySetYFirst(
     d_cvId
     g_yesOrNo
     )
     => t/nil
```

Description

Specifies whether the compactor compacts the specified cellview in the Y direction first or in the X direction first.

Arguments

d_cvId	Database object ID of the cellview.
g_yesOrNo	Specifies whether compactor compacts in the Y direction first. Valid Values: `t` compacts first in the Y direction, then in the X direction; `nil` compacts first in the X direction, then in the Y direction

Value Returned

`t`	Function ran successfully.
`nil`	Function failed.

Example

```
sySetYFirst( cellA t )
```

Compacts `cellA` in the Y direction first, then in the X direction.

sySetYPinGrid

```
sySetYPinGrid(
     d_cvId
     f_gridValue
     )
     => t/nil
```

Description

Sets the Y spacing of the grid used for placement of pins. The compactor places pins so that the lower-left corner of each pin region (that is, connection region) has a Y coordinate that is an exact multiple of *f_gridValue.*

Arguments

d_cvId	Database object ID of the cellview.
f_gridValue	Grid value that indicates the Y constraint on the pin locations.

Value Returned

t	The spacing was set.
nil	Function failed.

Example

```
sySetYPinGrid( cellA 5.0 )
```

Sets spacing of the pin grid in the Y direction to 5.0 for cellA.

sySetYPinGridOffset

```
sySetYPinGridOffset(
     d_cvId
     f_gridOffsetValue
     )
     => t/nil
```

Description

Sets the offset for the vertical spacing of the grid used for placement of pins. To set the subsequent spacing, use `sySetYPinGrid()`.

Arguments

d_cvId	Database object ID of the cellview.
f_gridOffsetValue	Offset of the vertical pin grid.

Value Returned

`t`	The offset was set.
`nil`	Function failed.

Example

```
sySetYPinGridOffset( cellA 3.0 )
```

Sets the offset for the vertical spacing of the grid used for placement of pins to 3 user units.

sySymToGeo

```
sySymToGeo(
     d_inputCVId
     l_outputCVName
     )
     => d_outputCVId/nil
```

Description

Converts a symbolic layout to a polygon layout. The `sySymToGeo()` function is a replacement for the `sySymToPolygon()` function.

Arguments

d_inputCVId	Database object ID of the input cellview.
l_outputCVName	Cellname and viewname of the output cellview.

Value Returned

d_outputCVId	Database object ID of the output cell.
`nil`	Function failed.

Example

```
sySetGeoOptions(cellAId 10 t "fill gap")
outputCellViewID = sySymToGeo(cellAId
list("cellB" "layout"))
```

Creates a polygon layout cellview with the cellname `cellB` and the viewname `layout` from the symbolic layout cellview `cellAId`. For `cellAId`, does the following:

- Converts symbols and cell instances to polygon data to the tenth level of the design hierarchy
- Fills any notches smaller than the design rule for the layer or mask layer
- Fills any gaps smaller than the design rule between two polygons of the same layer or the same mask layer

sySymToPolygon

`sySymToPolygon()`

Description

Replaced by `sySymToGeo`.

syUserAllDel

```
syUserAllDel(
     d_cvId
     )
     => t/nil
```

Description

Deletes all user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire width, and wire jog constraints. This function does not delete auto jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which constraints are to be deleted.

Value Returned

`t`	The constraints were deleted.
`nil`	Function failed.

Example

```
syUserAllDel( cellA )
```

Deletes all user constraints in cellview `cellA`.

syUserAllOff

```
syUserAllOff(
     d_cvId
     )
     => t/nil
```

Description

Deactivates all user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not deactivate auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which constraints are to be deactivated.

Value Returned

`t`	The constraint was deactivated.
`nil`	Function failed.

Example

```
syUserAllOff( cellA )
```

Deactivates all user constraints in cellview `cellA`.

syUserAllOn

```
syUserAllOn(
     d_cvId
     )
     => t/nil
```

Description

Activates all user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not activate auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which constraints are to be activated.

Value Returned

t	The constraint was activated.
nil	Function failed.

Example

```
syUserAllOn( cellA )
```

Activates all user constraints in cellview cellA.

syUserAllXDel

```
syUserAllXDel(
     d_cvId
     )
     => t/nil
```

Description

Deletes all horizontal user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not delete auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which horizontal constraints are to be deleted.

Value Returned

t	The constraints were deleted.
nil	Function failed.

Example

```
syUserAllXDel( cellA )
```

Deletes all horizontal user constraints in cellview cellA.

syUserAllXOff

```
syUserAllXOff(
     d_cvId
     )
     => t/nil
```

Description

Deactivates all horizontal user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not deactivate auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which horizontal constraints are to be deactivated.

Value Returned

`t`	The constraint was deactivated.
`nil`	Function failed.

Example

```
syUserAllXOff( cellA )
```

Deactivates all horizontal user constraints in cellview `cellA`.

syUserAllXOn

```
syUserAllXOn(
     d_cvId
     )
     => t/nil
```

Description

Activates all horizontal user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not activate auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which horizontal constraints are to be activated.

Value Returned

`t`	The constraint was activated.
`nil`	Function failed.

Example

```
syUserAllXOn( cellA )
```

Activates all horizontal user constraints in cellview `cellA`.

syUserAllYDel

```
syUserAllYDel(
     d_cvId
     )
     => t/nil
```

Description

Deletes all vertical user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not delete auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which vertical constraints are to be deleted.

Value Returned

`t`	The constraints were deleted.
`nil`	Function failed.

Example

```
syUserAllYDel( cellA )
```

Deletes all vertical user constraints in cellview `cellA`.

syUserAllYOff

```
syUserAllYOff(
     d_cvId
     )
     => t/nil
```

Description

Deactivates all vertical user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not deactivate auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which vertical constraints are to be deactivated.

Value Returned

`t`	The constraint was deactivated.
`nil`	Function failed.

Example

```
syUserAllYOff( cellA )
```

Deactivates all vertical user constraints in cellview `cellA`.

syUserAllYOn

```
syUserAllYOn(
     d_cvId
     )
     => t/nil
```

Description

Activates all vertical user constraints in the specified cellview. These constraints include alignment, separation, magnetic, wire-width, and wire-jog constraints. This function does not activate auto-jog constraints.

Arguments

d_cvId	Database object ID of the cellview for which vertical constraints are to be activated.

Value Returned;

t	The constraint was activated.
nil	Function failed.

Example

```
syUserAllYOn( cellA )
```

Activates all vertical user constraints in cellview cellAs.

4

Structure Compiler Functions

This section provides syntax, descriptions, and examples for the Cadence® SKILL functions associated with the Structure Compiler menu commands.

Introduction on page 538

- scAddCell on page 539
- scAddColumn on page 541
- scAddMaster on page 543
- scAddRow on page 544
- scBias on page 546
- scCellHeight on page 547
- scCellsPerColumn on page 548
- scCellsPerRow on page 550
- scCellWidth on page 552
- scChangeBias on page 553
- scChangeCell on page 554
- scChangeIcon on page 556
- scChangeMaster on page 558
- scChangeOrient on page 560
- scCompileArray on page 562
- scDeleteCell on page 564
- scDeleteColumn on page 566
- scDeletePlane on page 568

scDeleteRow on page 569

scDrawPlane on page 570

scGetPlanePoint on page 571

scIcon on page 572

scMaster on page 574

scOrient on page 576

scPlane on page 578

scPromotePin on page 579

scResizePlane on page 581

scStretchEmptyCell on page 582

scSwapCells on page 584

scSwapColumns on page 586

scSwapRows on page 588

scUnusedIcon on page 590

Introduction

This chapter provides syntax, descriptions, and examples for the SKILL functions associated with the Structure Compiler menu commands.

scAddCell

```
scAddCell(
     x_column:x_row
     t_icon
     t_direction
     [t_orientation]
     )
     => x_column:x_row/nil
```

Description

Adds a new cell to the array at a specified location.

Restrictions apply according to array type as follows:

Row-biased	Add cells only to the right or left of existing cells. Existing cells in a row move to the right to accommodate the new cell.
Column-biased	Add cells only above or below existing cells. Existing cells in a column move up to accommodate the new cell.
Unbiased	You cannot add individual cells.

Adding individual cells in row- or column-biased arrays might leave the array with ragged right or top edges. When this happens, `scAddCell` automatically fills the array with special padding cells to keep the array boundary rectangular. Arrays are always left- and bottom-justified.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column:x_row	Location of cell adjacent to which the new cell is inserted.
t_icon	Single-character icon for the new cell.
t_direction	Direction to add the new cell in relation to the cell specified in the first argument. Use `above` or `below` for a column-biased array, and `left` or `right` for a row-biased array.

t_orientation Orientation for the new cell. Valid values: `0`, `90`, `180`, `270`, `sideways`, `sideways&90`, `upsideDown`, `sideways&270`, `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, and `MXR90`.

Value Returned

x_column:x_row Location of the newly inserted cell.

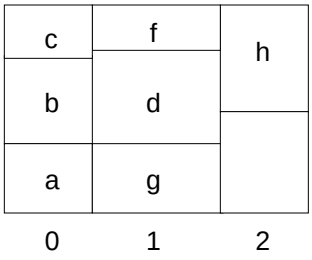
`nil` Indicates an error condition.

Example

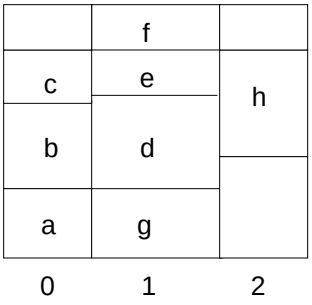
```
scPlane( geGetEditCellView( ) "myPlane"
     scAddCell( 1:2 "e" "below"))
```

Adds cell e below cell `f`, which was located at 1:2. Cell `f` moves up to accommodate the new cell. `scAddCell` automatically fills the array with pad cells to keep the boundary rectangular.

Before



After



scAddColumn

```
scAddColumn(
     x_column
     t_direction
     n_width
     )
     => x_column:x_row/nil
```

Description

Adds a column of empty cells in a column-biased array. When you add a new column to the left or right of an existing column (not permitted in row-biased arrays), the new column has the same number of cells as the column specified in the first argument. All cells in the new column are initially empty cells with the same height as the corresponding cells in the adjacent column. You can specify the width of the new cells in the width argument. The existing columns in the array move to the right when you add a new column.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column	Number of the column to which the new column is adjacent.
t_direction	Direction, `right` or `left`, to add the new column in relation to the column specified by the *x_column* argument.
n_width	Width of the new column.

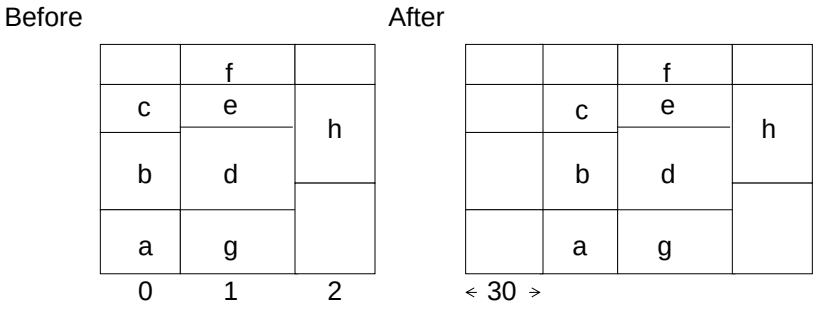
Value Returned

x_column:x_row	Location of the first cell in the new column.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     scAddColumn( 0 "left" 30))
```

`scAddColumn` adds a column of empty cells 30 user units wide to the left of column 0 and returns `0:0`, the location of the first cell in the new column.



scAddMaster

```
scAddMaster(
     t_master-name
     t_icon
     )
     => t_master-name/nil
```

Description

Enlarges the set of masters and/or icons of an array and updates the *masterList*. This function makes a change only within the database. It does not make any visible change to the array.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

t_master-name	Two-word string, such as `master1 layout`, identifying the cellname and view name of a master cell.
t_icon	Single-character icon representing the new master.

Value Returned

t_master-name	Cellname and view name of the master cell.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     scAddMaster( "master1 layout" "b")
```

`scAddMaster` adds `master1 layout` with icon b to the set of masters and icons available for use in cells of the array.

scAddRow

```
scAddRow(
     x_row
     t_direction
     n_height
     )
     => x_column:x_row/nil
```

Description

Adds a row of empty cells in a row-biased array. If you add a new row above or below an existing row (not permitted in column-biased arrays), the new row has the same number of cells as the adjacent row specified in the first argument. The cells in the new row are initially empty cells with the same width as the corresponding cells in the adjacent row; however, you can specify their height. You can then fill the new row using `scChangeCell`. The existing rows in the array move up when you add a new row.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_row	Number of the row to which the new row is adjacent.
t_direction	Direction, `above` or `below`, to add the new row in relation to the row specified by the *x_row* argument.
n_height	Height of the new row.

Value Returned

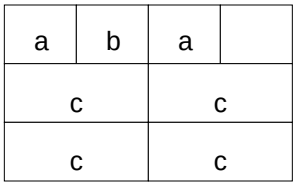
x_column:x_row	Location of the first cell in the new row.
`nil`	Indicates an error condition.

Example

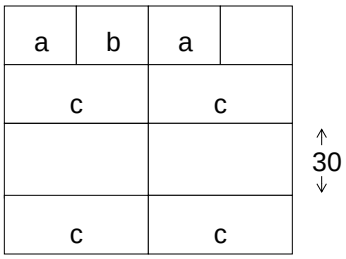
```
scPlane( geGetEditCellView( ) "myPlane"
     scAddRow( 0 "above" 30))
```

`scAddRow` adds a row of empty cells 30 user units high above row 0 and returns `0:1`, the location of the first cell in the new row.

Before



After



scBias

```
scBias( )
     => t_bias/nil
```

Description

Identifies the current bias of an array block.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

None.

Value Returned

`t_bias`	Returns one of the following values for *`t_bias`*:
`row`	if the array is row-biased,
`column`	if the array is column-biased,
`unbiased`	if the array is unbiased,
`nil`	Indicates an error condition

Example

```
scPlane( geGetEditCellView( ) "A" bias=scBias( ))
```

`scBias` assigns the current bias for array block A to the `bias` variable.

scCellHeight

```
scCellHeight(
     t_icon/x_column:x_row
     )
     => f_height/nil
```

Description

Takes an icon or a cell location as argument and returns the height of its corresponding master in user units.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

t_icon	Single-character cell icon.
x_column:*x_row*	Cell location.

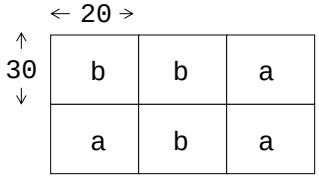
Value Returned

f_height	Height of the corresponding master in user units.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     height=scCellHeight( "b"))
```

For the array below, `scCellHeight` returns `30`, the height of cell b, which it assigns to the `height` variable.



scCellsPerColumn

```
scCellsPerColumn(
     x_column
     )
     => x_cells-in-the-column/nil
```

Description

Finds the number of cells in a specified column of an array.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column	Index of the column to check. If you are searching a row-biased array, 0 is the only valid value for this argument. `scCellsPerColumn (0)` returns the number of rows in the array.

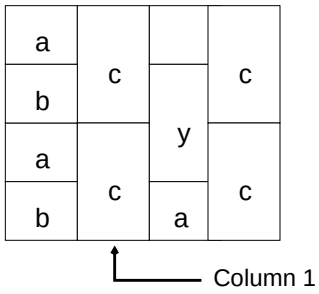
Value Returned

x_cells-in-the-column	Number of cells in the specified column.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     cells=scCellsPerColumn(1))
```

For the array below, `scCellsPerColumn` counts the number of cells in column 1 and returns a value of 2, which it assigns to the `cells` variable.



scCellsPerRow

```
scCellsPerRow(
     x_row
     )
     => x_cells-in-the-row/nil
```

Description

Finds the number of cells in a specified row of an array.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_row	Index of the row to check. If you are searching a column-biased array, 0 is the only valid value for this argument. `scCellsPerRow (0)` returns the number of columns in the array.

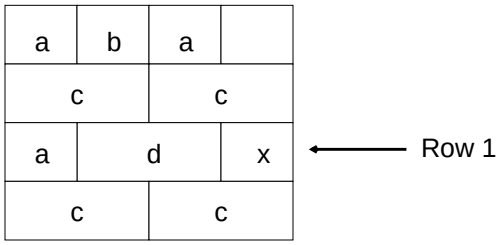
Value Returned

x_cells-in-the-row	Number of cells in the specified row.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     cells=scCellsPerRow(1))
```

For the array below, `scCellsPerRow` counts the number of cells in row 1 and returns a value of 3, which it assigns to the `cells` variable.



scCellWidth

```
scCellWidth(
     t_icon/x_column:x_row
     )
     => f_width/nil
```

Description

Takes an icon or a cell location as argument and returns the width of its corresponding master in user units.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

t_icon	Single-character cell icon.
x_column:*x_row*	Cell location.

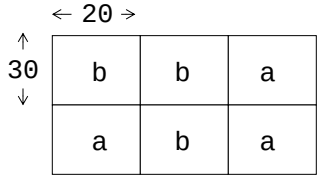
Value Returned

f_width	Width of the corresponding master in user units.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     width=scCellWidth( "b"))
```

For the array below, `scCellWidth` returns `20`, the width of cell b, which it assigns to the `width` variable.



scChangeBias

```
scChangeBias(
     t_bias
     )
     => t/nil
```

Description

Alters the bias of an array. Call this function only within the `scPlane` function or in the procedure field of a template plane. `scChangeBias` fails if the array does not meet the criteria for the intended bias. This function makes a change only within the database. It does not make any visible change to the array.

Prerequisites

Before you can replace a cell with another of a different width, you must explicitly change the array bias to `row`. You cannot change the bias directly from row-biased to column-biased, or from column-biased to row-biased, in the same function call. You must change the bias to unbiased as an intermediate step. If the array block meets the criteria for an unbiased array after returning from the `scPlane` function, it becomes an unbiased array even if you changed its bias using this function call.

Arguments

t_bias	New bias for the array. Valid Values: `row`, `column`, and `unbiased`.

Value Returned

`t`	The function completed successfully.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     scChangeBias("row"))
```

`scChangeBias` changes the bias of the `myPlane` array to `row`.

scChangeCell

```
scChangeCell(
    x_column:x_row
    t_icon
    [t_orientation]
    )
    => x_column:x_row/nil
```

Description

Changes the master represented in an individual cell.

The change depends on the array type as follows:

Unbiased	The new master must be exactly the same height and width as the previous one (after copying the orientation) so that the row and column structure of the array is unaffected by the change.
Row-biased	The new master must be the same height but does not need to be the same width as the previous one.
Column-biased	The new master must be the same width but does not need to be the same height as the previous one.

You can also replace an existing cell with an empty cell. The empty cell assumes the size of the cell it replaces.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column:*x_row*	Location of an individual cell.
t_icon	Icon representing the master cell with which you want to replace the existing cell at the specified array location. Use the `EmptyIcon` global symbol for this argument to replace an existing cell with an empty cell.

t_orientation Orientation for the new master.
Valid Values: `0`, `90`, `180`, `270`, `sideways`, `sideways&90`, `upsideDown`, `sideways&270`, `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, and `MXR90`.

Value Returned

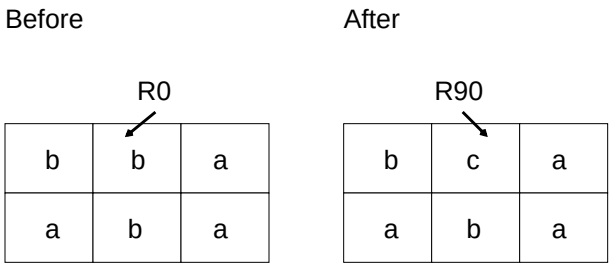
x_column:x_row Location of the changed cell.

`nil` Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
      scChangeCell(1:1 "c" "90"))
```

`scChangeCell` changes cell b at location 1:1 to the new master, represented by icon c, rotates the new cell 90 degrees before placing it, and returns `1:1`, the location of the changed cell.



scChangeIcon

```
scChangeIcon(
     t_fromIcon
     t_toIcon
     )
     => t_icon/nil
```

Description

Changes the icon that represents a specified master cell. `scChangeIcon` changes all the cells in the array with the specified old icon to the new icon.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

t_fromIcon	Old icon value.
t_toIcon	New icon value.

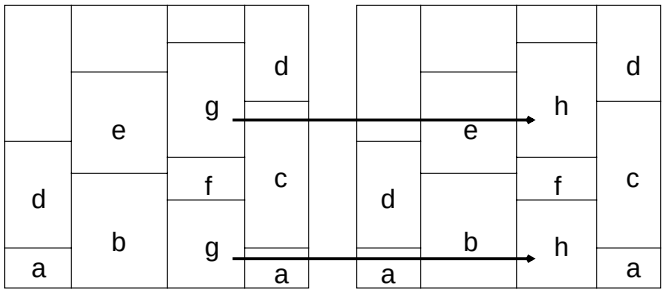
Value Returned

t_icon	New icon value.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     scChangeIcon( "g" "h"))
```

`scChangeIcon` changes all of the g icons in the array to h icons.



scChangeMaster

```
scChangeMaster(
     t_icon
     t_master-name
     )
     => t_master-name/nil
```

Description

Changes an existing icon to a new master. The cells with the specified icon in the array block change to the new master. The icon itself does not change.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane. The new master must be the same size as the old master.

Arguments

t_icon	Existing icon value.
t_master-name	Name of the new master in the form "*cell_name view_name*."

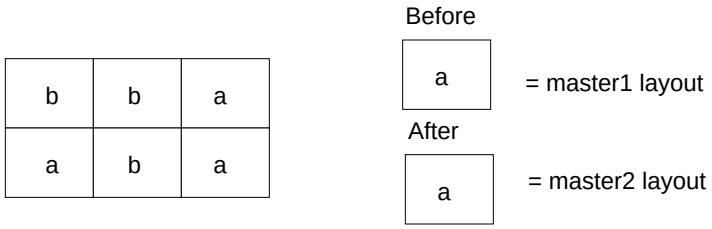
Value Returned

t_master-name	Name of the new master.
nil	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     scChangeMaster( "a" "master2 layout"))
```

If the masters field in the template for the figure below is *a*:`master1 layout`, then `scChangeMaster` changes the master cell for icon a from `master1 layout` to `master2 layout`. The icon does not change.



scChangeOrient

```
scChangeOrient(
     x_column:x_row
     t_orientation
     )
     => x_column:x_row/nil
```

Description

Changes the orientation of an individual cell. The dimensions of the transformed cell must be such that it fits correctly into the grid of the array.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column:x_row	Location of an individual cell.
t_orientation	Orientation for the new master. Valid Values: `0`, `90`, `180`, `270`, `sideways`, `sideways&90`, `upsideDown`, `sideways&270`, `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, and `MXR90`.

Value Returned

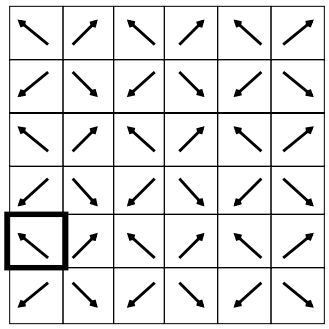
x_column:x_row	Location of the individual cell.
`nil`	Indicates an error condition.

Example

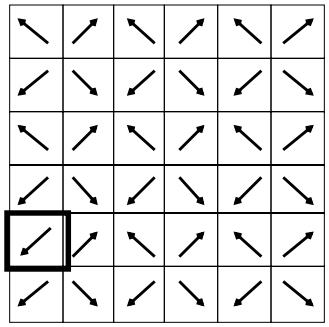
```
scPlane( geGetEditCellView( ) "myPlane"
     scChangeOrient(0:1 "0"))
```

`scChangeOrient` changes the orientation of the cell at location 0:1 to the identity transform.

Before



After



scCompileArray

```
scCompileArray(
     d_layoutCellView
     d_templateCellView
     [t_personalityFile]
     [t_parameterValue]
     [t_procedureFile]
     => t/nil
```

Description

Compiles an array.

Arguments

d_layoutCellView	Database identification number of the cellview in which to place the compiled array.
d_templateCellView	Database identification number of the cellview that will be used as a template.
t_personalityFile	Name of the file containing the personality matrix.
t_parameterValue	List of parameters in the form *((*`param value`*) (param value)...)*
t_procedureFile	Name of the procedure file to use.

Value Returned

`t`	The function executed successfully.
`nil`	Indicates an error condition.

Example

```
scCompileArray( myLayout myTemplate "myPersonalityFile"
              "((rows \"2\") (cols \"5\"))" "myProcedureFile")
```

Places the compiled array in the `myLayout` cellview. `myTemplate` is the template cellview. `myPersonalityFile` contains the personality matrix. The list of parameters indicates two

rows and five columns. Quotation marks inside strings are preceded by a backslash. `myProcedureFile` is the name of the procedure file.

scDeleteCell

```
scDeleteCell(
     x_column:x_row
     )
     => t/nil
```

Description

Deletes a cell.

Restrictions apply to array type as follows:

Row-biased	You can delete cells only to the right or left of existing cells in row-biased arrays. Existing cells in a row move to the left to fill in gaps left by the deleted cell.
Column-biased	You can delete cells only above or below existing cells. Existing cells in a column move down to fill in gaps left by the deleted cell.
Unbiased	You cannot delete individual cells.

Deleting individual cells in row- or column-biased arrays might leave the array with ragged right or top edges. When this happens, `scDeleteCell` automatically adds or adjusts padding cells to keep the array boundary rectangular. Arrays are always left- and bottom-justified.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column:x_row	Location of cell to be deleted.

Value Returned

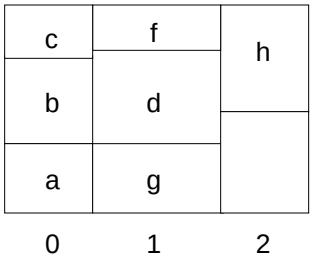
`t`	The function completed successfully.
`nil`	Indicates an error condition.

Example

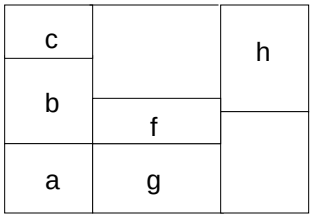
```
scPlane( geGetEditCellView( ) "myPlane"
     scDeleteCell( 1:1 ))
```

`scDeleteCell` deletes cell d, located at `1:1`. Cell f moves down to fill the gap left by the deleted cell. `scDeleteCell` automatically inserts a pad cell to keep the boundary rectangular.

Before



After



scDeleteColumn

```
scDeleteColumn(
     x_column
     )
     => t/nil
```

Description

Deletes a column in a column-biased array. The existing columns in the array move to the left to fill the gap.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column	Number of the column to delete.

Value Returned

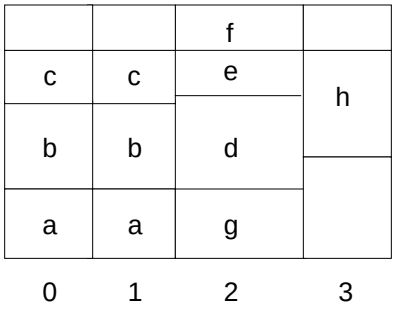
`t`	The function completed successfully.
`nil`	Indicates an error condition.

Example

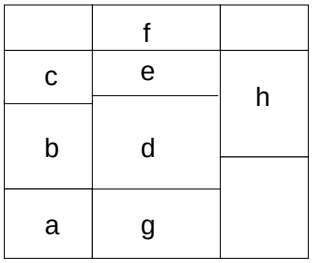
```
scPlane( geGetEditCellView( ) "myPlane"
     scDeleteColumn( 1))
```

`scDeleteColumn` deletes column 1. Columns 2 and 3 move to the left to fill the gap left by the previous column 1.

Before



After



scDeletePlane

```
scDeletePlane(
     d_cellView
     t_planeName
     => t/nil
```

Description

Deletes a plane in an open cellview using the name you give.

Arguments

d_cellView	Object identifier of the cellview that contains the plane to delete.
t_planeName	Name of the plane.

Value Returned

t	The function executed successfully.
nil	Indicates an error condition.

Example

```
scDeletePlane( geGetEditCellView( ) "myplane")
```

Deletes the plane named myplane from the cellview being edited in the current window.

scDeleteRow

```
scDeleteRow(
     x_row
     )
     => t/nil
```

Description

Deletes a row in a row-biased array. The existing rows in the array move down to fill the gap.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_row	Number of the row to delete.

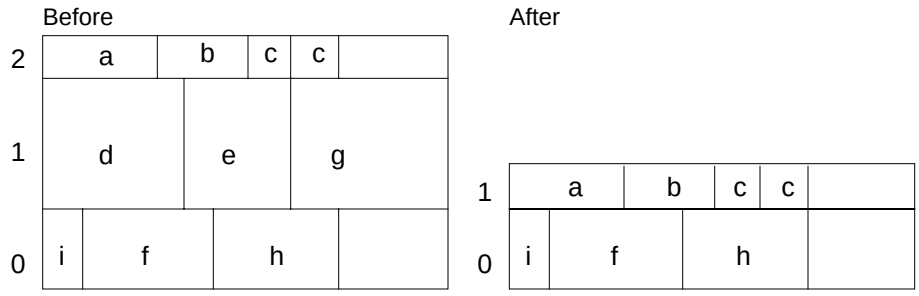
Value Returned

`t`	The function completed successfully.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     scDeleteRow( 1))
```

`scDeleteRow` deletes row 1. Row 2 moves down to fill the gap left by the previous row 1.



scDrawPlane

```
scDrawPlane(
     d_cellView
     t_planeName
     l_bbox
     => d_mosaicId/nil
```

Description

Creates a plane in an open cellview using the name and dimensions you give.

Arguments

d_cellView	Object identifier of the cellview on which to draw the plane.
t_planeName	Name of the plane.
l_bbox	Bounding box coordinates of the new plane.

Value Returned

d_mosaicId	Object identifier for the newly created plane.
nil	Indicates an error condition.

Example

```
scDrawPlane( geGetEditCellView( ) "addressDecode"
list(0:0 20:100))
```

Creates a new plane called addressDecode in the cellview being edited in the current window.

scGetPlanePoint

```
scGetPlanePoint(
     d_cellView
     [t_prompt/f_xcoord:f_ycoord]
     )
     => (t_plane-name x_column:x_row)/nil
```

Description

Facilitates high-level interactive access to array elements using a coordinate pair to identify a plane. You can supply the coordinate as an argument in the function call or you can prompt the end user to specify the coordinate by pointing with the cursor and clicking the mouse button. You can execute this function independently of the `scPlane` function.

Arguments

d_cellView	Cellview containing the array structure you want to manipulate.
t_prompt	Prompts the end user to point interactively to an array element, using the mouse and cursor.
f_xcoord:f_ycoord	An (X:Y) coordinate pair that identifies the plane and the location of the element in the plane that you specify.

Value Returned

t_plane-name	Plane identified interactively by the end user or the coordinate point you specified directly.
x_column:x_row	Column and row location corresponding to the specified point.
`nil`	Indicates an error condition.

Example

```
scGetPlanePoint( geGetEditCellView( ) "Please point to an array element."))
```

`scGetPlanePoint` returns the name of the plane that the user pointed to interactively and the column and row location of the point.

sclcon

```
scIcon(
     x_column:x_row/t_master-name
     )
     => t_icon/nil
```

Description

Returns the icon attribute of an array cell at a given location or of the named master cell.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column:x_row	Location of an array cell.
t_master-name	Two-word string, such as `master1 layout`, identifying the cellname and view name of a master cell for which you want an icon attribute.

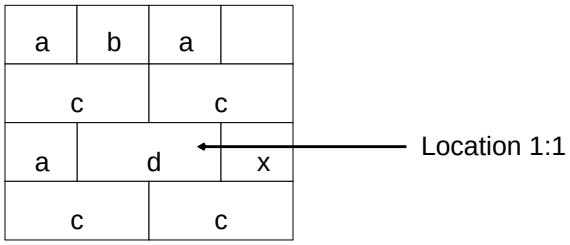
Value Returned

t_icon	Icon representing the specified master as the master at the specified location.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane" icon=scIcon( 1:1))
```

For the array below, `scIcon` finds location 1:1 and returns `d`, the icon attribute of the cell at that location, and assigns that value to the `icon` variable.



scMaster

```
scMaster(
     t_icon/x_column:x_row
     )
     => t_master-name/nil
```

Description

Finds the master of an icon or of a cell at a given location. A global list named *masterList* contains the icon and master pairs used in the array block and array plane.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

t_icon	Icon for a master within a given array block.
x_column:x_row	Location of a master within a given array block.

Value Returned

t_master-name	String containing the master cell and view names, separated by spaces.
`nil`	Indicates an error condition.

Examples

Assume the masters field in the template for the figure below is `a:master1 layout; b:master2 layout`.

b	b	a
a	b	a

Example 1

```
scPlane( geGetEditCellView( ) "myPlane" scMaster( "a"))
```

`scMaster` returns `master1 layout`.

Example 2

```
scPlane( geGetEditCellView( ) "myPlane" scMaster( 1:0))
```

`scMaster` returns `master2 layout`.

scOrient

```
scOrient(
     x_column:x_row
     )
     => t_orientation/nil
```

Description

Returns the orientation attribute of an array cell at a given location. The orientation pattern field defines and assigns the orientation to each cell of the array.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column:x_row	Location of an array cell.

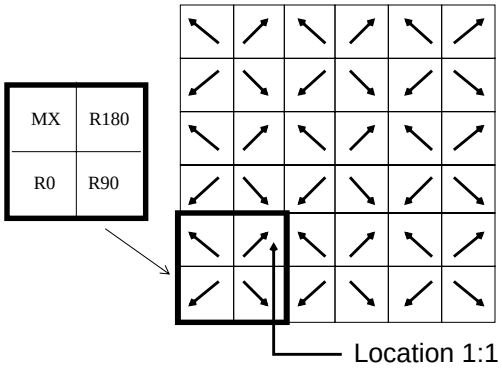
Value Returned

t_orientation	Returns one of the following values: `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, and `MXR90`.
`nil`	Indicates an error condition.

Example

```
scPlane(geGetEditCellView( ) "myPlane"
     orient=scOrient( 1:1))
```

`scOrient` finds location 1:1 and returns the `orientation` attribute of the cell at that location, `180`, and assigns the value to the `orient` variable.



scPlane

```
scPlane(
     d_cellView
     t_plane-name)
     ['g_SKILL-func
     'g_SKILL-func...]
     )
     => d_mosaicId/nil
```

Description

Calls functions directly from SKILL, rather than through the Structure Compiler.

Arguments

d_cellView	Cellview containing the array structure you want to manipulate.
t_plane-name	Name of the plane to which the functions specified as the subsequent arguments are applied. If the plane name you specify does not exist, `scPlane` prints an error message and does not evaluate the functions inside the parentheses.
'g_SKILL-func	Any SKILL function.

Value Returned

d_mosaicId	The `mosaicId` of the plane.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "ramCore",
     scChangeBias("row") scAddRow(1 "above" 30))
```

Adds a new row of cells to an array block named `ramCore`.

scPromotePin

```
scPromotePin(
    x_column:x_row
    t_terminal_name
    d_pinID
    )
    => d_pinID/nil
```

Description

Promotes a pin on cells within array blocks to the outer level. If a terminal with the name you specify already exists, `scPromotePin` adds the new pin to this terminal. If no terminal with this name exists, `scPromotePin` creates the terminal, then creates a new pin in the top-level cellview which it associates with this terminal. `scPromotePin` creates the new pin on top of the pin identified by *`d_pinId`* in the specified array location.

Note: This function can promote a pin that cannot be promoted according to the syntax of the pins field in the array structure template. However, `scPromotePin` neither copies nor creates an `accessDirection` property on the new pin.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

`x_column:x_row`	Location of the cell in the array block whose terminal is to be promoted.
`t_terminal_name`	Terminal name with which to associate the pin on the top-level cellview.
`d_pinID`	Pin identifier referring to a pin on the master cellview that is present in that array location. `scPromotePin` verifies that the correct master is in the specified array location.

Value Returned

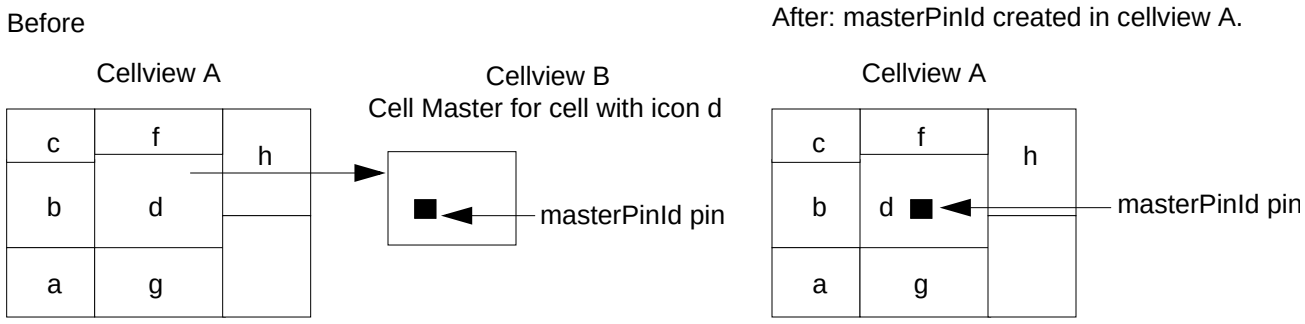
`d_pinID`	Pointer to the newly created pin on the top-level cellview.

`nil` Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     newpin=scPromotePin( 1:1 inputA masterPinId)
```

`scPromotePin` promotes the `masterPinId` pin in the master of the cell located at 1:1 so the pin is visible in cellview A. `inputA` is the name of the terminal on which the pin should be attached in cellview A. `scPromotePin` returns the *`d_pinID`* of the newly created pin on the top-level cellview A and assigns that value to the `newpin` variable.



scResizePlane

```
scResizePlane(
     d_cellView
     t_planeName
     l_bbox
     )
     => d_mosaicId/nil
```

Description

Resizes a plane in an open cellview using the name and dimensions you give.

Arguments

d_cellView	Object identifier of the cellview on which to resize the plane.
t_planeName	Name of the plane.
l_bbox	Bounding box coordinates of the resized plane.

Value Returned

d_mosaicId	Object identifier for the resized plane.
nil	Indicates an error condition.

Example

```
scResizePlane( geGetEditCellView( ) "addressDecode"
list(0:0 20:100))
```

Resizes the addressDecode plane in the cellview being edited in the current window.

scStretchEmptyCell

```
scStretchEmptyCell(
    x_column:x_row
    [n_dimension]
    )
    => f_width:f_height/nil
```

Description

Changes or inquires about the dimensions of an empty cell in an array. A special `scEmptyIcon` symbol refers to empty cells in an array for which no corresponding master exists. Empty cells can be any size. The function alters the width of the cell in a row-biased array and the height of the cell in a column-biased array. You cannot change the size of an individual cell in an unbiased array.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column:x_row	Location of an empty cell in an array.
n_dimension	A number (integer or real) that represents the new width or height to which the specified cell should be stretched (or shrunk). Omit this argument if you only want to inquire about the dimension of an empty cell.

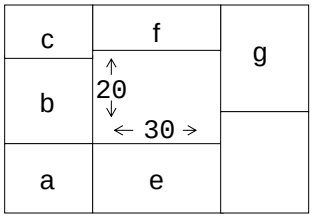
Value Returned

f_width:f_height	A two-element list defining the dimensions of the cell.
`nil`	Indicates an error condition.

Example 1

```
scPlane( geGetEditCellView( ) "myPlane"
     dimensions=scStretchEmptyCell(1:1))
```

For the column-biased array in the figure below, `scStretchEmptyCell` queries the empty cell at location 1:1 and returns `(20.0 30.0)`, the cell dimensions, and assigns the value to the `dimensions` variable.

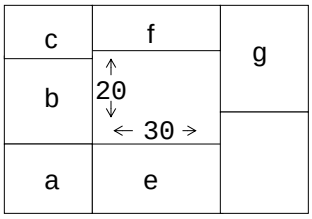


Example 2

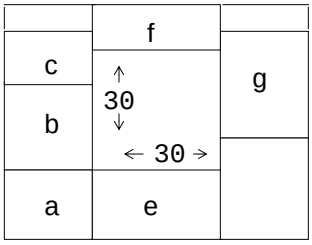
```
scPlane( geGetEditCellView( ) "myPlane"
     dimensions=scStretchEmptyCell(1:1 30))
```

For the column-based array below, `scStretchEmptyCell` stretches the height of the empty cell at location 1:1 to 30 and returns `(30.0 30.0)`, the new dimension of the empty cell, and assigns the value to the `dimensions` variable. The function automatically fills the array with padding cells to keep its boundary rectangular.

Before



After



scSwapCells

```
scSwapCells(
     x_column1:x_row1
     x_column2:x_row2
     [t_orientation1 [t_orientation2]]
     )
     => t/nil
```

Description

Swaps the position of two cells. After the transformation by the specified orientation, the two cells must have the same height and width in an unbiased array, the same height in a row-biased array, and the same width in a column-biased array.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column1:x_row1	Location of first cell to be swapped.
x_column2:x_row2	Location of second cell to be swapped with first cell.
t_orientation1	Orientation of first cell after the swap. Valid Values: `0`, `90`, `180`, `270`, `sideways`, `sideways&90`, `upsideDown`, `sideways&270`, `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, and `MXR90`.
t_orientation2	Orientation of second cell after the swap. If you want to specify an orientation for the second cell, you must also specify the orientation for the first cell. Valid values are the same as above.

Value Returned

`t`	The function executed successfully.
`nil`	Indicates an error condition.

Example

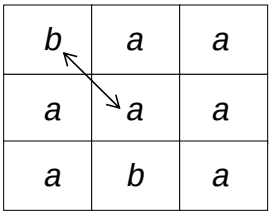
```
scPlane( geGetEditCellView( ) "myPlane"
     scSwapCells(1:1 0:2))
```

`scSwapCells` swaps the cell located at 1:1 with the cell located at 0:2 with no change in the orientation of either cell.

Before

a	*a*	*a*
a	*b*	*a*
a	*b*	*a*

After



scSwapColumns

```
scSwapColumns(
     x_column1
     x_column2
     )
     => t/nil
```

Description

Swaps entire columns. You cannot use `scSwapColumns` with row-biased arrays because they do not have real columns. The columns being interchanged do not need to be the same height.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_column1	First column to be swapped.
x_column2	Second column to be swapped with the first column.

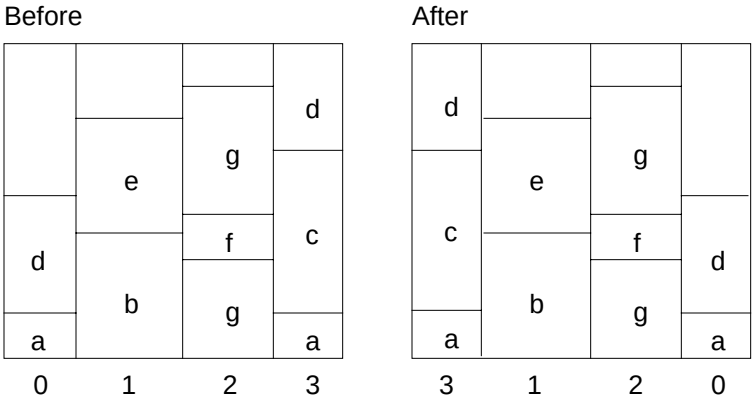
Value Returned

`t`	The function executed successfully.
`nil`	Indicates an error condition.

Example

```
scPlane(geGetEditCellView( ) "myPlane" scSwapColumns(0 3))
```

For the array below, `scSwapColumns` swaps column 0 with column 3.



scSwapRows

```
scSwapRows(
     x_row1
     x_row2
     )
     => t/nil
```

Description

Swaps entire rows. You cannot use `scSwapRows` with column-biased arrays because they do not have real rows. The rows being interchanged do not need to be the same width.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

x_row1	First row to be swapped.
x_row2	Second row to be swapped with the first row.

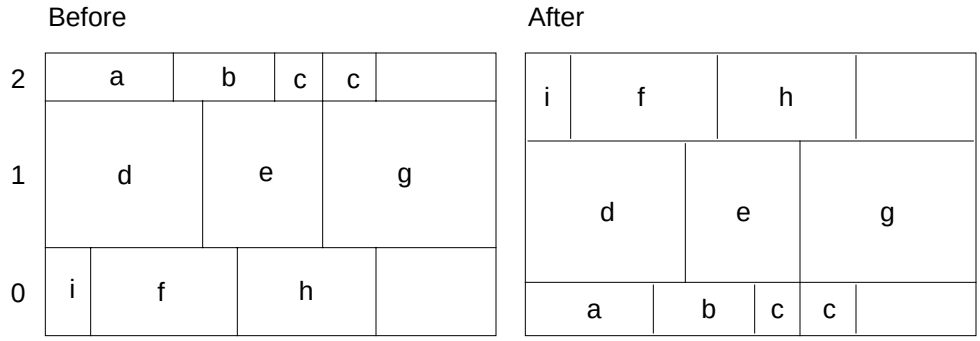
Value Returned

`t`	The function ran successfully.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane" scSwapRows(0 2))
```

For the array below, `scSwapRows` swaps row 0 with row 2.



scUnusedIcon

```
scUnusedIcon(
     )
     => t_icon/nil
```

Description

Returns a unique value that you can use as an icon when adding a new master to the list of masters in an array block. Refer to the `scAddMaster` function.

Prerequisites

Call this function only within the `scPlane` function or in the procedure field of a template plane.

Arguments

None.

Value Returned

t_icon	Value guaranteed not to be currently in use for any other master in the array block.
`nil`	Indicates an error condition.

Example

```
scPlane( geGetEditCellView( ) "myPlane"
     scAddMaster( "master1 layout" scUnusedIcon( ))
```

`scUnusedIcon` returns an icon value not currently used in the array block.

5

Placement and Routing Translation Functions

This chapter provides syntax, descriptions, and examples for the Cadence® SKILL language functions associated with the Export to Router and Import from Router commands and interprocess communication between the Cadence Design Framework II product, the Virtuoso custom placer, and the Virtuoso custom router.

Introduction on page 592

Translator User Interface Functions on page 592

- iccDisplayExportForm on page 593
- iccDisplayImportForm on page 594
- iccExportCellview on page 595
- iccImportCellview on page 598
- iccNewRules on page 600
- iccOpenCurrentCellviewRules on page 601
- iccOpenRules on page 602
- iccStartICC on page 603

Interprocess Communication Functions on page 604

- iccIsConnected on page 605
- iccSendCommand on page 606
- iccSendSkillCommand on page 607

Introduction

This chapter provides syntax, descriptions, and examples for the Cadence® SKILL language functions associated with

- The Export to Router and Import from Router commands (for the placement and routing translator) in the Virtuoso layout tools
- Interprocess communication between the Cadence Design Framework II product, the Virtuoso custom placer, and the Virtuoso custom router

Only the functions documented in this chapter are supported for public use in MPS communication between the Cadence Design Framework II product, the Virtuoso custom placer, and the Virtuoso custom router.

For more information about the placement and routing translator, see the *Virtuoso Custom Placement and Routing Preparation Guide.*

Translator User Interface Functions

These functions are provided so that you can define bindkeys that correspond to the placement and routing translator commands in the Command Interpreter Window and layout window menus. These functions are designed for use with bindkeys and are not recommended for use as a general layout-to-placement-and-routing translator programmer interface.

iccDisplayExportForm

```
iccDisplayExportForm([d_cellview_id])
     => t/nil
```

Description

Displays the Export to Router form.

Arguments

If the Export to Router form is not already displayed, the export layout cellview will be set to *cellview_id*, if given, or the cellview in the current layout window. If the Export to Router form is already displayed, the export layout cellview is not changed and *cellview_id* is ignored.

Value Returned

`t`	In all cases.
`nil`	The export form was already displayed. Otherwise returns t if the export form is displayed.

iccDisplayImportForm

```
iccDisplayImportForm([d_cellview_id])
     => t/nil
```

Description

Displays the Import from Router form.

Arguments

If the Import from Router form is not already displayed, the import layout cellview will be set to *d_cellview_id*, if given, or the cellview in the current layout window. If the Import from Router form is already displayed, the import layout cellview is not changed and *d_cellview_id* is ignored.

Value Returned

`t`	The form was successfully displayed.
`nil`	The form was already displayed.

iccExportCellview

```
iccExportCellview(
     [?layoutLCV      list(t_library t_cell t_view)]
     [?layoutArea     l_boundingBox]
     [?netlistSource
                  list(nil 'layoutCellview 'value list(t_library
                         t_cell t_view))
                | list(nil 'schematicCellview 'value list(t_library
                         t_cell t_view))
                | list(nil 'netlistFile 'value t_path)]
     [?alternateViews list(t_view1 t_view2 ...)]
     [?background t|nil]
     [?exportDirectory t_path]
     [?rulesFile       t_path]
     [?conductorDepth  x_conDepth]
     [?keepoutDepth    x_keepoutDepth]
     [?pinConnection   'must_join | 'strong | 'weak]
     [?optionList      list(['fullConnectivity]
                       ['cutToEdge]
                       ['interLayer]
                       {'incrementalUpdate | 'noIncrementalUpdate})]
     [?startICC        t|nil]
     [?iccOptions      t_options]
     [?doneCallback    s_functionName]
     )
     =>t/nil
```

Description

Exports the specified cellview of type maskLayout to the router.

For details about any argument, see "About the Export to Router Form" in the *Virtuoso Custom Placement and Routing Preparation Guide*.

This function does not check that the values you specify as arguments are valid for the translator.

Arguments

`?layoutLCV`	The library, cell, and view to export.
`?layoutArea`	Coordinates of a bounding box that defines the area to export. The default is `nil`, and you export the entire cellview.

`?netlistSource`	The file or cellview from which to extract the netlist. You can specify the same layout cellview that you are exporting, a schematic cellview, or a netlist file. The default source is the layout cellview you are exporting.
`?alternateViews`	Alternate view names to select for translation. By default, the translator does not select alternate views.
`?background`	Controls whether to export in the foreground or background mode. The default is `t` and export is in background mode.
`?exportDirectory`	The directory where the translator writes the output files. The default is the directory where you started the Design Framework II product.
`?rulesFile`	The rules file to control translation. By default, the translator uses the rules in the technology file for the layout cellview.
`?conductorDepth`	The depth to which conductor shapes are calculated and translated. Valid values are integers from `0` to `32`. The default is `32`.
`?keepoutDepth`	The depth to which keepout shapes are calculated and translated. Valid values are integers from `0` to `32`. The value must be greater than the value you set for `?conductorDepth`. The default is `32`.
`?pinConnection`	The default pin connection type. The default is `strong`.
`?optionList`	A list of command line options to pass to the translator. The options `incrementalUpdate` and `noIncrementalUpdate` are mutually exclusive. If you specify both, `incrementalUpdate` is used. The default is `nil`.
`?startICC`	Set to `t` to start the router after a successful translation. If translation is unsuccessful, or you specify `nil`, the router l does not start. The default is `nil`.
`?iccOptions`	A list of command line options for the router. These options are appended to the router command string. The default is `nil`, and the translator does not pass any extra options to the router.

`?doneCallback`

The symbolic name of a function to be called after translation completes, even if translation fails. The function has the form

`myFunction(`*`exportObject exportStatus`*`)`

`exportObject` is the internal export data structure (contents currently undefined). *`exportStatus`* is the status returned by the translator. A `0` status indicates that the translation was successful. Any other value indicates a failure.

The default is `nil` for no function.

Value Returned

`t`

Export in foreground mode was successful, or export in background mode was successfully started.

`nil`

The operation failed.

iccImportCellview

```
iccImportCellview(
     ?layoutLCV       list(t_libraryName t_cellName t_viewName)
     ?iccFile         t_file | list(t_file 'session)
     [?sectionList    list(['placement] ['routes] ['boundary])]
     [?optionList     list(['segmentedPath] ['usePinPurpose])]
     [?background     t|nil]
     [?doneCallback   s_functionName]
     )
     =>t/nil
```

Description

Imports the specified router file into the specified cellview of type maskLayout, creating the cellview if necessary.

For details about any argument, see "About the Import from Router Form" in the *Virtuoso Custom Placement and Routing Preparation Guide*.

Important

This function does not check that the values you specify as arguments are valid for the translator.

Arguments

`?layoutLCV` The library, cell, and view to create or update.

Note: You cannot import into an open, writable cellview.

`?iccFile` The router output file to import, or a list containing both the file name and type. In the current release, only session files may be imported.

`?sectionList` The sections in the router file to import. The choices of sections depend on the type of file you import. The default is `nil`, and all sections are translated.

`?optionList` A list of one or more options to pass to the translator. The default is `nil`, and no extra options pass to the translator.

`?background` Controls whether to import in the foreground or background mode. The default is `t` and export is in background mode.

`?doneCallback`

The symbolic name of a function to call after the import completes (even if it fails). The function has the form

`myFunction(`*`g_importObject g_importStatus`*`)`

`g_importObject` is an internal import data structure (whose contents are undefined in Release 1.0). *`g_importStatus`* is the status the translator returns. A status value of `0` indicates that the translation was successful. Any other value indicates a failure. The default is `nil`, and no function is called.

Value Returned

`t`

Import in foreground mode was successful, or import in background mode was successfully started.

`nil`

The operation failed.

iccNewRules

```
iccNewRules([tech_library_name])
     => t/nil
```

Description

Displays the New Rules form. This command is equivalent to the *New Rules* command.

Arguments

tech_library_name	The technology library to be selected in the New Rules form.

Value Returned

`t`	If you do not cancel the form.
`nil`	If you cancel the form.

iccOpenCurrentCellviewRules

```
iccOpenCurrentCellviewRules()
     =>t/nil
```

Description

Displays the Open Rules browser for the current cellview. This command is equivalent to the *Open Rules* command.

If the current cellview technology contains placement and routing translation rules, this technology file is selected in the browser by default. Otherwise, the browser prompts the user to select a file.

Value Returned

`t`	If you open a rules file with the browser.
`nil`	If you cancel the browser.

iccOpenRules

```
iccOpenRules([location path tech_library_name])
     => t/nil
```

Description

Displays the Open Place and Route Rules file-library browser so that you can open a rules file from a file or a library. This command is equivalent to the *Open Rules* command.

Arguments

location	The location of the rules file can be either `path` or `library`. If *location* is `path`, the file browser is displayed in the open file-library browser. If *location* is `library`, then the library browser is displayed in the open file-library browser.
path	The default path selected in the browser.
tech_library_name	The default library selected in the browser.

If *location*, *path*, or *tech_library_name* is omitted or is set to `nil`, the previous *location*, *path*, or *tech_library_name* is selected.

Value Returned

`t`	If you open a rules file with the browser.
`nil`	If you cancel the browser.

iccStartICC

```
iccStartICC([options])
     => t/nil
```

Description

Starts the router as separate process. This function is equivalent to the *Start Router* command.

Arguments

options	A string of router command line options. If *options* is given, the router will be started with those options.

Value Returned

t	The router started successfully or was already running.
nil	The router did not start.

Interprocess Communication Functions

The functions in this section let you check the status of the connection between a Design Framework II cellview and the router and pass commands to the router.

You can connect Design Framework II cellviews to router windows. Multiple cellviews can be connected simultaneously. Once the cellview and router are connected, you can send commands from the Design Framework II product to the router.

For general information about the Design Framework II product and interprocess communication, see the *Interprocess Communication SKILL Functions Reference*.

iccIsConnected

```
iccIsConnected(d_cellview_ID)
     => t/nil
```

Description

Determines whether a cellview has an associated router process.

To launch the router and establish a connection, use the commands *Route – Connect to Router* in the Virtuoso® window for the cellview.

Arguments

d_cellview_ID	The ID of the cellview.

Value Returned

t	The cellview is connected to an router window.
nil	No connection is open.

iccSendCommand

```
iccSendCommand(d_cellview_ID t_command)
     => t_retString
```

Description

Sends a command to the router command line interpreter.

The results are as if you had typed the command in the router command entry field.

Arguments

d_cellview_ID	The ID of the cellview.
t_command	A string to pass to the router. Strings that are constants should be quoted. Strings that are variables need not be quoted.

Example

This example displays the router design report in the report window.

```
iccSendCommand(getEditCellview() "report design")
```

Value Returned

t_retString	Whatever string the router returns after successfully executing a task.

iccSendSkillCommand

```
iccSendSkillCommand(d_cellview_ID t_command)
     => t_retString
```

Description

Sends a SKILL command to the router.

You can use only the core SKILL commands documented in the *SKILL Language Reference.* Other, application-specific commands are not supported in the router.

Arguments

d_cellview_ID	The ID of the cellview.
t_command	A SKILL command to pass to the router.

Value Returned

t_retString	Whatever string the router returns after successfully executing a task.

6

Virtuoso XL Functions

This section provides syntax and descriptions for the Cadence® SKILL functions associated with the Virtuoso® XL Layout Editor (Virtuoso XL).

Introduction on page 612

lxCmdOptions on page 613

lxCmdShiftOptions on page 614

lxEditPartitioning on page 615

lxEditPinPlacement on page 616

lxEditPlacementStyle on page 617

lxGetLXInfo on page 618

lxHiAlign on page 619

lxHiChain on page 620

lxHiClone on page 621

lxHiConnectInstPin on page 622

lxHiCreateInstFromSch on page 623

lxHiCreateMPP on page 624

lxHiEditComponentTypes on page 625

lxHiHideIncNets on page 626

lxHiLockSelected on page 627

lxHiMoveAutomatically on page 628

lxHiPlcDisableCongestionDisplay on page 629

lxHiPlcEnableCongestionDisplay on page 630

lxHiProbe on page 631

lxHiReInitDesign on page 632

lxHiSetCorrespondence on page 633

lxHiSetOptions on page 634

lxHiShowIncNets on page 635

lxHiStack on page 636

lxHiSwapComps on page 637

lxHiUnlockSelected on page 638

lxHiUpdateCellViewPair on page 639

lxHiUpdateComponentsAndNets on page 640

lxHiUpdateDeviceCorr on page 641

lxHiUpdateLayoutParameters on page 642

lxHiUpdateSchematicParameters on page 643

lxHiUpdateSwitchViews on page 644

lxHiVerifyDesign on page 645

lxHiVerifyStatus on page 646

lxPermPermutePins on page 647

lxPlcAppendPlaceSetupFieldValue on page 648

lxPlcGetPlaceSetupFieldValue on page 650

lxPlcIsPlaceSetupFieldEnabled on page 651

lxPlcReplacePlaceSetupField on page 652

lxPlcSetPlaceSetupFieldEnableState on page 654

lxPlcSetPlaceSetupFieldValue on page 656

lxProbeRemoveAll on page 658

lxToggleIncNets on page 659

lxTpSaveTemplate on page 660

leWeAddVia on page 662

leWeCheckRoutes on page 663

leWeDisableAsManyAsFitMenuItem on page 664

leWeDisableCopyRouteMirrorMenuItem on page 665

leWeDisableViaPatternMenuItem on page 666

leWeFinishRoute on page 667

leWeHiCheckRoutes on page 668

leWeHiCopyRoute on page 669

leWeHiCriticWire on page 670

leWeHiPull on page 671

leWeHiRouteOptions on page 672

leWePathWidthMenuCB on page 673

leWePickUpDroppedWires on page 674

leWeRefresh on page 675

leWeReports on page 676

leWeSetGatherBusWiresMenuItemText on page 677

leWeSetInteractiveCheckingText on page 678

leWeToggle on page 679

Introduction

This section provides syntax, descriptions, and examples for the SKILL functions associated with the Virtuoso XL.

These functions are provided so that you can define bindkeys that correspond to the Virtuoso layout editor menu commands. These functions are designed for use with bindkeys and are not recommended for use as a general Virtuoso XL programmer interface.

Only the Virtuoso XL functions documented in this chapter are supported for public use. All other Virtuoso XL functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

lxCmdOptions

```
lxCmdOptions()
```

Description

Binds the right, middle, and left mouse buttons to commands. Entering this command in the command interpreter window (CIW) has no effect.

Arguments

None.

Value Returned

None.

lxCmdShiftOptions

```
lxCmdShiftOptions()
     => nil
```

Description

Activates the bindkeys associated with `Shift`-click and `Shift`-right click during the operation of enterFunction commands. Entering this command in the command interpreter window (CIW) has no effect.

Arguments

None.

Value Returned

`nil`	The command executed successfully.

lxEditPartitioning

```
lxEditPartitioning()
     => t/nil
```

Description

Opens the Partitioning form to let you define an area for confined components in the layout cellview.

Arguments

None.

Value Returned

`t`	The form displayed and was closed successfully.
`nil`	There was a problem and a warning was given.

lxEditPinPlacement

```
lxEditPinPlacement(
     [g_fromLayoutGenFormP]
     )
     => t/nil
```

Description

Opens the Pin Placement form to let you view pin information and constrain pins to boundary edges, order pins, fix pins, set pitch or to start the Pin Placer.

Arguments

g_fromLayoutGenFormP
: Tells the function if the call came from the Layout Generation form. This argument is obsolete; it is present for backward compatibility.
Valid Values: `t`, `nil`
Default: `nil`

Value Returned

`t`
: The form displayed and was closed successfully.

`nil`
: There was a problem and a warning was given.

lxEditPlacementStyle

```
lxEditPlacementStyle()
     => t
```

Description

Opens the Placement Planning form for standard cell, mixed styles, or manual user defined, in assisted or manual mode. These modes generate rows based on the parameters you enter in the form.

Arguments

None.

Value Returned

`t`	The form opened successfully.

lxGetLXInfo

```
lxGetLXInfo(
     { t_srcView | t_dstView | t_workingWinId }
     )
     => t/nil
```

Description

Returns the value of the specified LX internal variable. Only one argument may be used at a time.

Arguments

t_srcView	Source cellview ID. Valid Values: `cellview ID`, `netlist view` Default: `cellview ID`
t_dstView	Layout cellview ID. Valid Values: `cellview ID`
t_workingWinId	Working cellview ID. Valid Values: `cellview ID`

Value Returned

`t`	The cellview ID was returned.
`nil`	A user error caused a failure.

lxHiAlign

```
lxHiAlign()
     => t
```

Description

Starts the *Edit – Align* command and opens the Alignment form.

Arguments

None.

Value Returned

`t`	The Alignment form opened successfully.

lxHiChain

```
lxHiChain(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Transistor Chaining form to let you select transistors for chaining or list the transistors available for chaining. Opens the form for the specified cellview; otherwise, uses the current window.

Arguments

w_windowId	Window ID of the window specified. Valid Values: any cellview ID Default: the current window

Value Returned

t	The chaining of transistors was successful.
nil	The chaining of transistors was not completed.

Interactive Function

`lxHiChain( [w_windowId] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the child object and the parent object. If you do not specify *w_windowId*, the layout editor uses the current window.

lxHiClone

```
lxHiClone(
     [w_windowId]
     => t/nil
```

Description

Replicates a section of the layout that is associated with a section of the schematic so that the layout replica can be placed at more than one location in the layout with each part preserving the hierarchical structure of the design. You can clone devices, pins, and (if selected from the layout window) interconnect structures such as wires and paths made of shapes.

Arguments

w_windowId	Window ID of the window specified. Valid Values: any cellview ID Default: the current window

Value Returned

t	The cloning of devices was successful.
nil	The cloning of devices was not completed.

Interactive Function

`lxHiClone( [w_windowId] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the child object and the parent object. If you do not specify *w_windowId*, the layout editor uses the current window.

lxHiConnectInstPin

```
lxHiConnectInstPin()
     => nil
```

Description

Starts the *Connect Instance Pin* command to let you add an instance pin to a net.

Arguments

None.

Value Returned

`nil`	The command was cancelled.

lxHiCreateInstFromSch

```
lxHiCreateInstFromSch(
     [w_windowId]
     )
     => t/nil
```

Description

Starts the *Pick From Schematic* command to let you place some or all schematic elements in the layout cellview. Opens the form for the specified cellview; otherwise, uses the current window.

Arguments

w_windowId	Window ID of the window specified. Valid Values: any cellview ID Default: the current window

Value Returned

t	The command started successfully.
nil	The command was cancelled.

Interactive Function

`lxHiCreateInstFromSch( [w_windowId] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the child object and the parent object. If you do not specify *w_windowId*, the layout editor uses the current window.

lxHiCreateMPP

```
lxHiCreateMPP(
     [w_windowId]
     )
     => nil
```

Description

Starts the *Create Multipart Path* command to let you create multipart paths.
Starts the command for the specified cellview; otherwise, uses the current window.

Arguments

w_windowId	Window ID of the window specified. Valid Values: any cellview ID Default: the current window

Value Returned

nil	The command was cancelled.

Interactive Function

`lxHiCreateMPP( [w_windowId] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the child object and the parent object. If you do not specify *w_windowId*, the layout editor uses the current window.

lxHiEditComponentTypes

```
lxHiEditComponentTypes(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Edit Component Types form to let you set the `lxComponentType` parameter. The `lxComponentType` parameter controls how components are assigned to rows and controls the chaining and folding parameters. Opens the form for the specified cellview; otherwise, uses the current window.

Arguments

w_windowId	Window ID of the window specified. Valid Values: any cellview ID Default: the current window

Value Returned

`t`	The form displayed and was closed successfully.
`nil`	The form was cancelled.

Interactive Function

`lxHiEditComponentTypes( [w_windowId] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the child object and the parent object. If you do not specify *w_windowId*, the layout editor uses the current window.

lxHiHideIncNets

```
lxHiHideIncNets()
     => d_cellViewId
```

Description

Starts the *Hide Incomplete Nets* command and removes the Show Incomplete Nets form if it is present. Also prints a cellview ID in the CIW.

Arguments

None.

Value Returned

d_cellViewId	Database ID of the cellview in which the pin is created.

lxHiLockSelected

```
lxHiLockSelected()
     => t/nil
```

Description

Locks selected objects temporarily at the current location with the current orientation by setting temporary fixed constraints. If the selected objects are constrained by other fixed constraints, a message window asks you to approve overwriting those constraints. You can view fixed constraints in the constraint editor and the Constraint Status Browser window. When you save the design, a dialog box asks you if you want to convert the locked devices into permanent fixed constraints.

Arguments

None.

Value Returned

`t`	Fixed constraints were created successfully.
`nil`	Fixed constraints were not created.

lxHiMoveAutomatically

```
lxHiMoveAutomatically()
     => t
```

Description

Starts the *Place From Schematic* command. If a connectivity reference is not found, the Define Connectivity Reference form is opened.

Arguments

None.

Value Returned

`t`	The *Place From Schematic* command started successfully.

lxHiPlcDisableCongestionDisplay

```
lxHiPlcDisableCongestionDisplay()
     => nil
```

Description

Hides the congestion map generated by the Virtuoso custom placer.

Arguments

None.

Value Returned

`nil`	The congestion map was hidden successfully or was cancelled.

lxHiPlcEnableCongestionDisplay

```
lxHiPlcEnableCongestionDisplay()
     => t/nil
```

Description

Displays the congestion map generated by the Virtuoso custom placer.

Arguments

None.

Value Returned

`t`	The congestion map displayed successfully.
`nil`	The congestion map failed to display.

lxHiProbe

```
lxHiProbe()
     => t
```

Description

Starts the *Probe* command and opens the *Probe Options* form. This command lets you select a design element (component, net, or pin) in the layout (or schematic) cellview to highlight the corresponding element in the schematic (or layout) cellview.

Arguments

None.

Value Returned

`t`	The *Probe Options* form was hidden successfully or was cancelled.

lxHiReInitDesign

```
lxHiReInitDesign()
     => t
```

Description

Starts the *Gen from Source* command. This command places design elements from the schematic cellview in an empty layout cellview or clears the layout cellview so you can restart.

Arguments

None.

Value Returned

`t`	The *Gen from Source* command started successfully.

lxHiSetCorrespondence

```
lxHiSetCorrespondence()
     => t/nil
```

Description

Opens the XL Create Device Correspondence form to allow you to match schematic devices to layout devices and match layout devices to schematic devices. You can choose whether you want to match devices in manual mode or computer-aided mode. In computer-aided mode, the system creates potential matches, and you choose the matches you want. In manual mode, the system creates a list of unmatched schematic devices and a list of unmatched layout devises; you choose a schematic device and match it to a layout device and visa versa. For both modes, the system changes the layout device name to match the schematic device name.

Arguments

None.

Value Returned

`t`	The form displayed and was closed successfully.
`nil`	The form was cancelled.

lxHiSetOptions

```
lxHiSetOptions()
     => t/nil
```

Description

Opens the Layout XL Options form to allow you to set environment variables.

Arguments

None.

Value Returned

`t`	Environment variables were set successfully.
`nil`	The Layout XL Options form was closed without changes to the environment variables.

lxHiShowIncNets

```
lxHiShowIncNets()
     => t
```

Description

Starts the *Show Incomplete Nets* command and opens the Show Incomplete Nets form. It also prints in the command interpreter window (CIW) the number of incomplete nets you had when the command was initiated.

Arguments

None.

Value Returned

`t`	The form was closed or was cancelled successfully.

lxHiStack

```
lxHiStack(
     [w_windowId]
     )
     => t/nil
```

Description

Opens the Set Transistor Folding form to let you divide a transistor or a transistor stack (an abutted group of MOS transistors) into two or more fingers. If you did not preselect any transistors, no transistor-specific information is shown in the form until you select at least one transistor. If you preselected transistors to fold, the name of first transistor selected is shown in the *Transistor Name* field.

Arguments

w_windowId	Window ID of the window specified. Valid Values: any cellview ID Default: the current window

Value Returned

`t`	Transistor folding and or chaining was completed successfully.
`nil`	The form was cancelled.

Interactive Function

`lxHiStack( [w_windowId] ) => t/nil`

Enter this function with only the window ID argument; the system prompts you to point to the child object and the parent object. If you do not specify *w_windowId*, the layout editor uses the current window.

lxHiSwapComps

```
lxHiSwapComps()
     => t
```

Description

Starts the *Swap Components* command to let you swap the location of two components in the layout cellview. This command switches the locations of the components you select and retains the orientation of the component associated with the position. Connections to the components do not move. You will not be allowed to swap components that are part of a satisfied constraint (unless the swap maintains the constraint).

Arguments

None.

Value Returned

`t`	Two components were swapped successfully.

lxHiUnlockSelected

```
lxHiUnlockSelected(
     g_all
     )
     => t/nil
```

Description

Removes the lock (temporary fixed constraints) set through `lxHiLockSelected()`, then the components can be moved with the *Move* or *Stretch* commands. You can also remove these temporary locks using the constraint editor.

Arguments

g_all	This is an obsolete argument but it is kept for backward compatibility. Valid Values: `t`, `nil` Default: `nil`

Value Returned

`t`	A lock set was removed from the selected components successfully.
`nil`	No components were selected to unlock.

lxHiUpdateCellViewPair

```
lxHiUpdateCellViewPair(
     )
     => ( {t_source t_lib t_cell t_view [t_topcell] | nil} )
```

Description

Starts the *Update – Source* command and opens the Define Connectivity Reference form. You can choose one of three types of connectivity references: `Schematic`, `Netlist`, or `None`.

Arguments

None.

Value Returned

t_source	Connectivity was referenced to this source name, enclosed in quotation marks.
t_lib	The library name referenced to *source*, enclosed in quotation marks.
t_cell	The cell name referenced in *lib*, enclosed in quotation marks.
t_view	The view name referenced by *cell*, enclosed in quotation marks.
t_topcell	The netlist top cell name referenced to *cell*.
`nil`	The form did not close successfully.

Example

```
lxHiUpdateCellViewPair()
=> ("NETLIST" "VXL_99" "and2_spice" "netlist" "MIV")
```

Referenced `NETLIST` to library `VXL_99`, cell `and2_spice`, view `netlist`, top cell `MIV`.

lxHiUpdateComponentsAndNets

```
lxHiUpdateComponentsAndNets()
     => t
```

Description

Starts the *Update – Components and Nets* command to let you map devices in the schematic cellview to one or more devices (including shapes) in the layout cellview.

Arguments

None.

Value Returned

`t`	The command started successfully.

lxHiUpdateDeviceCorr

```
lxHiUpdateDeviceCorr()
     => t
```

Description

Starts the *Update – Device Correspondence* command to let you update connectivity of devices in the layout cellview from a schematic cellview.

Arguments

None.

Value Returned

`t`	The command started successfully.

lxHiUpdateLayoutParameters

```
lxHiUpdateLayoutParameters()
     => t
```

Description

Starts the *Update – Layout Parameters* command to let you implement parameter changes in the schematic cellview that affect physical implementations in the layout cellview.

Arguments

None.

Value Returned

`t`	The command started successfully.

lxHiUpdateSchematicParameters

```
lxHiUpdateSchematicParameters()
     => t
```

Description

Starts the *Update – Schematic Parameters* command to let you update the parameters of devices in the schematic cellview to match those in the layout cellview.

Arguments

None.

Value Returned

`t`	The command started successfully.

lxHiUpdateSwitchViews

```
lxHiUpdateSwitchViews()
     => t
```

Description

Starts the *Connectivity – Change Instance View* command to let you change the layout cellview of a device or selected set of devices in a hierarchical design.

Arguments

None.

Value Returned

`t`	The command started successfully.

lxHiVerifyDesign

```
lxHiVerifyDesign()
     => w_windowId
```

Description

Starts the *Check – Against Source* command. This command checks the schematic cellview for changes made after the last time the layout cellview was changed. It also opens the Info form and writes the differences into the form.

Arguments

None.

Value Returned

w_windowId	The window ID in which the command ran successfully.

lxHiVerifyStatus

```
lxHiVerifyStatus()
     => w_windowId
```

Description

Starts the *Check – Shorts and Opens* command. This command checks the schematic cellview for open and shorted connections. It also opens the Info form and writes the number of incomplete nets, shorts between nets, and invalid overlaps into the form.

Arguments

None.

Value Returned

w_windowId	The window ID in which the command ran successfully.

lxPermPermutePins

```
lxPermPermutePins()
     => t
```

Description

Starts the *Permute Pins* command to let you exchange the connectivity or net connections of the pins of a component. Pins to be permuted must belong to different nets and must first be defined as permutable terminals using the *permuteRule* property for the device.

Arguments

None.

Value Returned

`t`	The command started successfully.

lxPlcAppendPlaceSetupFieldValue

```
lxPlcAppendPlaceSetupFieldValue(
     t_section
     t_fieldName
     g_value
     [t_subsection]
     )
     => t/nil
```

Description

Appends the specified value to the end of the value of the specified field in the placement planning form.

Arguments

t_section	Name of the placement setup section. Valid Values: depends on style Default: none
t_fieldName	Name of the field or subsection in *t_section*. Valid Values: depends on style Default: none
g_value	Value to append to *t_fieldName*. Valid Values: depends on style Default: nil
t_subsection	Name of the placement setup subsection. Valid Values: depends on style Default: none

Value Returned

t	The field value was appended successfully.
nil	The field value append failed, no change.

Example

```
lxPlcAppendPlaceSetupFieldValue("setup" "types" newType)
```

Appends newType to types in setup.

```
lxPlcAppendPlaceSetupFieldValue("NMOS" "setup" newType "types")
```

Appends newType to types in setup of NMOS.

lxPlcGetPlaceSetupFieldValue

```
lxPlcGetPlaceSetupFieldValue(
     t_section
     t_fieldName
     [t_subsection]
     )
     => g_retValue/nil
```

Description

Returns the value of the specified field in a setup section in the placement planning form.

Arguments

t_section	The unique name for the new handle. Enclose character strings in quotation marks (""). Valid Values: depends on style Default: none
t_fieldName	Name of the field or subsection in *t_section*. Valid Values: depends on style Default: none
t_subsection	Name of the placement setup subsection. Valid Values: depends on style Default: none

Value Returned

g_retValue	The field value.
nil	One of the field values was wrong.

Example

```
lxPlcGetPlaceSetupFieldValue("setup" "types")
```

Asks for the value of types in setup.

```
lxPlcGetPlaceSetupFieldValue("NMOS" "types" "setup")
```

Asks for the value of types in setup of NMOS.

lxPlcIsPlaceSetupFieldEnabled

```
lxPlcIsPlaceSetupFieldEnabled(
     t_section
     t_fieldName
     [t_subsection]
     )
     => t/nil
```

Description

Checks whether the specified field in a placement setup section in the placement planning form is enabled.

Arguments

t_section	Name of the placement setup section. Valid Values: depends on style Default: none
t_fieldName	Name of the field or subsection in *t_section*. Valid Values: depends on style Default: none
t_subsection	Name of the placement setup subsection. Valid Values: depends on style Default: none

Value Returned

t	The specified field was checked successfully.
nil	One of the field values was wrong.

Example

```
lxPlcIsPlaceSetupFieldEnabled("setup" "types")
```

Asks for the enable status of `types` in `setup`.

```
lxPlcIsPlaceSetupFieldEnabled("NMOS" "types" "setup")
```

Asks for the enable status of `types` in `setup` of `NMOS`.

lxPlcReplacePlaceSetupField

```
lxPlcReplacePlaceSetupField(
     t_section
     t_fieldName
     g_newField
     [t_subsection]
     )
     => t/nil
```

Description

Deletes or replaces the specified field in a placement setup section in the placement planning form.

Arguments

t_section	Name of the placement setup section. Valid Values: depends on style Default: none
t_fieldName	Name of the field or subsection in *t_section*. Valid Values: depends on style Default: none
g_newField	New field. Valid Values: depends on style Default: none
t_subsection	Name of the placement setup subsection. Valid Values: depends on style Default: `nil`

Value Returned

`t`	The specified field is deleted or replaced successfully.
`nil`	The specified field is not deleted or replaced, no change.

Example

```
lxPlcReplacePlaceSetupField("setup" "types" nil)
```

Deletes `types` in `setup`.

```
lxPlcReplacePlaceSetupField("NMOS" "setup" newField "types")
```

Replaces `types` in `setup` of `NMOS with newField`.

lxPlcSetPlaceSetupFieldEnableState

```
lxPlcSetPlaceSetupFieldEnableState(
     t_section
     t_fieldName
     t_subsection
     [b_newState]
     )
     => t/nil
```

Description

Changes the enable state of *t_fieldName* in a placement setup section in the placement planning form.

Arguments

t_section	Name of the placement setup section. Valid Values: depends on style Default: none
t_fieldName	Name of the field or subsection in *t_section*. Valid Values: depends on style Default: none
t_subsection	Name of the placement setup subsection. Valid Values: depends on style Default: none
b_newState	New enable state. Valid Values: `t`, `nil` Default: none

Value Returned

`t`	The change of enable state to the specified field was successful.
`nil`	The change of enable state to the specified field failed, no change.

Example

```
lxPlcSetPlaceSetupFieldEnableState("setup" "types" nil)
```

Disables `types` in `setup`.

```
lxPlcSetPlaceSetupFieldEnableState("NMOS" "setup" t "types")
```

Enables `types` in `setup` of `NMOS with t`.

lxPlcSetPlaceSetupFieldValue

```
lxPlcSetPlaceSetupFieldValue(
     t_section
     t_fieldName
     g_value
     [t_subsection]
     )
     => g_retValue/nil
```

Description

Appends the value of *t_fieldName* in a placement setup section in the placement planning form.

Arguments

t_section	Name of the placement setup section. Valid Values: depends on style Default: none
t_fieldName	Name of the field or subsection in *t_section*. Valid Values: depends on style Default: none
g_value	New field value. Valid Values: depends on style Default: `nil`
t_subsection	Name of the placement setup subsection. Valid Values: depends on style Default: none

Value Returned

g_retValue	The new field value, appended successfully.
`nil`	The new field value append failed, no change.

Example

```
lxPlcSetPlaceSetupFieldValue("setup" "types" nil)
```

Sets `types` in `setup`.

```
lxPlcSetPlaceSetupFieldValue("NMOS" "setup" newField "types")
```

Replaces `types` in `setup` of `NMOS with newField`.

lxProbeRemoveAll

```
lxProbeRemoveAll()
     => t
```

Description

Removes all probes from the layout cellview.

Arguments

None.

Value Returned

`t`	All probes were removed successfully.

lxToggleIncNets

```
lxToggleIncNets()
     => t
```

Description

If the *Show Incomplete Nets* command is on, this command turns it off. If the *Show Incomplete Nets* command is off, this command turns it on and opens the Show Incomplete Nets form.

Arguments

None.

Value Returned

`t`	The command was toggled successfully.

lxTpSaveTemplate

```
lxTpSaveTemplate()
     => nil
```

Description

Opens the Template Save form.

Arguments

None.

Value Returned

`nil`	The Template Save form was closed successfully.

Wire Editing Functions

This section provides syntax and descriptions for the Cadence® SKILL functions associated with the wire editing capabilities of Virtuoso XL.

leWeAddVia

```
leWeAddVia(
     )
     => t/nil
```

Description

Adds a via and changes the current routing layer while in the *Create — Path* command. If the current routing layer has more than one layer available for selection, the *Add Via* form is opened, which displays available vias and routing layers that can be reached from the current routing layer.

Arguments

None

Return Values

`t`	Specifies which via has been placed.
`nil`	Via is not available. Vias need to be specified in the technology file or rules file.

Example

```
leWeAddVia()
```

leWeCheckRoutes

```
leWeCheckRoutes(
     w_windowId
     t_commandName
     )
     => t/nil
```

Description

Checks the current cellview for violations based on the options selected in the *Verify — Check Routes* form.

Arguments

w_windowId	Window ID in which the current cellview is displayed.
t_commandName	Name of command.

Return Values

`t`	Displays in the CIW, the options selected in the *Verify — Check* form and reports any violations.
`nil`	Window ID is invalid.

Example

```
leWeCheckRoutes(wId "leWeCheckRoutes")
```

leWeDisableAsManyAsFitMenuItem

```
leWeDisableAsManyAsFitMenuItem(
     b_value
     )
     => t
```

Description

Enables or disables *Route Only If All Succeed* and *Route As many As Possible* while routing multiple paths in the *Create Path* command. Also sets the text accordingly in the MMB *Layout* pop up menu.

Arguments

b_value	When set to `t`, *Finish Routing* must complete all paths successfully or none are routed and the text in the *Layout* pop up menu is set to *Route Only If All Succeed.* When set to `nil`, *Finish Routing* completes as many connections as it can and the text in the *Layout* pop up menu is set to *Route As many As Possible.*

Return Values

`t`	Return value is always `t`.

Example

```
leWeDisableAsManyAsFitMenuItem(t)
```

leWeDisableCopyRouteMirrorMenuItem

```
leWeDisableCopyRouteMirrorMenuItem(
     t_value
     b_setenv
     )
     => t/nil
```

Description

Mirrors copied routes on the X axis, Y axis, or both when in the *Copy Route* command. Also sets the text of *Mirror* in the MMB *Layout* pop up menu to the selected orientation.

The orientation values are:

- X (`MX` mode) mirrors the copied routes on the x-axis.
- Y (`MY` mode) mirrors the copied routes on the y-axis.
- XY (`M180` mode) mirrors the copied routes on both axes (45 degrees).
- `None` (`R0` mode) returns orientation to non-mirror copying.

Arguments

t_value	Sets the orientation value. Valid values: `MX`, `MY`, `M180`, `R0`
b_setenv	Sets the value of *t_value* to true or false. Valid values: `t` or `nil`.

Return Values

`t`	Sets the selected orientation value.
`nil`	Invalid orientation value.

Example

```
leWeDisableCopyRouteMirrorMenuItem("R0" t)
```

leWeDisableViaPatternMenuItem

```
leWeDisableViaPatternMenuItem(
     t_type
     )
     => t/nil
```

Description

Sets the via pattern when multiple vias are added to bus wires in the *Create Path* command. Also sets the text of *Via Pattern* in the MMB *Layout* pop up menu to the selected pattern.

The via patterns are:

- Perpendicular
- Diagonal 1
- Diagonal 2
- Stagger
- Out Taper
- In Taper

Arguments

t_type	Sets the via pattern. Valid values: `perp`, `diag_1`, `diag_2`, `stagger`, `out_taper`, `in_taper`

Return Values

`t`	Sets the selected via pattern.
`nil`	Invalid *t_type* value.

Example

```
leWeDisableViaPatternMenuItem("perp")
```

leWeFinishRoute

```
leWeFinishRoute(
     )
     => t/nil
```

Description

Tries to finish the routing connection(s) from the last digitized point(s), while in the *Create — Path* command.

Arguments

None

Return Values

`t`	All paths completed successfully.
`nil`	Unable to complete routing for all paths.

Example

```
leWeFinishRoute()
```

leWeHiCheckRoutes

```
leWeHiCheckRoutes(
     )
     => t
```

Description

Opens the *Verify — Check Routes* form, which lets you set checking options for routing conflicts and routing rule violations.

Arguments

None

Return Values

`t`	*OK* is clicked in the form.
`nil`	*Cancel* is clicked in the form.

Example

```
leWeHiCheckRoutes()
```

leWeHiCopyRoute

```
leWeHiCopyRoute(
     [w_windowId]
     )
     => t/nil
```

Description

Copies existing routes, including vias, to unrouted connections that have similar lengths and topology.

Note: Displays warnings if *Copy Route* will create a violation or if the target area is not a qualified pin or via.

Arguments

w_windowId	Optional window ID in which the current cellview is displayed.

Return Values

`t`	Copies selected routes.
`nil`	Invalid window ID

Example

```
leWeHiCopyRoute()
```

leWeHiCriticWire

```
leWeHiCriticWire(
     [w_windowId]
     )
     => t/nil
```

Description

Eliminates notches and removes extra bends in selected paths and attempts to make local adjustments to the existing paths without rip-up and rerouting. When using area-selection to select paths, the entire path must be selected.

Arguments

w_windowId	Optional window ID in which the current cellview is displayed

Return Values

`t`	Removes extra bends in selected path.
`nil`	Invalid window ID.

Example

```
leWeHiCriticWire()
```

leWeHiPull

```
leWeHiPull(
     [w_windowId]
     )
     => t/nil
```

Description

Compacts routes within a selected area. The compaction area is defined by digitizing a starting point, which creates a pull bar and digitizing an end point which defines the end of the compaction area. The pull bar is used to pull the routes in a vertical or horizontal direction.

Arguments

w_windowId	Optional window ID in which the current cellview is displayed.

Return Values

`t`	Routes are pulled into a compaction area.
`nil`	Invalid window ID.

Example

```
leWeHiPull()
```

leWeHiRouteOptions

```
leWeHiRouteOptions(
     )
     => t/nil
```

Description

Opens the *Options — Route* form, which contains a *General* tab of options for routing commands, timing constraints, routing to cursor, interactive checking, auto shielding, push routing, push components and a text field which lets you enter a rules file and a do file. Also contains a *Bus* tab of options for enabling bus routing, enabling tandem pairs, via pattern fitting and spacing of gathered buses.

Arguments

None.

Return Values

`t`	*OK* or *Apply* are clicked in the form.
`nil`	*Cancel* is clicked in the form.

Example

```
leWeHiRouteOptions()
```

leWePathWidthMenuCB

```
leWePathWidthMenuCB(
     'matchPinWidth
     'matchPinWide
     'matchPinNarrow
     'matchPathWidth
     'specify
     'reset
     [w_windowId]
     => t/nil
```

Description

Callback function of the Layout menu *Width* item. When using the *Create – Path* command the *Width* option lets you match pin or path widths, or specify a width.

Arguments

`'matchPinWidth`	The callback of *Match Pin Off*.
`'matchPinWide`	The callback of *Match Pin Wide*.
`'matchPinNarrow`	The callback of *Match Pin Narrow*.
`'matchPathWidth`	The callback of *Match Path Width on/off*.
`'specify`	The callback of *Specify* width.
`'reset`	Resets the *Width* sub-menu.
w_windowId	Optional window ID in which the current cellview is displayed

Return Values

`t`	Sets the pin or path width. The `'specify` argument opens the *Create – Path* form, which lets you type in the width of the path.
`nil`	Invalid window ID.

Example

```
leWePathWidthMenuCB('matchPinWidth windowId)
```

leWePickUpDroppedWires

```
leWePickUpDroppedWires(
     )
     => t
```

Description

Restarts bus routing at a new location after paths have been dropped. Guides appear from the new starting points back to the dropped paths, which lets you route the all the paths from the new location. After routing the paths from the new location, you can route the dropped paths individually from the location where it was dropped to the location where the paths were restarted.

Arguments

None.

Return Values

`t`	Return value is always `t`.

Example

```
leWePickUpDroppedWires()
```

leWeRefresh

```
leWeRefresh(
     )
     => t/nil
```

Description

Notifies the software that the rules file or `.do` file has been changed.

Arguments

None.

Return Values

`t`	Command has been executed.
`nil`	No available cellview.

Example

```
leWeRefresh()
```

leWeReports

```
leWeReports
     )
     => t/nil
```

Description

Opens the *Reports* form, which displays reports about component placement and routing rules.

- *General* contains options for displaying reports which includes information about nets in the design, the unconnects, conflicts, rule violations, number of pins, vias, and length information for each routing layer.
- *Component* displays components placement information, nets and pins associated with the component and component pin positions.
- *Net* displays rules applied to specific nets, connections, coordinates, components, component pins, layers and vias.
- *Rules* displays information about routing rules applied to the current design.

Arguments

None.

Return Values

`t`	*OK* or *Apply* are clicked in the form.
`nil`	*Cancel* is clicked in the form.

Example

```
leWeReports(t)
```

leWeSetGatherBusWiresMenuItemText

```
leWeSetGatherBusWiresMenuItemText(
     b_toggle
     )
     => t
```

Description

Enables or disables the gathering of bus wires and set the text accordingly in the *Layout* pop up menu.

Arguments

b_toggle	When set to `t`, the wires are spaced by the amount specified in the *Spacing for Gather Bus Wires* settings and sets the text on the menu to *Turn gather bus wires on.* When set to `nil`, the spacing of the wires is determined by the pin spacing and the text on the *Layout* pop up menu is set to *Turn gather bus wires off.*

Return Values

`t`	Return value is always `t`.

Example

```
leWeSetGatherBusWiresMenuItemText(t)
```

leWeSetInteractiveCheckingText

```
leWeSetInteractiveCheckingText(
     b_toggle
     )
     => t
```

Description

Enables or disables checking of design rule violations and sets the *Turn checking on/off* text accordingly in the MMB button *Layout* pop up menu.

Arguments

b_toggle	When set to `t`, does not allow design rule violations and sets the *Layout* pop up menu text to *Turn checking on*. When set to `nil`, design rule violations are ignored and the text in the *Layout* pop up menu is set to *Turn checking off*.

Return Values

`t`	Return value is always `t`.

Example

```
leWeSetInteractiveCheckingText(t)
```

leWeToggle

```
leWeToggle(
      )
     =>t
```

Description

Toggles the environment between Virtuoso XL and the wiring editing enabled.

Arguments

None.

Return Values

`t`	Return value is always `t`.

Example

```
leWeToggle( )
```

7

Virtuoso Constraint Manager Functions

This chapter provides syntax, descriptions, and examples for the SKILL functions associated with the Virtuoso Constraint Manager.

Common Generator Functions on page 688

- cmxfGenIsId on page 688
- cmxfStopGen on page 689

Category Functions on page 690

- cmxfCatAttrGetDefVal on page 690
- cmxfCatAttrGetType on page 691
- cmxfCatGetId on page 692
- cmxfCatGetName on page 693
- cmxfCatGetNumAttrs on page 694
- cmxfCatGetWeight on page 695
- cmxfCatIsAttr on page 696
- cmxfCatIsId on page 697
- cmxfGenCatToCon on page 698
- cmxfGenSuperCat on page 699
- cmxfGenSuperCatToCat on page 700
- cmxfSCatGetId on page 701
- cmxfSCatGetName on page 702
- cmxfSCatIsId on page 703
- cmxfStartGenCatToCon on page 704

cmxfStartGenSuperCat on page 705

cmxfStartGenSuperCatToCat on page 706

Net Functions on page 707

cmxfGenNetToCon on page 707

cmxfNetIsConstrained on page 708

cmxfStartGenNetToCon on page 709

Fig Functions on page 710

cmxfFigIsConstrained on page 710

cmxfGenFigToCon on page 711

cmxfShapeNameGen on page 712

cmxfStartGenFigToCon on page 713

Axis Functions on page 714

cmxfAxisCreate on page 714

cmxfAxisDelete on page 716

cmxfAxisGetCV on page 717

cmxfAxisGetDirection on page 718

cmxfAxisGetId on page 719

cmxfAxisGetName on page 720

cmxfAxisGetX on page 721

cmxfAxisGetY on page 722

cmxfAxisIsActive on page 723

cmxfAxisIsConstrained on page 724

cmxfAxisIsHidden on page 725

cmxfAxisIsId on page 726

cmxfAxisIsInherited on page 727

cmxfAxisIsReadOnly on page 728

cmxfAxisIsValid on page 729
cmxfAxisModify on page 730
cmxfAxisNameGen on page 731
cmxfAxisSetActive on page 732
cmxfAxisSetDirection on page 733
cmxfAxisSetX on page 734
cmxfAxisSetY on page 735
cmxfGenAxis on page 736
cmxfGenAxisToCon on page 737
cmxfStartGenAxis on page 738
cmxfStartGenAxisToCon on page 739
Cluster Functions on page 740
cmxfClstrChngMems on page 740
cmxfClstrCreate on page 742
cmxfClstrDelete on page 743
cmxfClstrGetBBox on page 744
cmxfClstrGetCenter on page 745
cmxfClstrGetCV on page 746
cmxfClstrGetId on page 747
cmxfClstrGetName on page 748
cmxfClstrGetSize on page 749
cmxfClstrHasMems on page 750
cmxfClstrIsActive on page 751
cmxfClstrIsConstrained on page 752
cmxfClstrIsHidden on page 753
cmxfClstrIsId on page 754

cmxfClstrIsInherited on page 755

cmxfClstrIsMember on page 756

cmxfClstrIsReadOnly on page 757

cmxfClstrIsValid on page 758

cmxfClstrNameGen on page 759

cmxfClstrSetActive on page 760

cmxfGenClstr on page 761

cmxfGenClstrToCon on page 762

cmxfGenClstrToFig on page 763

cmxfGenClstrToMemName on page 764

cmxfStartGenClstr on page 765

cmxfStartGenClstrToCon on page 766

cmxfStartGenClstrToFig on page 767

cmxfStartGenClstrToMemName on page 768

Net-Class Functions on page 769

cmxfGenNC on page 769

cmxfGenNCToCon on page 770

cmxfGenNCToMemName on page 771

cmxfGenNCToNet on page 772

cmxfNCChngMems on page 773

cmxfNCCreate on page 774

cmxfNCDelete on page 775

cmxfNCGetCV on page 776

cmxfNCGetId on page 777

cmxfNCGetName on page 778

cmxfNCGetSize on page 779

cmxfNCHasMem on page 780
cmxfNCIsActive on page 781
cmxfNCIsConstrained on page 782
cmxfNCIsHidden on page 783
cmxfNCIsId on page 784
cmxfNCIsInherited on page 785
cmxfNCIsMember on page 786
cmxfNCIsReadOnly on page 787
cmxfNCIsValid on page 788
cmxfNCNameGen on page 789
cmxfNCSetActive on page 790
cmxfStartGenNC on page 791
cmxfStartGenNCToCon on page 792
cmxfStartGenNCToMemName on page 793
cmxfStartGenNCToNet on page 794

Constraint Functions on page 795

cmxfConChngMems on page 795
cmxfConCreate on page 797
cmxfConDelete on page 799
cmxfConGetAttrVal on page 800
cmxfConGetCatId on page 801
cmxfConGetCV on page 802
cmxfConGetId on page 803
cmxfConGetName on page 804
cmxfConGetRefFig on page 805
cmxfConGetWeight on page 806

cmxfConHasMems on page 807

cmxfConIsActive on page 808

cmxfConIsHidden on page 809

cmxfConIsId on page 810

cmxfConIsInherited on page 811

cmxfConIsOverCon on page 812

cmxfConIsSatisfied on page 814

cmxfConIsReadOnly on page 813

cmxfConIsValid on page 815

cmxfConModify on page 816

cmxfConNameGen on page 817

cmxfConSetActive on page 818

cmxfConSetAttrVal on page 819

cmxfConSetWeight on page 820

cmxfConVerify on page 821

cmxfGenCon on page 822

cmxfGenConToFig on page 823

cmxfGenConToMemName on page 824

cmxfGenConToNet on page 825

cmxfStartGenCon on page 826

cmxfStartGenConToFig on page 827

cmxfStartGenConToMemName on page 828

cmxfStartGenConToNet on page 829

Report and GUI Functions on page 830

cmxfAsc on page 830

cmxfGuiReportGen on page 832

cmxfGuiStartUp on page 833

Common Generator Functions

The following section provides Constraint Manager common generator SKILL functions that can be used with any Constraint Manager generator.

cmxfGenIsId

```
cmxfGenIsId(
     d_genId
     )
     => t/nil
```

Description

Verifies that the specified generator ID is valid.

Arguments

d_genId	Database ID of the generator.

Return Values

`t`	Generator ID is valid.
`nil`	Generator ID is not valid.

cmxfStopGen

```
cmxfStopGen(
     d_genId
     )
     => t/nil
```

Description

Stops the specified generator process.

Arguments

d_genId	Database ID of the generator.

Return Values

t	Stops the generator process.
nil	Generator ID is invalid or already not running.

Category Functions

The following section provides Constraint Manager category SKILL functions to query super categories and categories.

cmxfCatAttrGetDefVal

```
cmxfCatAttrGetDefVal(
     d_catId
     t_attrName
     )
     => g_attrVal/nil
```

Description

Queries the default value of a category attribute.

Arguments

d_catId	Database ID for the category.
t_attrName	Name of attribute for the specified category.

Return Values

g_attrVal	Default value of the attribute.
nil	Category ID is invalid or the attribute name is invalid.

cmxfCatAttrGetType

```
cmxfCatAttrGetType(
     d_catId
     t_attrName
     )
     => n_dbcType/nil
```

Description

Returns the type of attribute for the specified attribute name and category ID.

Arguments

d_catId	Database ID of the category.
t_attrName	Attribute name of the specified category.

Return Values

n_dbcType	Numeric value corresponding to database literal types.
nil	Category ID, or attribute name is invalid.

cmxfCatGetId

```
cmxfCatGetId(
     t_catName
     )
     => d_catId/nil
```

Description

Returns the category ID of the specified category.

Arguments

t_catName	Name of the category.

Return Values

d_catId	Database ID of the category.
nil	Category does not exist.

cmxfCatGetName

```
cmxfCatGetName(
     d_catId
     )
     => t_catName/nil
```

Description

Returns the name of the category specified by the category ID.

Arguments

d_catId	Database ID of the category.

Return Values

t_catName	Category name.
nil	Category is invalid.

cmxfCatGetNumAttrs

```
cmxfCatGetNumAttrs(
     d_catId
     )
     => n_numAttrs/nil
```

Description

Returns the number of attributes in the specified category.

Arguments

d_catId	Database ID of the category.

Return Values

n_numAttrs	Number of attributes in a category.
nil	Category ID is invalid.

cmxfCatGetWeight

```
cmxfCatGetWeight(
     d_cmxCatId
     )
     => x_conWeight/nil
```

Description

Returns the default weight of the specified category constraint.

Arguments

d_cmxCatId	Database ID of the category constraint.

Return Values

x_conWeight	Default weight of the category constraint.
nil	Category constraint ID is invalid.

cmxfCatIsAttr

```
cmxfCatIsAttr(
     d_catId
     t_attrName
     )
     => t/nil
```

Description

Verifies that the specified attribute name is a valid attribute for the specified category.

Arguments

d_catId	Database ID of the category.
t_attrName	Attribute name for the specified category.

Return Values

t	Attribute name is valid.
nil	Attribute name is invalid.

cmxfCatIsId

```
cmxfCatIsId(
     d_catId
     )
     => t/nil
```

Description

Verifies that the category ID is valid.

Arguments

d_catId	Database ID of the category.

Return Values

t	Category ID is valid.
nil	Category ID is invalid.

cmxfGenCatToCon

```
cmxfGenCatToCon(
     d_genId
     )
     => d_conId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenCatToCon()`, generates the next constraint ID.

Stop the generator after all constraints IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID for the generator.

Return Values

d_conId	Database ID for the constraint.
`nil`	No more constraints to generate, or generator ID is invalid.

cmxfGenSuperCat

```
cmxfGenSuperCat(
     d_genId
     )
     => d_scatId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenSuperCat()`, generates the next super category ID.

Stop the generator after all super category IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_scatId	Database ID of the super category.
`nil`	No more super category IDs to generate or the generator is invalid.

Example

```
cmxfGenSuperCat(cmxfStartGenSuperCat())
```

cmxfGenSuperCatToCat

```
cmxfGenSuperCatToCat(
     d_genId
     )
     => d_catId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenSuperCat()`, generates the next category ID.

Stop the generator after all category IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID for the generator.

Return Values

d_catId	Database ID for the category.
`nil`	No more categories to generate or the generator is invalid.

cmxfSCatGetId

```
cmxfSCatGetId(
     t_scatName
     )
     => d_scatId/nil
```

Description

Returns the super category ID specified by the super category name.

Arguments

t_scatName	Name of the super category.

Return Values

d_scatId	Database ID of the super category.
nil	Super category name is invalid.

cmxfSCatGetName

```
cmxfSCatGetName(
     d_scatId
     )
     => t_scatName/nil
```

Description

Returns the name of the super category specified by the super category ID.

Arguments

d_scatId	Database ID of the super category

Return Values

t_scatName	Name of the super category.
nil	Super category ID is invalid.

cmxfSCatIsId

```
cmxfSCatIsId(
     d_scatId
     )
     => t/nil
```

Description

Verifies that a given super category ID is valid.

Arguments

d_scatId	Database ID of the super category.

Return Values

`t`	Super category ID is valid.
`nil`	Super category ID is invalid.

cmxfStartGenCatToCon

```
cmxfStartGenCatToCon(
     d_cvId
     d_catId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating constraint IDs for the specified cellview and category.

Arguments

d_cvId	Database ID of the cellview.
d_catId	Database ID of the category.

Return Values

genId	Database ID of the new generator.
nil	No constraints to be generated, cellview ID is invalid, or category ID is invalid.

cmxfStartGenSuperCat

```
cmxfStartGenSuperCat(
     )
     => d_genId/nil
```

Description

Creates a new generator for generating super category IDs.

Arguments

None.

Return Values

d_genId	Database ID of the generator.
nil	No super categories to generate.

cmxfStartGenSuperCatToCat

```
cmxfStartGenSuperCatToCat(
     d_scatId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating category IDs.

Arguments

d_scatId	Database ID of the super category.

Return Values

d_genId	New database generator ID.
nil	No categories to be generated or the super category ID is invalid.

Net Functions

The following section provides Constraint Manager net SKILL functions to query constraints that might exist on nets.

cmxfGenNetToCon

```
cmxfGenNetToCon(
     d_genId
     )
     => d_conId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenNetToCon()`, generates the next constraint ID.

Stop the generator after all constraint IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_conId	Database ID of the constraint.
`nil`	No more constraints to generate, or generator ID is invalid.

cmxfNetIsConstrained

```
cmxfNetIsConstrained(
     d_netId
     )
     => t/nil
```

Description

Verifies that one or more constraints exist for the specified net.

Arguments

d_netId	Database ID for the net.

Return Values

`t`	One or more constraints exist on the net.
`nil`	No constraints exist on the net, or specified net ID is invalid.

cmxfStartGenNetToCon

```
cmxfStartGenNetToCon(
     d_netId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating constraint IDs for the specified net.

Arguments

d_netId	Database ID for the net.

Return Values

d_genId	Database ID of the new generator.
nil	No constraint to be generated, or net ID is invalid.

Fig Functions

The following section provides Constraint Manager fig SKILL functions to query constraints that might exist on figures.

cmxfFigIsConstrained

```
cmxfFigIsConstrained(
     d_figId
     )
     => t/nil
```

Description

Verifies that one or more constraints exist for the specified figure.

Arguments

d_figId	Database ID of the figure.

Return Values

`t`	One or more constraints exist on the figure.
`nil`	No constraints exist on the figure, or specified figure ID is not valid.

cmxfGenFigToCon

```
cmxfGenFigToCon(
     d_genId
     )
     => d_conId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenFigToCon()`, generates the next constraint ID.

Stop the generator after all constraint IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_conId	Database ID of the constraint.
`nil`	No more constraints to generate, or generator ID is invalid.

cmxfShapeNameGen

```
cmxfShapeNameGen(
     d_shapeId
     )
     => t_shapeName/nil
```

Description

Generates a name for the specified shape. If a name already exists for the shape, the same name is returned. Use as a wrapper around shape IDs whenever a shape name is needed.

Arguments

d_shapeId	Database ID of the shape.

Return Values

t_shapeName	Name for the specified shape ID.
nil	Shape ID is invalid.

cmxfStartGenFigToCon

```
cmxfStartGenFigToCon(
     d_figId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating constraint IDs for the specified figure ID.

Arguments

d_figId	Database ID for the figure.

Return Values

d_genId	Database ID of the generator.
nil	No constraints to be generated, or figure ID is invalid.

Axis Functions

The following section provides Constraint Manager axis SKILL functions which are used to create new axes, modify existing axes, remove axes, and perform query operations on them. An axis object contains a direction and a coordinate through which the axis passes. Horizontal `EW` (east-west) and vertical `NS` (north-south) directions are supported.

cmxfAxisCreate

```
cmxfAxisCreate(
     t_axisName
     d_cvId
     t_direction
     f_xPos
     f_yPos
     )
     => d_axisId/nil
```

Description

Creates a new axis by specifying an axis name, the cellview to which it will belongs, the axis direction, and the x and y values of the coordinate.

Arguments

t_axisName	Name of axis to be generated.
d_cvId	Database ID of the cellview.
t_direction	The direction of the axis. Valid Values: `NS` for a vertical axis, `EW` for a horizontal axis
f_xPos	Value for the horizontal coordinates of the axis.
f_yPos	Value for the vertical coordinates of the axis.

Return Values

d_axisId	Database ID of the new axis.

`nil`	Cellview ID is invalid, cellview is read-only, or direction is invalid. A nil can be used as the axis name in which case the name is auto-generated.

cmxfAxisDelete

```
cmxfAxisDelete(
     d_axisId
     )
     => t/nil
```

Description

Removes an existing axis. Constraints depending on this axis become invalid once the axis is removed.

Arguments

d_axisId	Database ID of the axis.

Return Values

`t`	Axis is removed.
`nil`	Axis ID is invalid, cellview is read-only, or axis is read-only.

cmxfAxisGetCV

```
cmxfAxisGetCV(
     d_axisId
     )
     => d_cvId/nil
```

Description

Returns the cellview ID of the cellview to which the axis belongs.

Arguments

d_axisId	Database ID of the axis.

Return Values

d_cvId	Database ID of the cellview.
nil	Axis ID is invalid.

cmxfAxisGetDirection

```
cmxfAxisGetDirection(
     d_axisId
     )
     => t_axisDirection/nil
```

Description

Returns the direction of the specified axis.

Arguments

d_axisId	Database ID of the axis.

Return Values

t_axisDirection	Axis direction: NS is a vertical axis, EW is a horizontal axis.
nil	Axis ID is invalid.

cmxfAxisGetId

```
cmxfAxisGetId(
     d_cvId
     t_axisName
     )
     => d_axisId/nil
```

Description

Returns the ID of axis specified by the cellview ID and the axis name.

Arguments

d_cvId	Database ID of the cellview.
t_axisName	Name of the axis.

Return Values

d_axisId	Database ID of the axis.
nil	Cellview ID or axis name is invalid.

cmxfAxisGetName

```
cmxfAxisGetName(
     d_axisId
     )
     => t_axisName/nil
```

Description

Returns the axis name specified by the axis ID.

Arguments

d_axisId	Database ID of the axis.

Return Values

t_axisName	Name of the axis.
nil	Axis ID is invalid.

cmxfAxisGetX

```
cmxfAxisGetX(
     d_axisId
     )
     => f_xPos/nil
```

Description

Returns the horizontal coordinate of the specified axis.

Arguments

d_axisId	Database ID of the axis.

Return Values

f_xPos	Horizontal coordinate of the axis.
nil	Axis ID is invalid.

cmxfAxisGetY

```
cmxfAxisGetY(
     d_axisId
     )
     => f_yPos/nil
```

Description

Returns the vertical coordinate of the specified axis.

Arguments

d_axisId	Database ID of the axis.

Return Values

f_yPos	Vertical coordinate of the axis.
nil	Axis ID is invalid.

cmxfAxisIsActive

```
cmxfAxisIsActive(
     d_axisId
     )
     => t/nil
```

Description

Verifies that the specified axis is currently active.

Arguments

d_axisId	Database ID of the axis.

Return Values

t	Specified axis is active.
nil	Axis ID is invalid, or axis is inactive.

cmxfAxisIsConstrained

```
cmxfAxisIsConstrained(
     d_axisId
     )
     => t/nil
```

Description

Verifies that one or more constraints exist for the specified axis.

Arguments

d_axisId	Database ID of the axis.

Return Values

`t`	One or more constraints exist on the axis.
`nil`	No constraints exist, or the axis ID is invalid.

cmxfAxisIsHidden

```
cmxfAxisIsHidden(
     d_axisId
     )
     => t/nil
```

Description

Verifies that the specified axis is hidden.

Arguments

d_axisId	Database ID of the axis.

Return Values

`t`	Axis is hidden.
`nil`	Axis is not hidden, or axis ID is invalid.

cmxfAxisIsId

```
cmxfAxisIsId(
     d_axisId
     )
     => t/nil
```

Description

Verifies that the specified axis ID is valid.

Arguments

d_axisId	Database ID of the axis.

Return Values

`t`	Axis ID is valid.
`nil`	Axis ID is invalid.

cmxfAxisIsInherited

```
cmxfAxisIsInherited(
     d_axisId
     )
     => t/nil
```

Description

Verifies that the specified axis ID is inherited from another cellview.

Arguments

d_axisId	Database ID of the axis.

Return Values

`t`	Specified axis is inherited.
`nil`	Axis ID is invalid, or axis is native.

cmxfAxisIsReadOnly

```
cmxfAxisIsReadOnly(
     d_axisId
     )
     => t/nil
```

Description

Verifies that the specified axis is read-only.

Arguments

d_axisId	Database ID of the axis.

Return Values

t	Axis is read-only.
nil	Axis is not read-only, or axis ID is invalid.

cmxfAxisIsValid

```
cmxfAxisIsValid(
     d_axisId
     )
     => t/nil
```

Description

Verifies that the specified axis is valid.

Arguments

d_axisId	Database ID of the axis.

Return Values

`t`	Axis is valid.
`nil`	Axis is invalid.

cmxfAxisModify

```
cmxfAxisModify(
     d_axisId
     t_direction
     f_xPos
     f_yPos
     )
     => t/nil
```

Description

Modifies the direction and coordinate values of an existing axis specified by the axis ID. Constraints depending on this axis, which are eligible for verification will be reverified.

Arguments

d_axisId	Database ID of the axis.
t_direction	The direction of the axis. Valid Values: `NS` for a vertical axis, `EW` for a horizontal axis.
f_xPos	Value for the horizontal coordinates of the axis.
f_yPos	Value for the vertical coordinates of the axis.

Return Values

`t`	Modifications are made.
`nil`	Axis ID is invalid, axis or cellview is read-only, or direction is invalid.

cmxfAxisNameGen

```
cmxfAxisNameGen(
     d_cvId
     )
     => t_axisName/nil
```

Description

Generates a unique name for a new axis.

Arguments

d_cvId	Database ID of the cellview.

Return Values

t_axisName	New axis name.
nil	Cellview ID is invalid.

cmxfAxisSetActive

```
cmxfAxisSetActive(
     d_axisId
     g_bool
     )
     => t/nil
```

Description

Controls the active state of the specified axis. If *g_bool* is `t`, the inactive axis is set active. If *g_bool* is `nil`, the axis is set as inactive. Making an active axis inactive will make all dependent constraints inactive. Making an inactive axis active will not make dependent constraints active.

Arguments

d_axisId	Database ID of the axis.
g_bool	Indicates whether to set the axis active (`t`) or inactive (`nil`).

Return Values

`t`	Active status is changed.
`nil`	Axis ID is invalid, axis ID is read-only, or a value other than `t` or `nil` was specified for *g_bool*.

cmxfAxisSetDirection

```
cmxfAxisSetDirection(
     d_axisId
     t_direction
     )
     => t/nil
```

Description

Sets a new direction for an existing axis. Constraints depending on this axis, which are eligible for verification will be reverified.

Arguments

d_axisId	Database ID of the axis.
t_direction	The direction of the axis. Valid Values: `NS` for a vertical axis, `EW` for a horizontal axis

Return Values

`t`	Changes the direction of the specified axis is changed.
`nil`	Axis ID is invalid.

cmxfAxisSetX

```
cmxfAxisSetX(
     d_axisId
     f_xPos
     )
     => t/nil
```

Description

Sets a new horizontal coordinate for the specified axis. Constraints depending on this axis, which are eligible for verification will be reverified.

Arguments

d_axisId	Database ID of the axis.
f_xPos	Value for the horizontal coordinate.

Return Values

t	The horizontal coordinate of the specified axis is changed.
nil	Axis ID is invalid.

cmxfAxisSetY

```
cmxfAxisSetY(
     d_axisId
     f_xPos
     )
     => t/nil
```

Description

Sets a new vertical coordinate for the specified axis. Constraints depending on this axis, which are eligible for verification will be reverified.

Arguments

d_axisId	Database ID of the axis.
f_xPos	Value for the vertical coordinate.

Return Values

`t`	The vertical coordinate of the specified axis is changed.
`nil`	Axis ID is invalid.

cmxfGenAxis

```
cmxfGenAxis(
     d_genId
     )
     => d_axisId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenAxis()`, generates the next axis ID.

Stop the generator after all axis IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the cellview.

Return Values

d_axisId	Database ID of the next axis.
`nil`	No more axes to generate, or generator ID is invalid.

cmxfGenAxisToCon

```
cmxfGenAxisToCon(
     d_genId
     )
     => d_conId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenAxisToCon()`, generates the next constraint ID.

Stop the generator after all constraint IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_conId	Database ID of the new constraint.
`nil`	No constraints to generate, or generator ID is invalid.

cmxfStartGenAxis

```
cmxfStartGenAxis(
     d_cvId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating axis IDs for the specified cellview.

Arguments

d_cvId	Database ID of the cellview.

Return Values

d_genId	Database ID of the new generator.
nil	No axes exist, or cellview ID is invalid.

cmxfStartGenAxisToCon

```
cmxfStartGenAxisToCon(
     d_axisId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating constraint IDs for the specified axis.

Arguments

d_axisId	Database ID of the axis.

Return Values

d_genId	Database ID of the new generator.
nil	No constraints to generate, or axis ID is invalid.

Cluster Functions

The following section provides Constraint Manager cluster SKILL functions, which are used to create new clusters, change membership, remove clusters, and perform query operations.

cmxfClstrChngMems

```
cmxfClstrChngMems(
     d_clstrId
     l_instNames
     l_shapeNames
     l_pinNames
     l_clstrNames
     )
     => t/nil
```

Description

Changes membership for an existing cluster by specifying the cluster ID and the four lists providing names of members. Only, instances, shapes, pins, and other clusters can be members of a cluster. Constraints depending on this cluster, which are eligible for verification will be reverified.

Arguments

d_clstrId	Database ID of the cluster.
l_instNames	Instances which are members of the cluster. `nil` can be passed as an argument if there are no members of this type.
l_shapeNames	Shapes which are members of the cluster. `nil` can be passed as an argument if there are no members of this type.
l_pinNames	Pins which are members of the cluster. `nil` can be passed as an argument if there are no members of this type.
l_clstrNames	C lusters which are members of the cluster. `nil` can be passed as an argument if there are no members of this type.

Return Values

`t`	The membership of the cluster is changed.
`nil`	Cluster ID is invalid, cellview is read-only, or if any of the member names in the lists is invalid.

cmxfClstrCreate

```
cmxfClstrCreate(
     t_clstrName
     d_cvId
     l_instNames
     l_shapeNames
     l_pinNames
     l_clstrNames
     )
     => d_clstrId/nil
```

Description

Creates a new cluster by specifying a cluster name, the cellview ID, and the four lists providing names of members. Only, instances, shapes, pins, and other clusters can be members of a cluster.

Arguments

t_clstrName	Name of new cluster to be created.
d_cvId	Database ID of the cellview.
l_instNames	Instances to be added to the new cluster. `nil` can be passed as an argument if there are no members of this type.
l_shapeName	Shapes to be added to the new cluster. `nil` can be passed as an argument if there are no members of this type.
l_pinNames	Pins to be added to the new cluster. `nil` can be passed as an argument if there are no members of this type.
l_clstrNames	Clusters to be added to the new cluster. `nil` can be passed as an argument if there are no members of this type.

Return Values

d_clstrId	Database ID of the new cluster.
`nil`	Cellview ID is invalid, member names encountered in the list supplied are invalid, or cellview is read-only.

cmxfClstrDelete

```
cmxfClstrDelete(
     d_clstrId
     )
     => t/nil
```

Description

Removes the specified cluster. Constraints depending on this cluster become invalid when the cluster is removed.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	Specified cluster is deleted.
`nil`	Cluster ID is invalid, cluster is read-only, or cellview is read-only.

cmxfClstrGetBBox

```
cmxfClstrGetBBox(
     d_clstrId
     )
     => l_bBox/nil
```

Description

Returns the bounding box of the specified cluster.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

l_bBox	Bounding box of the cluster, in the format ((*left bottom*) (*right top*)), where *left*, *bottom*, *right*, and *top* are floating-point values.
nil	Cluster ID is invalid.

cmxfClstrGetCenter

```
cmxfClstrGetCenter(
     d_clstrId
     )
     => l_center/nil
```

Description

Returns the point that represents the center of the cluster bounding box.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

l_center	Returns the center of the cluster bounding box in the format (*x y*) where *x* and *y* are floating point values.
nil	Cluster ID is invalid.

cmxfClstrGetCV

```
cmxfClstrGetCV(
     d_clstrId
     )
     => d_cvId/nil
```

Description

Returns the cellview ID to which the specified cluster belongs.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

d_cvId	Database ID of the cellview.
nil	Cluster ID is invalid.

cmxfClstrGetId

```
cmxfClstrGetId(
     d_cvId
     t_clstrName
     )
     => d_clstrId/nil
```

Description

Returns the database ID of the specified cluster.

Arguments

d_cvId	Database ID of the cellview.
t_clstrId	Name of the cluster.

Return Values

d_clstrId	Database ID of the specified cluster.
nil	Cellview ID or cluster name is invalid.

cmxfClstrGetName

```
cmxfClstrGetName(
     d_clstrId
     )
     => t_clstrName/nil
```

Description

Returns the name of the cluster specified by the cluster ID.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

t_clstrName	Name of the cluster specified by the cluster ID.
nil	Cluster ID is invalid.

cmxfClstrGetSize

```
cmxfClstrGetSize(
     d_clstrId
     )
     => n_numMembers/nil
```

Description

Returns the number of flattened members in the specified cluster.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

n_numMembers	Number of flattened members in the specified cluster.
nil	Cluster ID is invalid.

cmxfClstrHasMems

```
cmxfClstrHasMems(
     d_clstrId
     t_memType
     )
     => t/nil
```

Description

Verifies that members of specified type exist for the specified cluster.

Arguments

d_clstrId	Database ID of the cluster.
t_memType	Member type. Valid Values: `instance`, `shape`, `pin`, `cluster`

Return Values

`t`	One or more members of specified type exist for the cluster.
`nil`	No members of specified type exist for the cluster, cluster ID is invalid, or specified member type is invalid.

cmxfClstrIsActive

```
cmxfClstrIsActive(
     d_clstrId
     )
     => t/nil
```

Description

Verifies that the specified cluster is currently active.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	Specified cluster is active.
`nil`	Specified cluster is inactive, or cluster ID is invalid.

cmxfClstrIsConstrained

```
cmxfClstrIsConstrained(
     d_clstrId
     )
     => t/nil
```

Description

Verifies that one or more constraints exist for the specified cluster.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	One or more constraints exist on the cluster.
`nil`	No constraints exist, or cluster ID is invalid.

cmxfClstrIsHidden

```
cmxfClstrIsHidden(
     d_clstrId
     )
     => t/nil
```

Description

Verifies that the specified cluster is hidden.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	Cluster is hidden.
`nil`	Cluster is not hidden, or cluster ID is invalid.

cmxfClstrIsId

```
cmxfClstrIsId(
     d_clstrId
     )
     => t/nil
```

Description

Verifies that the specified cluster is valid.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	Constraint ID is valid.
`nil`	Constraint ID is invalid.

cmxfClstrIsInherited

```
cmxfClstrIsInherited(
     d_clstrId
     )
     => t/nil
```

Description

Verifies that the specified cluster is inherited from another cellview.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	Cluster is inherited from another cellview.
`nil`	Cluster is native, or cluster ID is invalid.

cmxfClstrIsMember

```
cmxfClstrIsMember(
     d_clstrId
     d_figId
     )
     => t/nil
```

Description

Verifies that the specified figure is a member of the cluster specified by *d_clstrId*. The figure may be a direct member of the cluster or may belong to any sub-cluster in the cluster hierarchy.

Arguments

d_clstrId	Database ID of the cluster.
d_figId	Database ID of the figure.

Return Values

t	Figure is a valid member of the cluster.
nil	Figure is not a member of the cluster, or cluster ID or figure ID is invalid.

cmxfClstrIsReadOnly

```
cmxfClstrIsReadOnly(
     d_clstrId
     )
     => t/nil
```

Description

Verifies that the specified cluster is read-only.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	Cluster is read-only.
`nil`	Cluster is not read-only, or cluster ID is invalid.

cmxfClstrIsValid

```
cmxfClstrIsValid(
     d_clstrId
     )
     => t/nil
```

Description

Verifies that the specified cluster is valid.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

`t`	Cluster is valid.
`nil`	Cluster is invalid, or cluster ID is invalid.

cmxfClstrNameGen

```
cmxfClstrNameGen(
     d_cvId
     )
     => t_clstrName/nil
```

Description

Generates a unique name for a new cluster.

Arguments

d_cvId	Database ID of the cellview.

Return Values

t_clstrName	Name of the new cluster specified by the cellview ID.
nil	Cellview ID is invalid.

cmxfClstrSetActive

```
cmxfClstrSetActive(
     d_clstrId
     g_bool
     )
     => t/nil
```

Description

Controls the active state of the specified cluster. If *g_bool* is `t`, the cluster is set active if it was inactive. If *g_bool* is `nil`, the cluster is set as inactive. Making an active cluster inactive will make all dependent constraints inactive. Making an inactive cluster active will not make dependent constraints active.

Arguments

d_clstrId	Database ID of the cluster.
g_bool	Indicates whether to set the cluster active (`t`) or inactive (`nil`).

Return Values

`t`	Status of the cluster is changed.
`nil`	Cluster is invalid, cluster is read-only, or a value other than `t` or `nil` was specified for *g_bool*.

cmxfGenClstr

```
cmxfGenClstr(
     d_genId
     )
     => d_clstrId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenClstr()`, generates the next cluster ID.

Stop the generator after all cluster IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_clstrId	Database ID for the new cluster.
`nil`	No clusters to be generated or if the generator ID is invalid.

cmxfGenClstrToCon

```
cmxfGenClstrToCon(
     d_genId
     )
     => d_conId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenClstrToCon()`, generates the next constraint ID.

Stop the generator after all constraint IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_conId	Database ID for the next constraint.
`nil`	No constraints to be generated, or generator ID is invalid.

cmxfGenClstrToFig

```
cmxfGenClstrToFig(
     d_genId
     )
     => d_figId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenClstrToFig()`, generates the next figure ID.

Stop the generator after all figure IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_figId	Database ID for the next figure.
`nil`	No figures to be generated, or generator ID is invalid.

cmxfGenClstrToMemName

```
cmxfGenClstrToMemName(
     d_genId
     )
     => t_memType/nil
```

Description

Given the generator ID obtained from `cmxfStartGenClstrToMemName()`, generates the next member name.

Stop the generator after all member names have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

t_memType	Member type. Valid Values: `instance`, `shape`, `pin`, `cluster`
`nil`	No more member names to generate, or generator is invalid.

cmxfStartGenClstr

```
cmxfStartGenClstr(
     d_cvId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating cluster IDs for the specified cellview.

Arguments

d_cvId	Database ID of the cellview.

Return Values

d_genId	Database ID for the new generator.
nil	No clusters to be generated, or cellview ID is invalid.

cmxfStartGenClstrToCon

```
cmxfStartGenClstrToCon(
     d_clstrId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating constraint IDs for the specified cluster.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

d_genId	Database ID for the new generator.
nil	No constraints to be generated, or cluster ID is invalid.

cmxfStartGenClstrToFig

```
cmxfStartGenClstrToFig(
     d_clstrId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating flattened figure IDs for the specified cluster.

Arguments

d_clstrId	Database ID of the cluster.

Return Values

d_genId	Database ID for the new generator.
nil	No figures to be generated, or cluster ID is invalid.

cmxfStartGenClstrToMemName

```
cmxfStartGenClstrToMemName(
     d_clstrId
     t_memType
     )
     => d_genId/nil
```

Description

Creates a new generator for generating member names for the specified cluster.

Arguments

d_clstrId	Database ID of the cluster.
t_memType	Member type. Valid Values: `instance`, `shape`, `pin`, `cluster`

Return Values

d_genId	Database ID for the new generator.
`nil`	No members of specified type exist for the cluster, cluster ID is invalid, or specified member type is invalid.

Net-Class Functions

The following section provides Constraint Manager net-class SKILL functions which are used to create new net-classes, change membership, remove net-classes, and perform query operations.

cmxfGenNC

```
cmxfGenNC(
     d_genId
     )
     => d_ncId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenNC()`, generates the next net-class ID.

Stop the generator after all net-class IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_ncId	Database ID of the net-class.
`nil`	No more net-classes to generate, or generator is invalid.

cmxfGenNCToCon

```
cmxfGenNCToCon(
     d_genId
     )
     => d_conId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenNCToCon()`, generates the next constraint ID.

Stop the generator after all constraint IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_conId	Database ID of the generated constraint.
`nil`	No more constraints to generate or the generator ID is invalid.

cmxfGenNCToMemName

```
cmxfGenNCToMemName(
     d_genId
     )
     => t_memName/nil
```

Description

Given the generator ID obtained from `cmxfStartGenNCToMemName()`, generates the next member name.

Stop the generator after all member names have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

t_memName	Next member name.
`nil`	No more member names to generate or the generator ID is invalid.

cmxfGenNCToNet

```
cmxfGenNCToNet(
     d_genId
     )
     => d_netId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenNCToNet()`, generates the next net ID.

Stop the generator after all net IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_netId	Database ID of the generated net.
`nil`	No more nets to generate or the generator ID is invalid.

cmxfNCChngMems

```
cmxfNCChngMems(
     d_ncId
     l_netNames
     )
     => t/nil
```

Description

Changes the membership of an existing net-class by providing a new list of net names. Changing the membership of existing net-class will result in all dependent and eligible constraints to be reverified if verification is in place.

Arguments

d_ncId	Database ID of the net-class.
l_netNames	List of net names. These become members of the new net-class.

Return Values

t	Successfully changes the membership status of the net-class.
nil	Net-class is invalid, list of net names is invalid, any membership rules are violated, the net-class is read-only or the cellview is read-only.

cmxfNCCreate

```
cmxfNCCreate(
     t_ncName
     d_cvId
     l_netNames
     )
     => d_ncId/nil
```

Description

Creates a new net-class by specifying a net-class name, the cellview to which the nets belong, and the list of nets to be added to the class.

Arguments

t_ncName	Name of the new net-class.
d_cvId	Database ID of the cellview.
l_netNames	Net names to become members of the new net-class.

Return Values

d_ncId	Database ID of the net-class.
nil	A net-class with the same name already exists, the cellview ID is invalid, any of the net names listed are invalid, no names are specified, net-class membership rules are not met, or cellview is read-only.

cmxfNCDelete

```
cmxfNCDelete(
     d_ncId
     )
     => t/nil
```

Description

Removes the specified net-class. Deleting a net-class marks all dependent constraints as invalid.

Arguments

d_ncId	Database ID of the net-class.

Return Values

`t`	Net-class is removed.
`nil`	Net-class ID is invalid, net-class is read-only, or cellview is read-only.

cmxfNCGetCV

```
cmxfNCGetCV(
     d_ncId
     )
     => d_cvId/nil
```

Description

Returns the cellview ID of the specified net-class.

Arguments

d_ncId	Database ID of the net-class.

Return Values

d_cvId	Database ID of the cellview.
nil	Net-class ID is invalid.

cmxfNCGetId

```
cmxfNCGetId(
     d_cvId
     t_ncName
     )
     => d_ncId/nil
```

Description

Returns the net-class ID.

Arguments

d_cvId	Database ID of the cellview.
t_ncName	Name of the net-class.

Return Values

d_ncId	Database ID of the net-class.
nil	Net-class does not exist, cellview ID is invalid, or net-class name is invalid.

cmxfNCGetName

```
cmxfNCGetName(
     d_ncId
     )
     => t_ncName/nil
```

Description

Returns the name of the net-class.

Arguments

d_ncId	Database ID of the net-class.

Return Values

t_ncName	Name of the net-class specified by the net-class ID.
nil	Net-class ID is invalid.

cmxfNCGetSize

```
cmxfNCGetSize(
     d_ncId
     )
     => n_numNets/nil
```

Description

Returns the number of nets in the specified net-class.

Arguments

d_ncId	Database ID of the net-class.

Return Values

n_numNets	Number of nets belonging to the specified net-class.
nil	Net-class ID is invalid.

cmxfNCHasMem

```
cmxfNCHasMem(
     d_ncId
     )
     => t/nil
```

Description

Verifies that one or more net members exist for the specified net-class.

Arguments

d_ncId	Database ID for the net-class.

Return Values

`t`	One or more nets member exist for the net-class.
`nil`	No nets member exist for the net-class, or net-class ID is invalid.

cmxfNCIsActive

```
cmxfNCIsActive(
     d_ncId
     )
     => t/nil
```

Description

Verifies that the specified net-class is valid.

Arguments

d_ncId	Database ID of the net-class.

Return Values

`t`	Net-class is valid.
`nil`	Net-class is invalid, or net-class ID is invalid.

cmxfNCIsConstrained

```
cmxfNCIsConstrained(
     d_ncId
     )
     => t/nil
```

Description

Verifies that the specified net-class is constrained.

Arguments

d_ncId	Database ID of the net-class.

Return Values

`t`	One or more constraints exist on the net-class.
`nil`	No constraints exist, or net-class ID is invalid.

cmxfNCIsHidden

```
cmxfNCIsHidden(
     d_ncId
     )
     => t/nil
```

Description

Verifies that the specified net-class is hidden.

Arguments

d_ncId	Database ID of the net-class.

Return Values

`t`	Net-class is hidden.
`nil`	Net-class is hidden, or net-class ID is invalid.

cmxfNCIsId

```
cmxfNCIsId(
     d_ncId
     )
     => t/nil
```

Description

Verifies that the specified net-class ID is valid.

Arguments

d_ncId	Database ID for the net-class.

Return Values

`t`	Net-class ID is valid.
`nil`	Net-class ID is invalid.

cmxfNCIsInherited

```
cmxfNCIsInherited(
     d_ncId
     )
     => t/nil
```

Description

Verifies that the specified net-class is inherited from another cellview.

Arguments

d_ncId	Database ID of the net-class.

Return Values

t	Net-class is inherited.
nil	Net-class is native, or net-class ID is invalid

cmxfNCIsMember

```
cmxfNCIsMember(
     d_ncId
     d_netId
     )
     => t/nil
```

Description

Verifies that the specified net is a member of the specified net-class.

Arguments

d_ncId	Database ID of the net-class.
d_netId	Database ID of the net.

Return Values

`t`	Specified net is a valid member of the net-class.
`nil`	Specified net is not a member of the net-class, net-class ID is invalid, or net ID is invalid.

cmxfNCIsReadOnly

```
cmxfNCIsReadOnly(
     d_ncId
     )
     => t/nil
```

Description

Verifies that the specified net-class is read-only.

Arguments

d_conId	Database ID of the net-class.

Return Values

`t`	Net-class is read-only.
`nil`	Net-class is not read-only, or net-class ID is invalid.

cmxfNCIsValid

```
cmxfNCIsValid(
     d_ncId
     )
     => t/nil
```

Description

Verifies that the specified net-class is valid.

Arguments

d_ncId	Database ID of the net-class.

Return Values

`t`	Net-class is valid.
`nil`	Net-class is invalid.

cmxfNCNameGen

```
cmxfNCNameGen(
     d_cvId
     )
     => t_ncName/nil
```

Description

Generates a unique name for a new net-class.

Arguments

d_cvId	Database ID of the cellview.

Return Values

t_ncName	Name of the new net-class.
nil	Cellview ID is invalid.

cmxfNCSetActive

```
cmxfNCSetActive(
     d_ncId
     g_bool
     )
     => t/nil
```

Description

Controls the active state of the specified net-class. If *g_bool* is `t`, the inactive net-class is set active. If *g_bool* is `nil`, the net-class is set inactive. Making an active net-class inactive will make all dependent constraints inactive. Making an inactive net-class active will not make dependent constraints active.

Arguments

d_ncId	Database ID of the net-class.
g_bool	Indicates whether to set the net-class active (`t`) or inactive (`nil`)

Return Values

`t`	The active status is changed.
`nil`	Net-class ID is invalid, *g_bool* has a value other than `t` or `nil`, net-class is read-only, or cellview is read-only.

cmxfStartGenNC

```
cmxfStartGenNC(
     d_cvId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating net-class IDs for the specified cellview.

Arguments

d_cvId	Database ID of the cellview.

Return Values

d_genId	Database ID of the new generator.
nil	No net-classes to be generated, or cellview ID is invalid.

cmxfStartGenNCToCon

```
cmxfStartGenNCToCon(
     d_ncId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating constraint IDs for the specified net-class.

Arguments

d_ncId	Database ID of the net-class.

Return Values

d_genId	Database ID of the new generator.
nil	No constraints to be generated, or net-class ID is invalid.

cmxfStartGenNCToMemName

```
cmxfStartGenNCToMemName(
     d_ncId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating member names for the specified net-class.

Arguments

d_ncId	Database ID for the net-class.

Return Values

d_genId	Database ID of the new generator.
nil	No net members, or net-class ID is invalid.

cmxfStartGenNCToNet

```
cmxfStartGenNCToNet(
     d_ncId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating net IDs for the specified net-class.

Arguments

d_ncId	Database ID of the net-class.

Return Values

d_genId	Database ID of the generator.
nil	No nets to be generated, or net-class ID is invalid.

Constraint Functions

The following section provides Constraint Manager constraint SKILL functions, which are used to create new constraints, change membership, remove constraints, manipulate constraints, and perform query operations.

cmxfConChngMems

```
cmxfConChngMems(
     d_conId
     l_netNames
     l_ncNames
     l_instNames
     l_shapeNames
     l_pinNames
     l_clstrNames
     )
     => t/nil
```

Description

Changes the members of the specified constraint. New membership is specified using six lists holding names for nets, net-classes, instances, shapes, pins, and clusters in order. After a successful change in membership, the constraint if eligible for verification, will be reverified.

Arguments

d_conId	Database ID of the constraint.
l_netNames	Nets which are members of the constraint. nil can be passed as an argument if there are no members of this type.
l_ncNames	Net-classes which are members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_instNames	Instances which are members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_shapeNames	Shapes which are members of the constraint. `nil` can be passed as an argument if there are no members of this type.

l_pinNames	Pins which are members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_clstrNames	Clusters which are members of the constraint. `nil` can be passed as an argument if there are no members of this type.

Return Values

`t`	The membership of the constraint is changed.
`nil`	Cellview is read-only, constraints are read-only, or constraint ID is invalid.

cmxfConCreate

```
cmxfConCreate(
     t_conName
     d_cvId
     t_catName
     l_netNames
     l_ncNames
     l_instNames
     l_shapeNames
     l_pinNames
     l_clstrNames
     l_attrNVL
     )
     => d_conId/nil
```

Description

Creates a new constraint. The member lists are expected in the order specified. The last argument is an attribute name value list that supplies a sequence of name-value pairs. Each name-value pair is a list of two components; an attribute name and the value of the attribute. Category membership rules are enforced at constraint creation time.

Arguments

t_conName	Name of constraint to be created.
d_cvId	Database ID of the cellview.
t_catName	Categories to be members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_netNames	Nets to be members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_ncName	Net-classes to be members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_instNames	Instances to be members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_shapeNames	Shapes to be members of the constraint. `nil` can be passed as an argument if there are no members of this type.

l_pinNames	Pins to be members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_clstrNames	Clusters to be members of the constraint. `nil` can be passed as an argument if there are no members of this type.
l_attrNVL	Attribute name-value list. The attribute name-value pairs can be specified in any order and missing attributes are automatically assigned default values from the attribute information in the category.

Return Values

d_conId	Database ID of the new constraint.
`nil`	Invalid membership rules, invalid attributes values, cellview ID is invalid, specified lists names are invalid, or cellview is read-only.

cmxfConDelete

```
cmxfConDelete(
     d_conId
     )
     => t/nil
```

Description

Removes the specified constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

t	The specified constraint is deleted.
nil	Constraint ID is invalid, or cellview is read-only.

cmxfConGetAttrVal

```
cmxfConGetAttrVal(
     d_conId
     t_attrName
     )
     => g_attrVal/nil
```

Description

Returns the value of a constraint attribute.

Arguments

d_conId	Database ID of the constraint.
t_attrName	Name of attribute.

Return Values

g_attrVal	Value of constraint attribute.
nil	Constraint is invalid, constraint ID is invalid, or attribute name is invalid.

cmxfConGetCatId

```
cmxfConGetCatId(
     d_conId
     )
     => d_catId/nil
```

Description

Returns the category ID of the specified constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

d_catId	Database ID of the category.
nil	Constraint ID is invalid.

cmxfConGetCV

```
cmxfConGetCV(
     d_conId
     )
     => d_cvId/nil
```

Description

Returns the cellview ID of the specified constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

d_cvId	Database ID of the cellview.
nil	Constraint ID is invalid

cmxfConGetId

```
cmxfConGetId(
     d_cvId
     t_catName
     t_conName
     )
     => d_conId/nil
```

Description

Returns the constraint ID of the specified constraint.

Arguments

d_cvId	Database ID of the cellview.
t_catName	Name of category name.
t_conName	Name of constraint.

Return Values

d_conId	Database ID of the constraint.
nil	Cellview ID, category, or constraint is invalid.

cmxfConGetName

```
cmxfConGetName(
     d_conId
     )
     => t_conName/nil
```

Description

Returns the name of the constraint specified by the constraint ID.

Arguments

d_conId	Database ID of the constraint.

Return Values

t_conName	Name of specified constraint.
nil	Constraint ID is invalid.

cmxfConGetRefFig

```
cmxfConGetRefFig(
     d_conId
     )
     => d_figId/nil
```

Description

Returns the database figure ID of the reference figure for the specified constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

d_figId	Database figure ID of the reference figure.
nil	Constraint does not have a reference figure, or constraint ID is invalid.

cmxfConGetWeight

```
cmxfConGetWeight(
     d_conId
     )
     => x_conWeight/nil
```

Description

Returns the integer weight of the specified constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

x_conWeight	Integer weight of the constraint.
nil	0 is returned if the constraint ID is invalid.

cmxfConHasMems

```
cmxfConHasMems(
     d_conId
     t_memType
     )
     => t/nil
```

Description

Verifies whether one or more members exist for the specified constraint.

Arguments

d_conId	Database ID of the constraint.
t_memType	Member type: Valid Values: `net`, `net-class`, `instance`, `shape`, `pin`, `cluster`

Return Values

`t`	One or more members of specified type exist for the constraint.
`nil`	No members of specified type exist for the constraint, constraint ID is invalid, or specified member type is invalid.

cmxfConIsActive

```
cmxfConIsActive(
     d_conId
     )
     => t/nil
```

Description

Verifies the constraint is active.

Arguments

d_conId	Database ID of the constraint.

Return Values

t	Constraint is active.
nil	Constraint is inactive, or constraint ID is invalid.

cmxfConIsHidden

```
cmxfConIsHidden(
     d_conId
     )
     => t/nil
```

Description

Verifies that the specified constraint is hidden.

Arguments

d_conId	Database ID of the constraint.

Return Values

`t`	Constraint is hidden.
`nil`	Constraint is not hidden, or constraint ID is invalid.

cmxfConIsId

```
cmxfConIsId(
     d_conId
     )
     => t/nil
```

Description

Verifies that the specified constraint ID is valid.

Arguments

d_conId	Database ID of the constraint.

Return Values

t	Constraint ID is valid.
nil	Constraint ID is invalid

cmxfConIsInherited

```
cmxfConIsInherited(
     d_conId
     )
     => t/nil
```

Description

Verifies that the constraint is an inherited constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

`t`	Constraint is an inherited constraint.
`nil`	Constraint is not an inherited constraint, or constraint ID is invalid.

cmxfConIsOverCon

```
cmxfConIsOverCon(
     d_conId
     )
     => t/nil
```

Description

Verifies that the specified constraint is an over constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

`t`	Constraint is an over constraint.
`nil`	Constraint is not an over constraint, or constraint ID is invalid.

cmxfConIsReadOnly

```
cmxfConIsReadOnly(
     d_conId
     )
     => t/nil
```

Description

Verifies that the specified constraint is read-only.

Arguments

d_conId	Database ID of the constraint.

Return Values

`t`	Constraint is read-only.
`nil`	Constraint is not read-only, or constraint ID is invalid.

cmxfConIsSatisfied

```
cmxfConIsSatisfied(
     d_conId
     )
     => t/nil
```

Description

Verifies that the specified constraint is currently satisfied.

Arguments

d_conId	Database ID of the constraint.

Return Values

`t`	Constraint is satisfied.
`nil`	Constraint is not satisfied, or constraint ID is invalid.

cmxfConIsValid

```
cmxfConIsValid(
     d_conId
     )
     => t/nil
```

Description

Verifies that the specified constraint is valid.

Arguments

d_conId	Database ID of the constraint.

Return Values

`t`	Constraint is valid.
`nil`	Constraint is invalid.

cmxfConModify

```
cmxfConModify(
     d_conId
     l_attrNVL
     )
     => t/nil
```

Description

Modifies the attribute values for the specified constraint. The new attribute name-value pairs are supplied as the second argument list. Only the attributes specified in the list will be modified, all others retain their existing values. After successful modification, the constraint, if eligible for verification, will be reverified.

Arguments

d_conId	Database ID of the constraint.
l_attrNVL	Attribute name-value list. The attribute name-value pairs can be specified in any order.

Return Values

t	The specified constraint is modified.
nil	Constraint ID is invalid, attribute name or value is invalid, constraint is read-only, or cellview is read-only.

cmxfConNameGen

```
cmxfConNameGen(
     d_cvId
     d_catId
     )
     => t_conName/nil
```

Description

Generates a unique name for a new constraint.

Arguments

d_cvId	Database ID of the cellview.
d_catId	Database ID of the category.

Return Values

t_conName	New constraint name.
nil	Cellview ID is invalid, or category ID is invalid.

cmxfConSetActive

```
cmxfConSetActive(
     d_conId
     g_bool
     )
     => t/nil
```

Description

Controls the active state of the specified constraint. If *g_bool* is `t`, the inactive constraint is set active. If *g_bool* is `nil`, the constraint is set as inactive. The constraint will not be activated if any constraint members are inactive. The inactive members of the constraint need to be removed or made active before the constraint can be made active. If a constraint successfully moves from an inactive state to an active state and is eligible for verification, it is automatically reverified.

Arguments

d_conId	Database ID of the constraint.
g_bool	Indicates whether to set the constraint active or inactive.

Return Values

`t`	The active status is changed.
`nil`	Constraint ID is invalid, cellview is read-only, or a value other than `t` of `nil` was specified for *g_bool*.

cmxfConSetAttrVal

```
cmxfConSetAttrVal(
     d_conId
     t_attrName
     g_attrVal
     )
     => t/nil
```

Description

Changes the value of a specific constraint attribute. The value of only one attribute can be changed with each call. If successful, the constraint will be reverified if it is eligible for verification.

Arguments

d_conId	Database ID of the constraint.
t_attrName	Name of constraint attribute.
g_attrVal	New attribute value.

Return Values

t	The constraint attribute is changed.
nil	Constraint ID is invalid, cellview ID is invalid, attribute name is invalid, attribute value is invalid, constraint is read-only, or cellview is read-only.

cmxfConSetWeight

```
cmxfConSetWeight(
     d_conId
     x_weight
     )
     => t/nil
```

Description

Sets the weight of the specified constraint.

Arguments

d_conId	Database ID of the constraint.
x_weight	New weight of the constraint. Valid values: 1 through 255

Return Values

t	The new weight is assigned.
nil	Constraint ID is invalid, cellview is read-only, or constraint is read-only.

cmxfConVerify

```
cmxfConVerify(
     d_conId
     )
     => t/nil
```

Description

Verifies that the specified constraint is eligible for verification and marks it as satisfied or unsatisfied. If the cellview to which the constraint belongs is read-only, no reverification is done and only the current constraint satisfied status is returned, `t` if satisfied, `nil` if not. In such cases, the result is the same as the one returned by `cmxfConIsSatisfied()`.

Arguments

d_conId	Database ID of the constraint.

Return Values

`t`	Constraint is eligible for verification is marked as satisfied.
`nil`	Constraint ID is invalid, or constraint is unsatisfied.

cmxfGenCon

```
cmxfGenCon(
     d_genId
     )
     => d_conId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenCon()`, generates the next constraint ID.

Stop the generator after all constraints IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_conId	Database ID of the constraint.
`nil`	No more constraints to generate, or generator ID is invalid.

cmxfGenConToFig

```
cmxfGenConToFig(
     d_genId
     )
     => d_figId/nil
```

Description

Creates a new generator for generating figure IDs for the specified constraint.

Arguments

d_genId	Database ID of the generator.

Return Values

d_figId	Database ID of the figure.
nil	No figures to be generated, or constraint ID is invalid.

cmxfGenConToMemName

```
cmxfGenConToMemName(
     d_genId
     )
     => t_memType/nil
```

Description

Given the generator ID obtained from `cmxfStartGenConToMemName()`, generates the next member name.

Stop the generator after all member names have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

t_memType	Name of the generated member.
`nil`	No more member names to generate, or generator ID is invalid.

cmxfGenConToNet

```
cmxfGenConToNet(
     d_genId
     )
     => d_netId/nil
```

Description

Given the generator ID obtained from `cmxfStartGenConToNet()`, generates the next net ID.

Stop the generator after all net IDs have been generated by using `cmxfStopGen()`.

Arguments

d_genId	Database ID of the generator.

Return Values

d_netId	Database ID of the generated net.
`nil`	No more nets to generate, or generator ID is invalid.

cmxfStartGenCon

```
cmxfStartGenCon(
     d_cvId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating all constraint IDs for the specified cellview.

Arguments

d_cvId	Database ID of the cellview.

Return Values

d_genId	Database ID of the new generator.
nil	No constraints to be generated, or cellview ID is invalid.

cmxfStartGenConToFig

```
cmxfStartGenConToFig(
     d_conId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating figure IDs for the specified constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

d_genId	Database ID of the new generator.
nil	No figures to be generated, or constraint ID is invalid.

cmxfStartGenConToMemName

```
cmxfStartGenConToMemName(
     d_conId
     t_memType
     )
     => d_genId/nil
```

Description

Creates a new generator for generating member names for the specified constraint.

Arguments

d_conId	Database ID of the constraint.
t_memType	Name of member type. Valid Values: `net`, `net-class`, `instance`, `shape`, `pin`, `cluster`

Return Values

d_genId	Database ID of the new generator.
`nil`	No members of the specified type, member type is invalid, or constraint ID is invalid.

cmxfStartGenConToNet

```
cmxfStartGenConToNet(
     d_conId
     )
     => d_genId/nil
```

Description

Creates a new generator for generating net IDs for the specified constraint.

Arguments

d_conId	Database ID of the constraint.

Return Values

d_genId	Database ID of the new generator.
nil	No nets to be generated, or constraint ID is invalid.

Report and GUI Functions

The following section provides Constraint Manager constraint SKILL functions, which are used to create reports, start the Constraint Manager GUI, and dump or load constraint files..

cmxfAsc

```
cmxfAsc(
     t_loadDumpMode
     t_libFile
     t_fileNm
     t_consMode
     [d_rep]
     [t_runDir]
     [t_saveMode]
     )
     => t/Error
```

Description

Loads or dumps constraints to or from a specified ASCII file.

Arguments

t_loadDumpMode	Specifies whether to load constraints from a file or dump the constraints to a file. Valid value: `load` or `dump`
t_libFile	ASCII file that contains list of library cellview and view names whose constraints will be loaded or dumped. Valid Values: library_*name*/cell_*name*/view_*name*
t_fileNm	The name of the ASCII constraint file.
l_conMode	Specifies which type of constraint (net-based or geometric) will be loaded or dumped. Valid Values: (`t t`) Net based and geometric constraints will be dumped or loaded (`t nil`) Net-based constraints and their support entities will be loaded or dumped

	(`nil t`) Geometric constraints and their support entities will be loaded or dumped
t_rep	Target cellview. If `nil` the current cellview will be used. Default: `nil`
t_rundir	The directory in which the constraints are dumped to or loaded from. Default: `cdsCon`
t_saveMode	Specifies where the constraints are being loaded from or dumped to. Valid Values: `Current` the current cellview. `All Views` all view representations of the current cellview. `Lib File` exports cellviews specified in the ASCII library file you create. Default: `Current`

Return Values

`t`	Constraint file is successfully processed.
`Error`	An error is detected.

cmxfGuiReportGen

```
cmxfGuiReportGen(
     p_outport
     d_cvId
     g_completeReport
     )
     => t/nil
```

Description

Creates a constraint report.

Arguments

p_outport

cvId	Database ID of the cellview.

Return Values

t	Successfully creates a report.
nil	Cellview ID is invalid.

cmxfGuiStartUp

```
cmxfGuiStartUp(
     )
     => t/nil
```

Description

Entry function to be called when constraint manager is started.

Arguments

None.

Return Values

`t`	Opens the Constraint Manager GUI.
`nil`	Constraint Manager is not available.