cadence®

Virtuoso Design Environment SKILL Reference

Product Version ICADVM20.1
October 2020

© 2020 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

Preface . . . 55
Scope . . . 56
Licensing Requirements . . . 56
Related Documentation . . . 56
What's New and KPNS . . . 56
Installation, Environment, and Infrastructure . . . 56
Virtuoso Tools . . . 57
SKILL Documents . . . 57
Additional Learning Resources . . . 57
Video Library . . . 57
Virtuoso Videos Book . . . 58
Rapid Adoption Kits . . . 58
Help and Support Facilities . . . 58
Customer Support . . . 59
Feedback about Documentation . . . 59
Understanding Cadence SKILL . . . 60
Using SKILL Code Examples . . . 60
Sample SKILL Code . . . 60
Accessing API Help . . . 61
Typographic and Syntax Conventions . . . 62
Identifiers Used to Denote Data Types . . . 63

1
Graphics Editor Functions . . . 65
Design Functions . . . 66
geChangeCellView . . . 67
geChangeEditMode . . . 68
geDiscardEdits . . . 69
geNewWindow . . . 70
geReturn . . . 71
geReturnToLevel . . . 72

© 2020 All Rights Reserved.

geSave . . . 73
geSaveAs . . . 74
geSaveHier . . . 76
Automatic Menu Enabling SKILL Functions . . . 77
geGetDisabledMenus . . . 79
geGetEnablePredicate . . . 81
geGetWindowStatus . . . 82
geSetEnablePredicate . . . 83
geSetWindowStatus . . . 85
geUpdateWindowMenus . . . 86
Edit/Display Functions . . . 87
geClearIgnoreProp . . . 87
geComputeQueryBbox . . . 88
geCopySelSet . . . 89
geDeleteSelSet . . . 90
geMenuForObjectsUnderPoint . . . 91
geSelectNextInSelSet . . . 92
geSelectPreviousInSelSet . . . 93
geSelectToggleObject . . . 94
geDispWinEnvProps . . . 95
geEditSelSet . . . 96
geEditSingleObject . . . 97
geEditToWindowPoint . . . 98
geEquivWindow . . . 100
geEquivWindowSet . . . 101
geFindPathToCellViewInst . . . 102
geClearNetNameDisplayFilter . . . 103
geGetNetNameDisplayFilter . . . 104
geIsNetNameDisplayActiveOnWindow . . . 105
geIsNetNameDisplayEnabled . . . 106
geGetSortedSelectByLine . . . 107
geGetCellViewWindow . . . 108
geGetEditCellView . . . 109
geGetEditCellViewWindow . . . 110
geGetEditRep . . . 111
geGetHierMemInst . . . 112

© 2020 All Rights Reserved.

geGetInstHier . . . 113
geGetInstHierPath . . . 114
geGetObjectSelectedSet . . . 115
geGetTopLevelCellView . . . 117
geGetTopCellViewWindow . . . 118
geGetWindowBox . . . 119
geGetWindowCellView . . . 120
geGetWindowRep . . . 121
geIsGraphicalWindow . . . 122
geSaveNetNameDisplayFilter . . . 123
geSetNetNameDisplayFilter . . . 124
geSetTCProbeDisplayStyle . . . 125
geTransparencyOptions . . . 126
geWindowToEditPoint . . . 127
geEnableNetNameDisplay . . . 129
geNetNameDisplayOptionForm . . . 130
Refresh Functions . . . 131
geRefresh . . . 131
geRefreshCellView . . . 132
geRefreshCellViews . . . 133
geRefreshWindow . . . 134
geRefreshWorldView . . . 135
View Level Functions . . . 136
geDeleteAllAreaViewLevel . . . 136
geDeleteAllInstViewLevel . . . 137
geDeleteAreaViewLevel . . . 138
geDeleteInstViewLevel . . . 139
geGetAllAreaViewLevel . . . 140
geGetAllInstViewLevel . . . 141
geGetAreaViewLevel . . . 142
geGetInstViewLevel . . . 143
geSetAreaViewLevel . . . 144
geSetInstViewLevel . . . 146
Viewport Functions . . . 148
geRestoreWinView . . . 148
geSaveWinView . . . 149

© 2020 All Rights Reserved.

Procedural Level Functions 150
geEditInPlace 150
gePush 152
geSwitch 154
geSwitchInContext 156
Window Scrolling Functions 158
geCenter 158
geGetScrollPercent 159
geScroll 160
geSetScrollPercent 161
geShift 162
geZoomToGrid 163
Selection Functions 164
Selected Set 164
Point Selection 164
Single Mode Point Selection 165
Area Selection 165
User Interface 166
SKILL-Level Access 168
geAdvWireSelection 169
geAddSelectBox 170
geAddSelectPoint 171
geAdjustPoint 172
geCyclePotentialSelectNext 173
geCycleSelectNext 174
geDeselectAll 175
geDeselectAllFig 176
geDeselectAllObject 177
geDeselectArea 178
geDeselectFig 179
geDeselectFigs 180
geDeselectFigNoFilter 181
geDeselectFigPoint 182
geDeselectObject 183
geDeselectObjectNoFilter 184
geDeselectObjectPoint 185

© 2020 All Rights Reserved.

geDeselectPoint 186
geDeselectProtected 187
geGetPotentialSelection 188
geGetSelSet 189
geGetSelectedSet 190
geGetSelSetCount 191
geGetSelectedInstancesCount 192
geGetSelProtectedObjects 193
geGetSelSetFigPoint 194
geGetWindowSelProtectionHighlight 195
geGetWindowSelProtectionSelectability 196
geHiDragFig 197
geHiDisplaySelectionForm 200
geHighlightProtected 201
geHiSetSelProtectionEF 202
geHiUnsetSelProtectionEF 203
geIsFigAllSelected 204
geIsFigPartiallySelected 205
geIsFigSelected 206
geIsObjectAllSelected 207
geIsObjectPartiallySelected 208
geIsObjectSelected 209
geIsSelProtection 210
gePointQuery 211
gePointToFig 212
geSelectAll 213
geSelectAllFig 214
geSelectArea 215
geSelectByLine 217
geSelectBy2PointsLine 218
geSelectFig 219
geSelectFigs 220
geSelectFigNoFilter 221
geSelectFigPoint 222
geSelectObject 223
geSelectObjectNoFilter 224

© 2020 All Rights Reserved.

geSelectPoint . . . 225
geSelectProtected . . . 227
geSelectViaPile . . . 228
geSetEnterFunctionSelectionMode . . . 229
geSelObjectsPartiallySelected . . . 230
geSetSelProtection . . . 231
geSetSelProtectionAllObjects . . . 232
geSetSelProtectionHighlightOptionForm . . . 233
geSetSelProtectionHighlightOptions . . . 234
geSingleSelectBox . . . 235
geSingleSelectPoint . . . 236
geSubSelectBox . . . 237
geSubSelectPoint . . . 238
geToggleAreaSelectOption . . . 239
geToggleDisplayResolution . . . 240
geToggleFilterSize . . . 241
geTogglePartialSelect . . . 242
geToggleWindowSelProtectionHighlight . . . 243
geToggleWindowSelProtectionSelectability . . . 244
geUnhighlightProtected . . . 245
geUnsetAllSelProtection . . . 246
geUnsetSelProtection . . . 247
DSA Function . . . 248
geDSAFreeze . . . 248
Stored Selection . . . 249
geDeleteStoredSelectionSet . . . 249
geDeselectStoredSelectionSet . . . 250
geGetStoredSelectionSets . . . 251
geHiStoreSelectionForm . . . 252
geIsAutoStoreSelectionOn . . . 253
geSelectNextAutoStoreSet . . . 254
geSelectPreviousAutoStoreSet . . . 255
geSelectStoredSelectionSet . . . 256
geSetAutoStoreSelection . . . 257
geStoreSelectionSet . . . 258
Highlight Functions . . . 259

© 2020 All Rights Reserved.

geAddHilightArc 260
geAddHilightBlockage 261
geAddHilightCircle 262
geAddHilightCurvedPath 263
geAddHilightCurvedPolygon 264
geAddHilightDistance 265
geAddHilightDonut 266
geAddHilightDot 267
geAddHilightEllipse 268
geAddHilightFig 269
geAddHilightLabel 271
geAddHilightLine 273
geAddHilightPath 274
geAddHilightPathSeg 275
geAddHilightPolygon 277
geAddHilightRow 278
geAddHilightRuler 279
geAddHilightSlicedCircle 280
geAddHilightSlicedDonut 281
geAddHilightTrackPattern 282
geAddHilightRectangle 283
geChangeHilightPacketName 284
geCreateHilightGroup 285
geCreateHilightSet 286
geCreateWindowHilightSet 287
geDeleteAllHilightSet 288
geDeleteAllWindowHilightSet 289
geDeleteHilightGroup 290
geDeleteHilightObject 291
geDeleteHilightSet 292
geDeleteHilightSetInWindows 293
geDrawHilightSet 294
geGetCurrentHilightSet 295
geGetCurrentWindowHilightSet 296
geGetFigHilightObjectId 297
geGetHilightObjectFigId 298

© 2020 All Rights Reserved.

geInsertHilightObjectsInGroup . . . 299
geIsValidHilightGroup . . . 300
geIsValidHilightId . . . 301
geIsValidHilightObject . . . 302
geIsValidHilightSet . . . 303
gePopHilightStack . . . 304
gePopWindowHilightStack . . . 305
gePushHilightStack . . . 306
geResetHilightSetAlphaBlendingParameters . . . 307
geResetHilightSetHaloParameters . . . 308
geSetHilightSetAlphaBlendingParameters . . . 309
geSetHilightSetHaloParameters . . . 311
geShareWindowHilightSet . . . 313
Background Cellview Functions . . . 314
geBackgroundCellView . . . 314
geDisableBackgroundCellView . . . 315
geEnableBackgroundCellView . . . 316
geGetBackgroundCellView . . . 317
geResetBackgroundCellView . . . 318
geSetBackgroundCellView . . . 319
geZoomToBackgroundAndEditCV . . . 320
geZoomToBackgroundCV . . . 321
Marker Functions . . . 322
geCreateMarker . . . 322
geCreateMarkerByBBox . . . 324
geCreateMarkerByPoints . . . 326
geDeleteAllMarker . . . 327
geGetMarkerDisplayText . . . 329
geGetMarkerOwner . . . 330
geGetMarkerReason . . . 331
geGetMarkerType . . . 332
geHasMarker . . . 333
geHiCommonFindMarker . . . 334
geHiDeleteAllMarker . . . 335
geHiDeleteMarker . . . 336
geHiExplainMarker . . . 337

© 2020 All Rights Reserved.

geHiFindMarker . . . 338
geIsMarkerShape . . . 339
Probing Functions . . . 340
Probe Types . . . 343
SKILL Interface . . . 347
geAddInstProbe . . . 349
geEnterAddInstProbe . . . 351
geGetAdjustedPath . . . 353
geAddNetProbe . . . 355
geEnterAddNetProbe . . . 357
geAddPathProbe . . . 359
geAddTermProbe . . . 361
geEnterAddTermProbe . . . 363
geDeleteAllProbe . . . 365
geDeleteInstProbe . . . 366
geEnterDeleteInstProbe . . . 367
geDeleteNetProbe . . . 368
geEnterDeleteNetProbe . . . 369
geDeletePathProbe . . . 370
geDeleteProbe . . . 371
geDeleteTermProbe . . . 372
geEnterDeleteTermProbe . . . 373
geExplainProbe . . . 374
geGetAllProbe . . . 375
geSetMaxCreatedProbe . . . 376
geGetCrossProbeValue . . . 377
geGetProbeNetStopLevel . . . 378
geGetProbeUserLPP . . . 379
geAddNetProbeFromTo . . . 380
geSetLimitedProbe . . . 382
geSetProbeDirection . . . 383
geClearProbeNetFilter . . . 384
geGetProbeNetFilter . . . 385
geSaveProbeNetFilter . . . 386
geSetProbeNetFilter . . . 387
Probe File Format . . . 388

© 2020 All Rights Reserved.

geLoadNextProbe 390
geLoadProbe 392
geMakeProbeWithColor 394
geMakeProbeWithoutColor 397
geMakeProbeWithLabel 399
geMakeProbeWithOnColor 403
gePrintAllProbe 406
geProbeP 407
geRegCrossProbeTrigger 408
geSaveProbe 409
geSetCrossProbeValue 410
geSetProbeNetStopLevel 411
geSetProbeUserLPP 412
geiLoadProbe 413
geiSaveProbe 414
Icon Creation and Loading Functions 415
geCellViewToDlist 415
geLoadIcon 416
geSaveIcon 418
geUnRegCrossProbeTrigger 419
Utility Functions 420
geAskOverwriteMode 420
geClosePolygon 421
geCompressPointArray 422
geCoordp 423
geCreatePacket 424
geGetInstTransform 426
geGetLockInfo 427
geInstIdToFileSpec 428
geIsPathReentrant 430
geIsPolygonReentrant 432
geOpen 433
geQuerySelSet 435
geRegSaveAllTrigger 436
geResetCellView 437
geTransformUserBBox 438

© 2020 All Rights Reserved.

geTransformUserPoint 439
geValidateWindow 440
geWriteCellView 441
Microwave Object Functions 442
geCreateBend 442
geCreateTaper 444
geCreateTrl 446
Mouse Functions 447
mouseAddPt 448
mouseAddSelectPt 449
mouseFinishPt 450
mousePopUp 451
mouseSingleSelectPt 452
mouseSubSelectPt 453
mouseUndoPt 454

2
Database Access Functions 457

Data Stored as Objects 459
Database Types 459
dbGetDatabaseType 459
Object Identifier (dbObject) 460
Object Types and Classes 460
Relationships Among Objects 462
Access Function Argument Types 462
dbIsId 467
dbValidP 468
dbobjectp 469
Inherited Connections Functions 470
dbCreateSigNetExpr 471
dbCreateTermNetExpr 472
dbCVHasInheritedSig 473
dbCVHasInheritedTerm 474
dbCVHasNetSetProp 475
dbDeleteSigNetExpr 476

© 2020 All Rights Reserved.

dbDeleteTermNetExpr 477
dbFix 478
dbGetCellViewInheritedSig 479
dbGetCellViewInheritedTerm 480
dbGetSigNetExpr 481
dbGetTermNetExpr 482
dbIsValidSigNetExpr 483
dbIsValidTermNetExpr 484
dbIsSigInherited 485
dbIsTermInherited 486
dbParseSigNetExpr 487
dbParseTermNetExpr 488
dbReplaceSigNetExpr 489
dbReplaceTermNetExpr 490
Net and Terminal Name Syntax 491
Instance Name Syntax 493
Undo and Redo Functions 494
dbSetUndoHandle 494
dbUndoToHandle 496
dbGetUndoHandles 497
dbGetRedoHandles 498
dbRedoToHandle 499
dbEnableUndo 500
Pcell Freeze Functions 501
dbFreeze 501
Fluid Shape Functions 503
dbGetFluidShapeByName 504
dbGetFluidShapeName 505
dbGetFluidShapes 506
dbIsFluidPcell 507
dbIsFluidShape 508
dbSetFluidShape 509
Pcell Abutment Support Functions 510
dbAbutIsInteractive 511
dbAbutIsReAbutGroup 512
dbSetFigAbutName 513

© 2020 All Rights Reserved.

dbSetFigAbutRefName 514
dbGetFigAbutName 515
dbGetFigAbutRefName 516
dbHasFigAbutName 517
dbHasFigAbutRefName 518
dbFindAbutFig 519
dbUnabutGroup 520
Shape Creation and Retrieval Functions 521
dbCopyShape 521
dbCreateLine 522
dbCreatePath 523
dbCreatePolygon 525
dbCreateRect 526
dbCreateEllipse 527
dbCreateArc 528
dbCreateDonut 530
dbCreateDot 531
dbCreateLabel 532
dbCreateTextDisplay 534
dbHasShape 539
dbMoveShape 540
dbSetTextDisplayNameValueVisible 541
Shape Conversion Functions 542
dbConvertArcToLine 543
dbConvertDonutToPolygon 544
dbConvertEllipseToPolygon 545
dbConvertPathToPolygon 546
dbConvertPathSegToPolygon 547
dbConvertLineToPath 548
dbConvertRectToPolygon 549
dbCompressPointArray 550
Mosaic Creation and Retrieval Functions 551
dbCreateParamSimpleMosaic 552
dbCreateParamSimpleMosaicByMasterName 554
dbCreateSimpleMosaic 556
dbFindMosaicByName 558

© 2020 All Rights Reserved.

Property Creation and Modification Functions 559
dbCreateProp 559
dbCreateRangeProp 560
dbCreateEnumProp 561
dbCreateHierProp 562
dbFindProp 563
dbGetPropByName 564
dbReplaceProp 565
dbReplaceRangeProp 566
dbReplaceEnumProp 568
dbReplaceHierProp 569
dbCopyProp 570
dbCopyPropList 571
dbResetPropList 572
dbDeletePropByName 574
dbCopySingleProp 575
dbMergeSingleProp 576
dbReplacePropList 577
dbSearchPropByName 578
Object Deletion Functions 579
dbDeleteObject 579
Group Creation and Modification Functions 581
dbAddObjectToGroup 581
dbChangeGroupType 582
dbCreateGroup 583
dbDeleteGroupByName 585
dbDeleteObjectFromGroup 586
dbGetGroupByName 587
dbGetGroupsWithDef 588
dbMoveGroupMember 589
dbDeleteGroupMemberFromGroup 590
Shape Transformation Functions 591
dbTransformCellView 591
dbTransformPoint 592
dbTransformPointList 593
dbTransformBBox 595

© 2020 All Rights Reserved.

dbConcatTransform 596
dbCanonicalizeAnyAngleTransform 597
dbConvertAnyAngleTransformFromDbToMY 599
Bounding Box Functions 601
lowerLeft 602
upperRight 603
leftEdge 604
rightEdge 605
bottomEdge 606
topEdge 607
centerBox 608
isLocation 609
isBBox 610
Property Bag Functions 611
dbOpenBag 611
dbReOpenBag 613
dbGetBag 614
dbGetBagOwner 615
dbCloseBag 616
dbPurgeBag 617
dbRefreshBag 618
dbMakeBagCurrent 619
dbSaveBag 620
dbIsBagReadable 621
dbIsBagWritable 622
dbGetBagTimeStamp 624
dbFindOpenBag 625
dbGetBagDdId 626
dbGetOpenBags 627
dbIsBagId 628
Layer Boolean Functions 629
dbLayerAnd 630
dbLayerAndNot 631
dbLayerOr 632
dbLayerXor 633
dbLayerSize 634

© 2020 All Rights Reserved.

dbLayerTile 636
dbLayerStraddle 637
dbLayerNoHoles 638
dbLayerHoles 639
dbLayerInside 640
dbLayerOutside 641
dbLayerEnclose 642
Point Array Functions 643
dbPointArrayAnd 644
dbPointArrayAndNot 645
dbPointArrayOr 646
dbPointArraySize 647
dbPointArrayXor 648
Soft Connect Terminal Functions 649
dbSetTermSoftConnect 650
dbGetSoftConnectTermConnectToLayer 651
dbGetSoftConnectTermPinlessLayer 652
dbIsTermSoftConnect 653
dbSetSoftConnectTermConnectToLayer 654
dbSetSoftConnectTermPinlessLayer 655
dbSoftConnectTermHasConnectToLayer 656
dbSoftConnectTermHasPinlessLayer 657
Information Retrieval Functions 658
dbDebugStats2 658
dbGetBuildInfo 659
dbGetFileDatabaseType 660
dbGetVersion 661
dbSetNIMsgMode 662
Blockage Functions 663
dbCreateAreaBlockage 664
dbCreateAreaHalo 665
dbCreateLayerBlockage 666
dbCreateLayerHalo 667
Boundary Functions 669
dbCreateAreaBoundary 670
dbCreateClusterBoundary 671

© 2020 All Rights Reserved.

dbCreateCoverObstruction 672
dbCreatePRBoundary 673
dbCreateSnapBoundary 674
dbFindAreaBoundaryByName 675
dbFindClusterBoundaryByName 676
dbSetBoundaryEdge 677
Bus Definition and Bit Order Functions 678
dbCreateBusNetDef 679
dbCreateBusTermDef 680
dbCreateVectorInstDef 681
dbFindBusNetDefByName 682
dbFindBusTermDefByName 683
dbFindVectorInstDefByName 684
Cluster Functions 685
dbCreateCluster 686
dbFindClusterByName 687
Counter Functions 688
dbDisableCellViewCounter 689
dbEnableCellViewCounter 691
dbGetCellViewCounter 692
dbGetFileTimeStamp 693
dbIsConnCurrent 694
dbSetConnCurrent 695
dbSetInstHeadersCounter 696
Interconnect Functions 697
dbCreateGuide 698
dbCreatePathSeg 699
dbCreateRoute 707
dbCreateVia 709
dbCreateCustomViaVariant 712
dbCreateStdViaVariant 713
dbFindViaVariantByName 716
dbGetViaLayer 717
dbIsCellViewStdViaMaster 718
dbSetPathSegPoints 719
dbSetViaParamDefault 720

© 2020 All Rights Reserved.

dbSetViaShapePurpose 721
dbGetRouteAuthor 722
dbSetRouteAuthor 723
xdvReplaceInstWithVias 724
Trim Layer Functions (ICADVM20.1 Only) 725
Trim Functions 726
dbGetAllTrimFillShapes 726
dbGetShapeTrimFillType 727
Patch Functions 728
dbGetPatchType 729
dbSetBeginPatch 730
dbSetEndPatch 731
dbSetFullPatch 732
dbSetPatchType 733
Marker Functions 734
dbCreateMarker 735
dbAddObjectsToMarker 737
dbRemoveObjectsFromMarker 738
dbGetMarkerName 739
dbSetMarkerName 740
Ruler Functions 741
dbCreateRuler 741
Query Functions 742
dbBoundaryQuery 742
dbFigGroupQuery 744
dbGuideQuery 746
dbInstQuery 748
dbLayerBlockageQuery 750
dbMarkerQuery 752
dbPlacementBlockageQuery 754
dbRowQuery 756
dbShapeQuery 758
dbSteinerQuery 761
dbViaQuery 763
Row Functions 765
dbCreateRow 766

© 2020 All Rights Reserved.

dbFindRowByName . . . 767
Row Header Functions . . . 768
dbFindRowHeader . . . 769
Steiner Functions . . . 770
dbCreateSteiner . . . 771
dbCreateTapSteiner . . . 772
dbIsTapSteiner . . . 773
Supply and Ground Sensitivity Functions . . . 774
dbGetTermGroundSensitivity . . . 775
dbGetTermSupplySensitivity . . . 776
dbSetTermGroundSensitivity . . . 777
dbSetTermSupplySensitivity . . . 778
Terminal Ordering Functions . . . 779
dbCreateInstTermByPosition . . . 780
dbFindTermByPosition . . . 781
Terminal Antenna Functions . . . 782
dbDeleteAllTerminalAntennaRules . . . 783
dbDeleteTerminalAntennaRuleOnLayer . . . 784
dbDeleteTerminalAntennaRules . . . 786
dbGetTerminalAntennaRuleOnLayer . . . 788
dbGetTerminalAntennaRules . . . 789
dbSetTerminalAntennaRuleOnLayer . . . 791
dbSetTerminalAntennaRules . . . 793
Track Pattern Functions . . . 795
dbCreateTrackPattern . . . 796
GCell Pattern Function . . . 797
dbCreateGCellPattern . . . 798
Via Functions . . . 799
Via GenVia Functions . . . 800
dbCreateGenViaVariant . . . 801
dbIsCellViewGenViaMaster . . . 803
dbIsGenVia . . . 804
dbIsGenViaHeader . . . 805
dbIsGenViaVariant . . . 806
dbSetGenViaParamDefault . . . 807
Via Header Function . . . 808

© 2020 All Rights Reserved.

dbFindViaHeaderByName 809
Optical Data Functions 810
dbCellViewHasOpticalData 811
dbIsNetOptical 812
dbListTypes 813
. 814

3
Cellview Input/Output Database Access Functions 815
dbAllCellViews 818
dbCellViewHasVirtHier 819
dbCheckRecursion 820
dbClose 821
dbComputeBBox 822
dbComputeBBoxNoNLP 823
dbCopyCellView 824
dbCreateLib 826
dbDefineProc 827
dbFeaturePrintInfo 828
dbFindOpenCellView 829
dbFlattenInst 830
dbFlattenInst2 833
dbFullLibPath 837
dbFullPath 838
dbGetAnyInstSwitchMaster 839
dbGetCellViewDdId 840
dbGetLibDataCompressionLevel 841
dbGetOpenCellViews 842
dbHasAutoSavedFile 843
dbHasPanicFile 844
dbIsNonSKILLPcell 845
dbOpenAutoSavedCellView 846
dbOpenCellViewByType 848
dbOpenHier 852
dbOpenPanicCellView 853

© 2020 All Rights Reserved.

dbOpenParamCellView 855
dbProduceMemName 856
dbPurge 858
dbRefreshCellView 859
dbRefreshLibDataCompressionLevel 860
dbRegPostSaveAsTrigger 861
dbRegPostSaveTrigger 862
dbRegPurgeTrigger 863
dbRegSaveTrigger 864
dbReopen 865
dbRestoreAndOpenAutoSavedFile 867
dbRestoreAndOpenPanicFile 868
dbSave 869
dbSearchCDF 871
dbSetAutoSave 872
dbSetAutoSaveCallback 874
dbSetLibDataCompressionLevel 876
dbUndefineProc 877
dbUnregPostSaveAsTrigger 878
dbUnregPostSaveTrigger 879
dbUnregPurgeTrigger 880
dbUnregSaveTrigger 881
dbUnsetLibDataCompressionLevel 882
dbWriteSkill 883
dbWriteSkillWithLib 886

4
Instance Creation and Retrieval Database Access Functions

889

dbCreateInst 891
dbCreateParamInst 893
dbCreateExtParamInst 895
dbCreateExtParamInstByMasterName 897
dbCreateXformPCell 899
dbDumpPcDefinePcell 900

© 2020 All Rights Reserved.

dbCheckParamCell 901
dbEvalParamCell 902
dbFixParamCell 903
dbHasInstance 904
dbIsCellViewModified 905
dbReplaceParam 906
dbReplaceInstParamList 907
dbFindAnyInstByName 908
dbFindMemInstByName 909
dbGetHierPathTransform 911
dbGetInstByName 912
dbGetInstanceByName 913
dbGetInstNameNumInst 914
dbGetInstTransform 915
dbIsAnyInstBaseNameUsed 916
dbCreateInstByMasterName 917
dbCreateParamInstByMasterName 919
dbDumpPcell 921
dbDumpPcellWithLib 922
dbCreateViaShapeArray 923
dbSetInstHeaderMasterName 925
dbIsSameMaster 926
dbIsObjectBound 927
dbGetCellViewSymmetry 928
dbSetCellViewSymmetry 929
dbGetCellViewEEQMaster 930
dbSetCellViewEEQMaster 931
dbIsInstTransparent 932
dbSetInstTransparent 933

5
Connectivity Database Access Functions 935

Connectivity Creation and Modification Functions 936
dbCreateNet 938
dbCreateNamedSubNet 940

© 2020 All Rights Reserved.

dbMakeNet 942
dbMergeNet 943
dbCreateUniqueNamedNet 944
dbSetNetNameDescend 946
dbIsNetNameDescend 947
dbNetIsSig 948
dbDeleteAllNet 949
dbCreateTerm 950
dbCreateInstTerm 952
dbHasShortedTerminals 954
dbCreatePin 955
dbCreateConn 957
dbCreateConnByName 958
dbCreateConnByNewName 959
dbSubFigFromNet 960
dbAddFigToNet 961
dbGetNetVoltageRange 962
dbSetNetVoltageRange 963
dbUnsetNetVoltageRange 964
dbGetNetVoltageRangeSource 965
dbSetNetVoltageRangeSource 966
dbUnsetNetVoltageRangeSource 967
dbGetCellViewNetVoltageRange 968
dbSetCellViewNetVoltageRange 969
dbIsCellViewNetVoltageRangeSet 970
dbIsNetVoltageRangeSet 971
dbUnsetCellViewNetVoltageRange 972
dbGetNetPowerDomain 973
dbSetNetPowerDomain 974
dbGetPowerDomainClusters 975
dbRenameNet 977
dbIsCellViewPhysicalOnly 978
dbUnsetCellViewPhysicalOnly 979
Equivalent Connectivity Time Functions 980
dbCellViewHasEquivalentConnectivityTime 980
dbGetCellViewEquivalentConnectivityTime 981

© 2020 All Rights Reserved.

dbSetCellViewEquivalentConnectivityTime 982
dbUnsetCellViewEquivalentConnectivityTime 983
Connectivity Access Functions 984
dbDeleteEmptyNet 985
dbFindNetByName 986
dbFindTermByName 987
dbFindSigByName 988
dbFindSigByAlias 989
dbGetMemNetSig 990
dbGetMemNetSigName 991
dbGetNetTerms 992
dbGetSigNameMemNets 993
dbIsSigNameGlobal 994
dbGetMemName 995
dbExpandToMemNameExpr 996
dbGetMaxNumBit 997
dbGetNameNumBit 998
dbMergeSignal 999
dbMoveChildNet 1000
dbMovePinToNet 1001

6
Pin Database Access Functions 1003

Pin Connection Functions 1004
External, Strong, and Weak Pin Connections 1004
dbCreateStrongGroup 1007
dbExternallyConnectPins 1008
dbGetPinConnection 1009
dbGetStrongGroups 1011
dbHasExternalConnections 1012
dbHasMustConnections 1013
dbMergeStrongGroups 1014
dbMustConnectPins 1015
dbStronglyConnectPins 1016
dbTransformAccessDirection 1017

© 2020 All Rights Reserved.

dbWeaklyConnectPins 1018
dbAddFigsToPin 1019
dbAddFigToPin 1020
dbGetPinFigAccessDirection 1021
dbGetPinFigName 1022
dbGetPinFigPlacementStatus 1023
dbSetPinFigAccessDirection 1024
dbSetPinFigName 1025
dbSetPinFigPlacementStatus 1026
dbSetTermMustJoin 1027
dbSubFigFromPin 1028
dbSubFigsFromPin 1029
dbUnsetTermMustJoin 1030
dbSetTermMustConnectAllPins 1031
dbTermHasMustConnectAllPins 1032
Pin Group Guide Functions 1033
dbCreateAreaPinGroupGuide 1036
dbCreateEdgePinGroupGuide 1037
dbFindPinGroupGuideByName 1038
dbGetAreaBoundaryPinGroupGuide 1039
dbGetAreaPinGroupGuideAreas 1040
dbGetFigPinGroupGuide 1041
dbGetPRBoundaryPinGroupGuides 1042
dbGetPinGroupGuideEdgeBoundary 1043
dbGetPinGroupGuideEdgeName 1044
dbGetPinGroupGuideEdgeStartOffset 1045
dbGetPinGroupGuideEdgeStopOffset 1046
dbGetPinGroupGuides 1047
dbGetPinPinGroupGuides 1048
dbAddAreaToPinGroupGuide 1049
dbAddFigToPinGroupGuide 1050
dbAddPinToPinGroupGuide 1051
dbSubAreaFromPinGroupGuide 1052
dbSubFigFromPinGroupGuide 1053
dbSubPinFromPinGroupGuide 1054
dbRemovePinGroupGuideEdge 1055

© 2020 All Rights Reserved.

dbSetPinGroupGuideEdge 1056
dbSetPinGroupGuideEdgeStartOffset 1057
dbSetPinGroupGuideEdgeStopOffset 1058
dbGetAreaBoundaryValidRoutingLayers 1059
dbAreaBoundaryHasLayerPriorityOrder 1060
dbAreaBoundaryHasValidRoutingLayers 1061
dbGetAreaBoundaryLayerPriorityOrder 1062
dbGetPinGroupGuideKeepoutSpacing 1063
dbGetPinGroupGuideLayerPriorityOrder 1064
dbGetPinGroupGuideMinPinSpacing 1065
dbGetPinGroupGuideMinPinWidth 1066
dbGetPinGroupGuideValidRoutingLayers 1067
dbGetPinMinPinSpacing 1068
dbGetPinMinPinWidth 1069
dbPinGroupGuideHasKeepoutSpacing 1070
dbPinGroupGuideHasLayerPriorityOrder 1071
dbPinGroupGuideHasMinPinSpacing 1072
dbPinGroupGuideHasMinPinWidth 1073
dbPinGroupGuideHasValidRoutingLayers 1074
dbPinHasMinPinSpacing 1075
dbPinHasMinPinWidth 1076
dbSetAreaBoundaryLayerPriorityOrder 1077
dbSetAreaBoundaryValidRoutingLayers 1078
dbSetPinGroupGuideKeepoutSpacing 1079
dbSetPinGroupGuideLayerPriorityOrder 1080
dbSetPinGroupGuideMinPinSpacing 1081
dbSetPinGroupGuideMinPinWidth 1082
dbSetPinGroupGuideValidRoutingLayers 1083
dbSetPinMinPinSpacing 1084
dbSetPinMinPinWidth 1085
dbUnsetAreaBoundaryLayerPriorityOrder 1086
dbUnsetAreaBoundaryValidRoutingLayers 1087
dbUnsetPinGroupGuideKeepoutSpacing 1088
dbUnsetPinGroupGuideLayerPriorityOrder 1089
dbUnsetPinGroupGuideMinPinSpacing 1090
dbUnsetPinGroupGuideMinPinWidth 1091

© 2020 All Rights Reserved.

dbUnsetPinGroupGuideValidRoutingLayers 1092
dbUnsetPinMinPinSpacing 1093
dbUnsetPinMinPinWidth 1094

7
Placement Database Access Functions 1095

Background Definition Functions 1099
dbCreateBackgroundDef 1099
dbCreateBackgroundDefByAttr 1101
dbGetBackgroundDefAttr 1103
dbGetCellViewBackgroundDef 1104
Component Type Set Definition Functions 1105
dbCreateCompTypeSetDef 1105
dbCreateCompTypeSetDefByAttr 1107
dbGetCellViewCompTypeSetDef 1109
dbGetCompTypeSetDefAttr 1110
dbGetCompTypeSetDefCompFilters 1111
dbSetCompTypeSetDefCompFilters 1112
dbUnsetCompTypeSetDefCompFilters 1113
Rail Definition Functions 1114
dbCreateRailDef 1114
dbCreateRailDefByAttr 1118
dbGetCellViewRailDef 1120
dbGetRailDefAttr 1121
dbGetRailDefFigType 1122
dbSetRailDefFigType 1123
Placement Area Definition Functions 1124
dbAddPlaceAreaBackgroundDef 1124
dbAddPlaceAreaCompTypeSetDef 1125
dbAddPlaceAreaRailDef 1126
dbCreatePlaceArea 1128
dbGetCellViewPlaceArea 1129
dbGetPlaceAreaBackgroundDef 1130
dbGetPlaceAreaCompTypeSetDef 1131
dbGetPlaceAreaOrient 1132

© 2020 All Rights Reserved.

dbGetPlaceAreaRailDef 1133
dbGetPlaceAreaUtilization 1134
dbGetPlacementGrid 1135
dbRemovePlaceAreaBackgroundDef 1136
dbRemovePlaceAreaCompTypeSetDef 1137
dbRemovePlaceAreaRailDef 1139
dbSetPlaceAreaOrient 1141
dbSetPlaceAreaUtilization 1142
dbSetPlacementGrid 1143
Row-based Functions 1144
dbAddRowBackgroundDef 1144
dbAddRowCompTypeSetDef 1145
dbGetRowBackgroundDef 1146
dbGetRowCompTypeSetDef 1147
dbAddRowRailDef 1148
dbGetRowRailDef 1149
dbRemoveRowCompTypeSetDef 1150
dbRemoveRowBackgroundDef 1151
dbRemoveRowRailDef 1152
Row Region Specification Functions (ICADVM20.1 Only) 1153
dbCreateRowRegionSpec 1153
dbFindRowRegionSpec 1158
dbGetCellViewRowRegionSpecs 1159
Row Region Functions (ICADVM20.1 Only) 1160
dbAttachRowRegionToPRBoundary 1160
dbCreateRowRegion 1162
dbDetachPRBoundaryFromRowRegion 1163
dbFindRowRegion 1164
dbFlattenRowRegion 1165
dbGetCellViewRowRegions 1166
dbGetRowRegionAttachedToPRBoundary 1167
dbGetRowRegionPlaceRows 1168
dbGetRowRegionPoints 1169
dbGetRowRegionRowRegionSpec 1170
dbGetRowRegionSpecRowRegions 1171
dbGetRowRegionUsesPartialRowSpec 1172

© 2020 All Rights Reserved.

dbIsRowRegion 1173
dbIsRowRegionAttachedToPRBoundary 1174
dbRowRegionPointsCutOut 1175
dbRebuildRowRegion 1176
dbRowRegionMatchesSpec 1177
dbSetRowRegionPoints 1179
dbSetRowRegionRowRegionSpec 1180
dbSetRowRegionUsesPartialRowSpec 1182
Miscellaneous Functions (ICADVM20.1 Only) 1183
dbGetPlaceRowVersion 1183
dbGetRowRegionVersion 1184
dbIsImplicit 1185
dbIsInRowRegion 1186

8
Topology Pattern Database Access Functions 1187
dbCreateTopology 1194
dbDeleteTopology 1195
dbGetCellViewTopologies 1196
dbGetNetTopologies 1197
dbGetTopologyName 1198
dbGetTopologyNet 1199
dbGetTopologyObjects 1200
dbGetTopologyPatternType 1201
dbSetTopologyName 1202
dbSetTopologyNet 1203
dbCreateTrunk 1204
dbDeleteTrunk 1205
dbGetTrunkAnchor 1206
dbGetTrunkDirection 1207
dbGetTrunkLongOffset1 1208
dbGetTrunkLongOffset2 1209
dbGetTrunkLongOffsetSource 1210
dbGetTrunkLongOffsetSource1 1211
dbGetTrunkLongOffsetSource2 1212

© 2020 All Rights Reserved.

dbGetTrunkName 1213
dbGetTrunkOrthoOffset 1214
dbGetTrunkSide 1215
dbGetTrunkTopology 1216
dbGetTrunkTwigs 1217
dbSetTrunkAnchor 1218
dbSetTrunkLongOffset1 1219
dbSetTrunkLongOffset2 1220
dbSetTrunkLongOffsetSource 1221
dbSetTrunkLongOffsetSource1 1222
dbSetTrunkLongOffsetSource2 1223
dbSetTrunkName 1224
dbSetTrunkDirection 1225
dbSetTrunkOrthoOffset 1226
dbSetTrunkSide 1227
dbCreateTwig 1228
dbDeleteTwig 1229
dbGetTwigName 1230
dbGetTwigObject 1231
dbGetTwigPinName 1232
dbGetShapeTwigs 1233
dbGetInstTermTwigs 1234
dbGetViaTwigs 1235
dbGetTwigTrunk 1236
dbSetTwigName 1237
dbSetTwigObject 1238
dbSetTwigPinName 1239
dbCreateTrunkConnect 1240
dbDeleteTrunkConnect 1241
dbGetTrunkConnectName 1242
dbGetTrunkConnectTopology 1243
dbGetTrunkConnectTrunks 1244
dbSetTrunkConnectName 1245
dbSetTrunkConnectTrunks 1246
dbCreateStrap 1247
dbDeleteStrap 1248

© 2020 All Rights Reserved.

dbGetStrapName 1249
dbGetStrapObjects 1250
dbGetStrapTopology 1251
dbGetStrapTwig 1252
dbGetInstTermStraps 1253
dbGetViaStraps 1254
dbGetShapeStraps 1255
dbAddStrapObject 1256
dbRemoveStrapObject 1257
dbSetStrapName 1258
dbSetStrapTwig 1259
dbIsTopologyPatternId 1260
dbNetHasTopology 1261

9
Generator Database Access Functions 1263

dbStartGenAnyInstToInstTerm 1264
dbStartGenInstHeader 1265
dbStartGenInstHeaderToAnyInst 1266
dbStartGenLPPHeaderToShape 1267
dbStartGenLPToShape 1268
dbStartGenMarker 1269
dbStartGenNet 1270
dbStartGenNetToInstTerm 1271
dbStartGenNetToRoute 1272
dbStartGenRoute 1273
dbStartGenShape 1274
dbStartGenTerm 1275
dbStartGenViaHeader 1276
dbStartGenViaHeaderToVia 1277
dbStopGen 1278

10
InPlace Cover Obstruction Database Access Functions . . 1279

dbCreateInPlaceCoverObstruction 1281

© 2020 All Rights Reserved.

dbDestroyInPlaceCoverObstruction 1282
dbGetInPlaceCoverObstructionAbstractFileName 1283
dbGetInPlaceCoverObstructionAbstractFilePath 1284
dbGetInPlaceCoverObstructionAbstractViewName 1285
dbGetInPlaceCoverObstructionBloat 1286
dbGetInPlaceCoverObstructionBlockageAttributeValue 1287
dbGetInPlaceCoverObstructionBlockageModel 1288
dbGetInPlaceCoverObstructionDoughnutHalo 1289
dbGetInPlaceCoverObstructionLayers 1290
dbGetInPlaceCoverObstructionMaxMask 1291
dbGetInPlaceCoverObstructionMergeDistance 1292
dbGetInPlaceCoverObstructionMinMask 1293
dbGetInPlaceCoverObstructionNeedsRemodeling 1294
dbGetInPlaceCoverObstructionPinCutOut 1295
dbGetInPlaceCoverObstructionPinRemodeling 1296
dbGetInPlaceCoverObstructionSpacingModel 1297
dbInPlaceCoverObstructionExists 1298
dbSetInPlaceCoverObstructionAbstractFileName 1299
dbSetInPlaceCoverObstructionAbstractFilePath 1300
dbSetInPlaceCoverObstructionAbstractViewName 1301
dbSetInPlaceCoverObstructionBloat 1302
dbSetInPlaceCoverObstructionBlockageAttributeValue 1303
dbSetInPlaceCoverObstructionBlockageModel 1304
dbSetInPlaceCoverObstructionDoughnutHalo 1305
dbSetInPlaceCoverObstructionMergeDistance 1306
dbSetInPlaceCoverObstructionNeedsRemodeling 1307
dbSetInPlaceCoverObstructionPinCutOut 1308
dbSetInPlaceCoverObstructionPinRemodeling 1309
dbSetInPlaceCoverObstructionSpacingModel 1310

11
Figure Database Access Functions 1313

FigGroup Creation and Modification Functions 1314
dbCreateFigGroup 1315
dbAddFigToFigGroup 1316

© 2020 All Rights Reserved.

dbRemoveFigFromFigGroup 1317
dbGetFigGroupByName 1318
Figure Creation and Retrieval Functions 1319
dbCopyFig 1320
dbMoveFig 1322
dbGetOverlaps 1323
dbGetTrueOverlaps 1326
dbGetNeighbor 1329
dbGetNeighborList 1332
dbGetMaxHierDepth 1334
dbGetPathSegStyle 1335
dbSetPathSegStyle 1336
dbGetPathBoundary 1337
dbProduceOverlap 1338
dbProduceOverlapInst 1340

12
Figure Set Database Access Functions 1343

FigSet Functions 1344
txCreateFigSet 1346
txGetFigSetData 1347
txAppendObjectToFigSet 1348
txClearFigSet 1349
txCloneFigSet 1350
txConcatFigSet 1351
txCopyFigSet 1352
txFindFigSet 1354
txDeleteFigSet 1355
txGetCellViewFigSets 1356
txGetFigSetBBox 1357
txIsFigSet 1360
txMakeFigSet 1361
txMoveFigSet 1363
txPrependObjectToFigSet 1365
txRemoveFigSetData 1366

© 2020 All Rights Reserved.

txRemoveObjectFromFigSet 1367
FigSetGroup Functions 1368
dbMakeFigSetGroup 1369
dbFindFigSetGroup 1370
dbFindFigSetGroupByName 1371
dbGetFigSetGroupName 1372
dbGetCellViewFigSetGroups 1373
dbIsFigSetGroup 1374
dbDeleteFigSetGroup 1375

13
Multi-Patterning Technology Database Access Functions (ICADVM20.1 Only) 1377
dbCellViewAreLayerShiftsValid 1380
dbCellViewClearLayerShifts 1381
dbCellViewHasLayerShifts 1382
dbCellViewUpdateLayerShifts 1383
dbCellViewInitForLayerShifting 1384
dbClearUnusedOccShapeColorData 1385
dbInstGetLayerShifts 1387
dbInstGetLayerShifts2 1388
dbInstHasLayerShifts 1390
dbInstSetLayerShifts 1391
dbInstSetLayerShifts2 1392
dbInstClearLayerShifts 1394
dbIsShapeColoringAllowed 1395
dbIsShapeColored 1396
dbGetShapeColor 1397
dbSetShapeColor 1398
dbIsShapeColorLocked 1399
dbSetShapeColorLocked 1400
dbIsBlockageColored 1401
dbIsBlockageColoringAllowed 1402
dbGetBlockageColor 1403
dbSetBlockageColor 1404

© 2020 All Rights Reserved.

dbIsViaColorStateLocked 1405
dbSetViaColorStateLocked 1406
dbIsViaColorStateLayerLocked 1407
dbSetViaColorStateLayerLocked 1408
dbGetViaLayerNumColorMasks 1409
dbGetViaLayer1Control 1410
dbGetViaLayer2Control 1411
dbGetViaCutLayerControl 1412
dbGetViaLayerControl 1413
dbSetViaLayer1Control 1415
dbSetViaLayer2Control 1417
dbSetViaCutLayerControl 1418
dbSetViaLayerControl 1419
dbSetViaColorInfo 1421
dbIsTrackPatternColored 1424
dbIsTrackPatternColorAlternating 1425
dbIsTrackPatternColoringAllowed 1426
dbGetTrackPatternFirstTrackColor 1427
dbSetTrackPatternFirstTrackColor 1428
dbSetTrackPatternColorAlternating 1429
dbCreateSameMaskGroup 1430
dbCreateDiffMaskGroup 1431
dbGetSameMaskDiffMaskGroups 1433
dbGetShapeSameMaskGroups 1434
dbColorShapeQuery 1435
dbGetIntegrationColorModel 1437
dbGetColorModel 1438
dbSetIntegrationColorModel 1439
dbSetColorModel 1440
dbRemoveLayerColorData 1441
dbColorShapeQuery2 1443
dbGetColoredOccShapes 1448
dbGetShapeEffectiveColor 1452
dbSetOccShapeColor 1456
dbSetOccShapeColorLocked 1459
dbGetHierColorOverride 1462

© 2020 All Rights Reserved.

dbGetShapeColorLockType 1465
dbSetShapeColorLockType 1468
dbGetViaLayerColorLockType 1471
dbSetViaLayerColorLockType 1474
dbCellViewHasUnknownColorLockChanges 1477
dbGetShapeEffectiveColorLockType 1479
dbCellViewResetAllColorLockTypes 1483
dbSetOccShapeColorLockType 1485

14
Width Spacing Pattern Database Access Functions (ICADVM20.1 Only) 1491

Width Spacing Pattern Functions 1492
dbCreateWidthSpacingPattern 1493
dbCreateWidthSpacingPatternWithColor 1500
dbFindWidthSpacingPattern 1505
dbGetWidthSpacingPatternAllowedRepeatMode 1506
dbGetWidthSpacingPatternDefaultRepeatMode 1507
dbGetWidthSpacingPatterns 1508
dbSetWidthSpacingPatternRepeatMode 1509
Width Spacing Pattern Group Functions 1510
dbCreateWidthSpacingPatternGroup 1511
dbGetWidthSpacingPatternGroups 1512
dbFindWidthSpacingPatternGroup 1513
Width Spacing Snap Pattern Def Functions 1514
dbCreateWidthSpacingSnapPatternDef 1515
dbFindWidthSpacingSnapPatternDefByName 1518
dbDeleteWidthSpacingSnapPatternDef 1519
dbGetWidthSpacingSnapPatternDefsByLP 1520
Related Snap Pattern Functions 1521
dbCreateRelatedSnapPatterns 1522
dbGetRelatedSnapPatterns 1524
dbFindRelatedSnapPatterns 1525
Pattern Region Functions 1526
dbGetPatternRegionActivePattern 1527

© 2020 All Rights Reserved.

dbSetPatternRegionActivePattern 1528
dbHasPatternRegionActivePattern 1529
dbGetPatternRegionAllowedPatterns 1530
dbSetPatternRegionAllowedPatterns 1531
dbGetPatternRegionAllowedPatternGroups 1532
dbSetPatternRegionAllowedPatternGroups 1533
dbGetPatternRegionCommittedPatterns 1534
dbGetPatternRegionCommittedPatternShiftColor 1535
dbGetPatternRegionCreateType 1536
dbGetPatternRegionShiftColor 1537
dbSetPatternRegionShiftColor 1538
dbHasPatternRegionInfo 1539
dbGetPatternRegionName 1540
dbSetPatternRegionName 1542
dbGetPatternRegionWidthSpacingSnapPatternDefName 1544
dbSetPatternRegionWidthSpacingSnapPatternDefName 1545
dbGetPatternRegionDefaultRepeatMode 1546
dbSetPatternRegionDefaultRepeatMode 1547
dbGetPatternRegionRepeatMode 1548
dbSetPatternRegionRepeatMode 1549
Active Wire Types Functions 1550
Per-Design Database Basis Functions 1551
dbGetCellViewActiveWireTypes 1551
dbSetCellViewActiveWireTypes 1552
dbClearAllActiveWireTypes 1553
Per-widthSpacingSnapPatternDef Basis Functions 1554
dbGetCellViewActiveWireTypesForWidthSpacingSnapPatternDef 1554
dbSetCellViewActiveWireTypesForWidthSpacingSnapPatternDef 1555
Per-Pattern Region Basis Functions 1556
dbGetPatternRegionActiveWireTypes 1556
dbSetPatternRegionActiveWireTypes 1557
dbSetPatternRegionNoActiveWireTypes 1558
dbHasPatternRegionNoActiveWireTypes 1559
Global Grid Offset Functions 1560
dbSetGlobalGridOffsetReferenceType 1561
dbGetGlobalGridOffsetReferenceType 1563

© 2020 All Rights Reserved.

dbGetGlobalGridDefaultRepeatMode 1564
dbSetGlobalGridDefaultRepeatMode 1565
dbGetGlobalGridRepeatMode 1566
dbSetGlobalGridRepeatMode 1567

15
Hierarchical Design Functions 1569

Hierarchy Database Functions 1570
hdbAddConfigToBag 1571
hdbBind 1573
hdbCloseConfigsInBag 1575
hdbCreateConfigBag 1576
hdbCreatePathVectorStack 1577
hdbExplainBinding 1578
hdbExplainSignature 1580
hdbExplainStop 1581
hdbGetPathStackStr 1582
hdbGetPathStr 1583
hdbGetPathVectorStackTop 1584
hdbIsAtStopPoint 1585
hdbIsConfig 1586
hdbIsOpenConfig 1587
hdbReOpenNewConfig 1588
hdbPopPathVect 1589
hdbPushPathVect 1590
hdbSaveACopy 1591
hdbSaveAs 1592
hdbSave 1593
hdbClose 1594
hdbOpen 1595
hdbCreatePathVector 1597
hdbDestroyPathVector 1598
hdbGetCellName 1599
hdbGetLibName 1600
hdbGetTopCellName 1601

© 2020 All Rights Reserved.

hdbGetTopLibName 1602
hdbGetTopViewName 1603
hdbGetViewName 1604
hdbPopCell 1605
hdbPushCell 1606
hdbSetObjBindRule 1607
hdbSetDefaultLibListString 1611
hdbSetDefaultViewListString 1612
hdbSetDefaultStopListString 1613
hdbSetTopCellViewName 1614
hdbBindBit 1615
hdbBitHasRules 1617
hdbHasIterInstBitRules 1618
hdbIsIterInst 1619
hdbIterInstHasBitRules 1621
hdbPushBitCell 1622
Occurrence Property Functions 1624
occpClose 1625
occpCreateProp 1626
occpGetBestProp 1627
occpGetBitInstProp 1628
occpGetCellProp 1630
occpGetDefaultProp 1631
occpGetInstProp 1632
occpGetPathProp 1634
occpOpen 1635
occpRmBitInstProp 1637
occpRmCellProp 1639
occpRmDefaultProp 1640
occpRmPathProp 1641
occpRmInstProp 1642
occpSave 1644
occpSetBitInstProp 1645
occpSetCellProp 1647
occpSetDefaultProp 1648
occpSetInstProp 1649

© 2020 All Rights Reserved.

occpSetPathProp 1651
Parent/Child Database Functions 1653
pcdbClose 1653
pcdbGetInstGen 1655
pcdbGetInstMasterGen 1656
pcdbGetIgnoredInstMasterGen 1657
pcdbInstMasterCell 1658
pcdbInstMasterCellFixed 1659
pcdbInstMasterLib 1660
pcdbInstMasterLibFixed 1661
pcdbInstMasterView 1662
pcdbInstMasterViewFixed 1663
pcdbInstName 1664
pcdbInstWidth 1665
pcdbNextInst 1666
pcdbNextInstMaster 1667
pcdbOpen 1668
pcdbViewHasPcdbInfo 1671
pcdbSave 1672

16
Virtuoso RF Functions (ICADVM20.1 Virtuoso MultiTech Framework Only) 1675
Enable VRF and MultiTech Framework Functions 1677
dbIsMultiTechEnabled 1677
dbIsVRFEnabled 1678
dbIsVRFInfraEnabled 1679
Curved Shape Functions 1680
dbCreateCurvedAreaBoundary 1680
dbCreateCurvedPath 1681
dbCreateCurvedPolygon 1682
dbCreateMultipleCurvedPolygons 1683
dbIsCurved 1684
dbGetCurvedAreaBoundaryPoints 1685
dbGetCurvedPolygonPoints 1686

© 2020 All Rights Reserved.

dbSetCurvedAreaBoundaryPoints 1687
dbSetCurvedPolygonPoints 1688
dbCreateRegionBoundary 1689
dbGetRegionBoundaryLayer 1691
dbGetRegionBoundaryLayerType 1692
dbGetRegionBoundaryName 1693
dbIsRegionBoundary 1694
VRF Package Infrastructure Functions 1695
pkxAddFuncDefPin 1696
pkxCreateCompDef 1697
pkxCreateCompDefPin 1698
pkxCreateFuncDef 1699
pkxCreateFuncDefPin 1700
pkxDeleteCompDef 1701
pkxFindCompDef 1702
pkxFindCompDefPin 1703
pkxFindFuncDef 1704
pkxFindFuncDefPin 1705
pkxGetCompDefPins 1707
pkxGetCompDefs 1708
pkxGetFuncDefPins 1709
pkxGetFuncDefs 1710
pkxGetProp 1711
pkxGetProps 1712
pkxSetProp 1713

17
Photonics Functions (ICADVM20.1 Photonics Only) 1715

Photonics Database SKILL functions 1716
dbCellViewHasPhotonicPinFig 1717
dbGetPhotonicPinFigAngle 1718
dbGetPhotonicPinFigRadius 1719
dbGetPhotonicPinFigWidth 1720
dbIsPhotonicsEnabled 1721
dbSetPhotonicPinFigAngle 1722

© 2020 All Rights Reserved.

dbSetPhotonicPinFigRadius 1723
dbSetPhotonicPinFigWidth 1724
dbUnsetPhotonicPinFigAngle 1725
dbUnsetPhotonicPinFigRadius 1726
dbUnsetPhotonicPinFigWidth 1727
Curvy Core Functions 1728
Function to Control the Curvy Core Feature 1731
ccIsEnabled 1731
Functions to Create Curvy Core Mathematical Objects 1732
ccCreateCurve 1732
ccCreateFacet 1734
ccCreateLineSegment 1736
ccCreatePolyCurve 1737
Functions to Create ccShapes 1739
ccCreateLine 1739
ccCreatePath 1741
ccCreatePie 1744
ccCreatePolygon 1747
ccCreateRect 1749
ccCreateSurface 1751
ccCreateSurfaceFromPath 1752
Functions for ccShape Geometries 1754
ccCopyFig 1754
ccConvertIntoCurvy 1756
ccGenFigs 1757
ccGenOffsetFig 1759
ccGetFromFig 1761
ccMoveFig 1762
ccRemoveFigs 1764
Functions for Curvy Core Mathematical Objects 1765
ccOffsetPolyCurve 1765
ccRenameFacet 1767
Functions to Query Object Attributes for Curvy Core Mathematical Objects 1768
ccCubicSplineConnector 1768
ccGetCurvePoint 1770
ccGetCurveDerivative 1772

© 2020 All Rights Reserved.

ccGetPolyCurveMinRadius 1774
ccIsLinearCurve 1775
Functions to Query Object Attributes for ccShapes 1776
ccGetAll 1776
ccGetPathBoundary 1777
ccGetPathCenterLine 1778
ccGetPathEdge 1779
ccFindSurfaceFacet 1780
Functions for Transformations 1781
ccLayerAnd 1781
ccLayerAndNot 1783
ccLayerOr 1785
ccMirrorFacet 1787
ccReversePolyCurve 1788
ccSmoothenFig 1789
ccSmoothenPolyCurve 1790
ccTransformFacet 1791
ccTransformPolyCurve 1792
Functions for Connectors 1793
ccBendConnector 1793
ccCreateConnectorWidthExpressionList 1796
ccCurveConnector 1800
ccSpiralConnector 1804
ccTromboneConnector 1807
ccWaypointConnector 1809
ccWideConnector 1813
Photonics Flow SKILL functions 1817
phoAddWaveguidePorts 1818
phoCompose 1821
phoComputeModeProps 1823
phoHiCompose 1825
phoCreatePort 1827
phoDrawCurveDeltaConnectorSymbol 1830
phoGetConnectorAnchors 1832
phoGenWaveguide 1834
phoHiDecompose 1836

© 2020 All Rights Reserved.

phoHiEditCompositeWG 1838
phoIsWaveguide 1839
phoTechDeleteModePropDataTable 1840
phoTechGetModePropDataTableParamValues 1841
phoTechHasModeProp 1842
phoTechHasModePropDataTable 1844
phoTechIsModePropLocalOnly 1845
phoTechLoadModePropDataTable 1847

18
Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions 1851

pgssDRC 1851
pvsApplyDRC 1854

19
Attribute Retrieval and Modification Database Access Functions 1857

dbGet Functions 1858
dbGetq, dbGet 1858
Using the ~> Operator 1861
dbSet Functions 1863
Attribute Types 1865
Object Types 1865
Generic Attributes 1867
Attributes of Cellviews 1880
Attributes of Instances, Instance Headers, and Mosaics 1885
Attributes of Instance Terminals, Pins, and Terminals 1890
Attributes of Bus Definitions, Nets, and Signals 1895
Attributes of Layer Purpose Pairs and Layer Headers 1902
Attributes of Groups, Group Members, and Properties 1904
Attributes of Objects and Figures 1908
Attributes of Shapes 1936
Technology File Attributes 1947

© 2020 All Rights Reserved.

20
Design Management Functions 1959

Library Organization 1960
- Version Control 1961
- Concurrence Control 1961

Attributes of Data Management Objects 1962
Triggering Function Calls 1964
- Setting the Trigger 1964
- Multiple Triggers for the Same Event 1964
- Setting Pre-triggers and Post-triggers 1965
- Trigger Names Allowed 1965
- Adding a PreObjAccess Trigger to ddGetObj 1966
- Adding a PreAutoCheckout Trigger to ddAutoCheckout 1967

Design Management Functions 1968
- ddAmUsingDM 1968
- ddAutoCheckin 1969
- ddAutoCheckout 1971
- ddAutoCtlGetVars 1973
- ddAutoCtlSetVars 1974
- ddCatRefresh 1976
- ddCheckin 1977
- ddCheckout 1978
- ddClearLibUnmanaged 1979
- ddCreateLib 1980
- ddDeleteObj 1982
- ddDeleteLocal 1984
- ddDumpLibList 1985
- ddGetCoManagedSet 1986
- ddGetCombineValue 1987
- ddGetDisplayValue 1988
- ddGetForcedLib 1989
- ddGetForcedLibEx 1990
- ddGetLibList 1991
- ddGetObj 1992
- ddGetObjAccess 1998

© 2020 All Rights Reserved.

ddGetObjAccessEx 1999
ddGetObjChildren 2001
ddGetObjChildrenEx 2002
ddGetObjDMSys 2003
ddGetObjFiles 2004
ddGetObjFilesEx 2005
ddGetObjGroup 2006
ddGetObjGroupEx 2007
ddGetObjLastModify 2008
ddGetObjLastModifyEx 2009
ddGetObjLib 2010
ddGetObjName 2011
ddGetObjOwner 2012
ddGetObjOwnerEx 2013
ddGetObjParent 2014
ddGetObjReadPath 2015
ddGetObjReadPathEx 2016
ddGetObjSize 2017
ddGetObjSizeEx 2018
ddGetObjType 2019
ddGetObjWritePath 2020
ddGetObjWritePathEx 2021
ddGetPathDMSys 2022
ddGetStartup 2023
ddGetTmpdirWarn 2025
ddGetTriggerList 2026
ddGetUpdatedLib 2027
ddIsId 2028
ddIsHiddenCell 2029
ddIsObjLibMarked 2030
ddIsObjReadable 2031
ddIsObjReadableEx 2032
ddIsObjWritable 2033
ddIsObjWritableEx 2035
ddIsPathManageable 2036
ddIsPathManaged 2037

© 2020 All Rights Reserved.

ddLibIs5X 2038
ddLibHasTmpDir 2039
ddLockDescribe 2040
ddLockFree 2041
ddLockGetStatus 2042
ddLockPath 2044
ddLockSet 2045
ddLockSetEx 2047
ddMapGetDataTypeFileName 2049
ddMapGetDataTypeList 2050
ddMapGetFileDataType 2051
ddMapGetFileTool 2052
ddMapGetFileViewType 2053
ddMapGetToolNameDataType 2054
ddMapGetToolNameList 2055
ddMapGetToolNameViewType 2056
ddMapGetViewTypeFileName 2057
ddMapGetViewTypeList 2058
ddMapIsFileNameKnown 2059
ddMapIsToolLegal 2060
ddMapIsToolNameKnown 2061
ddMarkObjMaster 2062
ddNeedCheckout 2063
ddObjHasNamedObj 2064
ddObjHasNamedObjEx 2065
ddObjIsInMaster 2067
ddObjIsInTmp 2068
ddProduceObjRelNames 2069
ddProduceObjRelNamesEx 2071
ddRegHiddenCellsFunc 2073
ddRegTrigger 2075
ddReleaseObj 2077
ddSetForcedLib 2078
ddSetForcedLibEx 2079
ddSetLibUnmanaged 2081
ddSetLibDmType 2082

© 2020 All Rights Reserved.

ddSetTmpdirWarn 2083
ddUnregHiddenCellsFunc 2084
ddUnregTrigger 2085
ddUpdateLibList 2086
Design Data Services 2088
Library Manager 2088
Hierarchy Editor 2088
Library Browser 2088
Environment Variables 2088
Current Library 2091
Synchronization 2091
Integrating Your Own Library Browser 2091
Registering the Custom Library Browser 2093
SKILL Procedural Interface Functions 2093
Customizing CIW Menus 2093
ciwAddTranslatorItem 2094
ciwCreateMenu 2095
ciwCreatePulldown 2096
ciwHiExit 2097
ciwMenuInit 2098
ciwMenuInstall 2099
ciwMenuLoadFile 2100
ciwRemoveTranslatorItem 2101
ddsAboutDFII 2102
ddsBrowserAction 2103
ddsCheckinPreferences 2105
ddsCheckoutPreferences 2106
ddsCheckSessionName 2107
ddsCloseLibManager 2108
ddsEditProp 2109
ddsEndSyncWithForm 2110
ddsFileBrowseCB 2111
ddsGetCurrentLib 2114
ddsGetCustomLibSelect 2115
ddsGetServiceName 2116
ddsHiCloseData 2117

© 2020 All Rights Reserved.

ddsHiCreateLibrary 2118
ddsHiDefragCellView 2119
ddsHiDefragLib 2120
ddsHiMakeReadOnly 2121
ddsHiOpenCdsLibEditor 2122
ddsHiPreferences 2123
ddsHiRefresh 2124
ddsLEWhatsNewAtStartup 2125
ddsOnLineHelp 2126
ddsOpenConvertToolBox 2127
ddsOpenLibManager 2128
ddsProgramName 2129
ddsRefresh 2130
ddsRegPreRefreshTrigger 2132
ddsRegPostRefreshTrigger 2134
ddsRegCustomLibSelect 2136
ddsHiSaveData 2138
ddsServSelect 2139
ddsSetLibManLCV 2140
ddsGetLibManLCV 2141
ddsSyncWithForm 2142
ddsUnregCustomLibSelect 2146
ddsUpdateSyncWithForm 2147
ddsViewDDProp 2148
ddsWhatsNew 2149
Custom Library Browser Template 2150
dmb Compatibility Functions 2158
dmbGetCurrentLib 2158
dmbLibBrowserFormDone 2159
dmbOpenLibBrowser 2161
dmbSetBrowserInteractions 2162
dmbSyncWithForm 2164
dmbSyncWithCellForm 2167
dmbUpdateBrowser 2170
Cell Category Functions (ddCat Interface) 2172
ddCatOpen 2173

© 2020 All Rights Reserved.

ddCatOpenEx 2175
ddCatAddItem 2177
ddCatSubItem 2178
ddCatSave 2179
ddCatRemove 2180
ddCatClose 2181
ddCatGetCatName 2182
ddCatGetCatMembers 2183
ddCatGetLibCats 2184
ddCatFindCat 2185
ddCatGetCellsNotInCat 2186
ddCatIsObjInCat 2187
Old Cell Category Functions 2188
dmAddCellToCategory 2188
dmCreateCellCategory 2190
dmRemoveCellCategory 2191
dmRemoveCellFromCategory 2192
Design Editor SKILL Functions 2193
Registering and Un-registering Applications 2193
deRegApp 2194
deUnRegApp 2199
deRegUserTriggers 2200
deUnRegUserTriggers 2202
deInheritEnabling 2203
deUnInheritEnabling 2204
deInheritMenus 2205
deUnInheritMenus 2206
deRegProbeTrigger 2207
deUnRegProbeTrigger 2208
deGetProbeTrigger 2209
deGetAppInfo 2210
deHiSetDefaultApplication 2213
deSetWorkspace 2214
Registering DE Plugins 2215
deRegPlugin 2218
deInstallPlugin 2221

© 2020 All Rights Reserved.

deRemovePlugin 2222
deFindPlugin 2223
deBeginConfigurePlugins 2224
deEndConfigurePlugins 2225
deFindAssistant 2226
deGetPlugins 2228
deGetAppClassInfo 2229
deoEnableTrigger 2231
deoExitTrigger 2233
deoGetAssistantList 2234
deoGetBindkeys 2235
deoMenuTrigger 2236
deoOnTabSwitch 2238
deoOnApplicationChange 2239
deoPostInstallTrigger 2240
deoRequireToolbars 2242
deoUpdateToolbar 2243
deoOnEvent 2245
Client Interface 2246
deFileNew 2246
deOpen 2247
deFileOpen 2255
deNew 2256
deNewCellView 2259
deOpenCellView 2261
deBack 2263
deForward 2264
dePush 2265
deSetHistoryOpenMode 2267
deSwap 2268
deToggleAssistants 2270
deToggleAssistantVisibility 2271
deToggleToolbars 2272
deFindToolbar 2273
deManageToolbar 2275
deCustomizeToolbars 2276

© 2020 All Rights Reserved.

deUnmanageToolbar 2277
deEditInPlace 2278
deReturn 2280
deCopy 2282
Trigger Arguments 2284
dataTrigger 2286
appTrigger 2288
userAppTrigger 2292
enableTrigger 2293
menuTrigger 2294
userMenuTrigger 2295
postInstallTrigger 2296
userPostInstallTrigger 2297
probeTrigger 2298
Licensing Functions 2300
lbuiGetCheckedOutLicenses 2300
lbuiPrintLicenseUseReport 2301
iclicCheckOutAndLock 2303
eliCheckOutAndLock 2304
Design Editor Support Routines 2305
deGetCellView 2305
deGetEditCellView 2306
deSetCellView 2307
deGetViewType 2308
deGetEditViewType 2309
deGetRootType 2310
deGetEditRootType 2311
deGetContext 2312
deGetEditContext 2313
deIsEditInContext 2314
deSetContext 2315
deGetMaxInContextLevel 2316
deGetConfigId 2317
deSetConfigId 2318
deGetVector 2319
deGetEIP 2320

© 2020 All Rights Reserved.

deGetPushed 2321
deGetProbeContext 2322
deSetProbeContext 2323
deGetEditLevel 2324
deGetMaxEditLevel 2325
deEnableApp 2326
deDisableApp 2327
deInstallApp 2328
deGetAllApplicationNames 2329
deGetAllRootTypes 2330
deGetAllViewTypes 2331
deGetRootViewTypeList 2332
deMapUserTypeToViewType 2333
deMapViewTypeToUserType 2334
deViewTypeIsRegistered 2335
deGetLowerTierApp 2336
Extended Design Editor Example 2338

A
Unsupported Functions 2345

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

Preface

The *Virtuoso Design Environment SKILL Reference* manual provides detailed information about the SKILL functions that interface to Virtuoso Design Environment applications.

Details about each function include its syntax, argument descriptions, values returned, and examples.

- Scope
- Licensing Requirements
- Related Documentation
- Additional Learning Resources
- Customer Support
- Feedback about Documentation
- Using SKILL Code Examples
- Understanding Cadence SKILL
- Typographic and Syntax Conventions
- Identifiers Used to Denote Data Types

© 2020 All Rights Reserved.

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.18) and advanced node (for example, ICADVM20.1) releases.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in the ICADVM20.1 advanced node and advanced methodologies release.
(IC6.1.8 Only)	Features supported only in mature node releases.
(ICADVM20.1 Virtuoso MultiTech Framework Only)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_MultiTech_Framework (95022) license.

Licensing Requirements

Product license 111 is checked out at `SKILL` executable startup or at workbench startup.

For information about licensing in the Virtuoso design environment, see *Virtuoso Software Licensing and Configuration Guide*.

Related Documentation

What's New and KPNS

- *Virtuoso Design Environment SKILL What's New*
- *Virtuoso Design Environment SKILL Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Design Environment SKILL Reference*

© 2020 All Rights Reserved.

- *Cadence Application Infrastructure User Guide*

Virtuoso Tools

- *Virtuoso Schematic Editor User Guide*
- *Virtuoso Multi-Patterning Technology User Guide* **(ICADVM20.1 Only)**

SKILL Documents

- The SKILL programming language is documented in the following manuals:
  - *Cadence SKILL Language User Guide*
  - *Cadence SKILL Language Reference*
  - *Cadence SKILL Development Reference*
  - *Cadence SKILL IDE User Guide*
- SKILL access to other applications is provided in the following manuals:
  - *Virtuoso Technology Data SKILL Reference*
  - *Virtuoso Layout Suite SKILL Reference*
  - *Virtuoso Schematic Editor SKILL Reference*
  - *Cadence User Interface SKILL Reference*
  - *Virtuoso Analog Design Environment L SKILL Reference*
  - *Cadence Interprocess Communication SKILL Reference*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

© 2020 All Rights Reserved.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses on Virtuoso Space-based Router:

- Virtuoso Layout Design Basics
- Virtuoso Layout for Advanced Nodes

Cadence also offers the following training courses on the SKILL programming language, which you can use to customize, extend, and automate your design environment:

- SKILL Language Programming Introduction
- SKILL Language Programming
- Advanced SKILL Language Programming

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.

© 2020 All Rights Reserved.

- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.

On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

© 2020 All Rights Reserved.

Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

Using SKILL Code Examples

The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.

axlGetRunStatus

```
axlGetRunStatus(
    t_sessionName                          <- Required argument
    [ ?optionName t_optionName ]           <- Optional keyword argument
    [ ?historyName t_historyName ]         <- Optional keyword argument
    )
    => l_statusValues                      <- Return value
```

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName` *`t_optionName`* and `?historyName` *`t_historyName`* are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

© 2020 All Rights Reserved.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, *`l_statusValues`*.

Example

```
axlSession=axlGetWindowSession( hiGetCurrentWindow())
=> "session0"
axlGetRunStatus("session0" ?historyName "Interactive.10" ?optionName "tests")
=> (1 2)
```

Return value

Value of the session name argument

Question mark and argument name before the value of the keyword argument

Question mark and argument name before the value of the keyword argument

Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName` *`t_functionName`* `] )` in the CIW.
- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

  To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

© 2020 All Rights Reserved.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
`|`	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument.
	Used with brackets to indicate that you can specify zero or more arguments.
	Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

© 2020 All Rights Reserved.

Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, `t` is the data type in `t_viewName`. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

Prefix	Internal Name	Data Type
`a`	array	array
`A`	amsobject	AMS object
`b`	ddUserType	DDPI object
`B`	ddCatUserType	DDPI category object
`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	generic design management (GDM) spec object
`h`	hdbobject	hierarchical database configuration object
`I`	dbgenobject	CDB generator object
`K`	mapiobject	MAPI object
`l`	list	linked list
`L`	tc	Technology file time stamp
`m`	nmpIlUserType	nmpIl user type
`M`	cdsEvalObject	cdsEvalObject
`n`	number	integer or floating-point number
`o`	userType	user-defined type (other)
`p`	port	I/O port
`pkx`	package	package

© 2020 All Rights Reserved.

Prefix	Internal Name	Data Type
q	gdmspecListIlUserType	gdm spec list
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
T	txobject	transient object
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	hdbpath
w	wtype	window type
sw	swtype	subtype session window
dw	dwtype	subtype dockable window
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

For more information, see *Cadence SKILL Language User Guide*.

© 2020 All Rights Reserved.

1

Graphics Editor Functions

This chapter describes the SKILL interface of the Graphics Editor. The Graphics Editor provides user-level functions for editing, edit-in-place, transformations, selecting, environment, and so forth. This approach ensures that all basic editing functions are done in an identical manner across all applications. Application functions created from this common core also behave similarly across the product line.

This chapter contains the following sections:

- Design Functions
- Automatic Menu Enabling SKILL Functions
- Edit/Display Functions
- Refresh Functions
- View Level Functions
- Viewport Functions
- Procedural Level Functions
- Window Scrolling Functions
- Selection Functions
- Stored Selection
- Highlight Functions
- Background Cellview Functions
- Marker Functions
- Probing Functions
- Probe File Format
- Icon Creation and Loading Functions

© 2020 All Rights Reserved.

- Utility Functions
- Microwave Object FunctionsMouse Functions

Design Functions

A design can be opened for reading or editing. To open a design, you select *Open* from the *File* menu in the command interpreter window (CIW). A form is displayed for you to specify the library, cell, and view name of the design and the access mode, which can be *edit* or *read.*

- Edit mode excludes other users from editing the same file, although they can still view the old copy saved to disk.
- Read mode allows users to share design files.

Whether a design is opened in edit or read mode, all designs under the hierarchy of that design are opened in read mode. The only exception is edit-in-place, where both the top design and the design being edited can be modified; the other designs maintain their non-modifiable status.

After a design is opened, you can change the mode from *read* to *edit.* If a design has already been modified, changing the mode from *edit* to *read* forces the system to prompt you to save or discard the changes made. If you choose the *discard* option, the changes are ignored and the original design is brought back from the disk. You cannot make a cell editable if you do not have write permission.

The name of the current design and its mode are displayed in the window banner.

© 2020 All Rights Reserved.

geChangeCellView

```
geChangeCellView(
     [ w_windowId ]
     [ t_libName ]
     [ t_cellName ]
     [ t_viewName ]
     [ t_accessMode ]
     )
     => w_windowId
```

Description

Opens a design in an existing window.

It is like `geOpen`; however, if the window is not specified, the current window is used. `geChangeCellView` does not create a new window. If you have modified the current design in the window, a form is displayed asking you what to do with the design. If the library is not in the search path, a form is displayed, prompting you to change the search path.

Arguments

w_windowId	Object ID of the window containing the cellview.
t_libName	Name of the library for the design.
t_cellName	Name of the cell for the design.
t_viewName	Name of the view for the design.
t_accessMode	Access mode. Valid Values: `r` (read), `a` (edit)

Value Returned

w_windowId	Returns the database ID of the window passed in.

Example

```
geChangeCellView( win2 "testLib" "testCell" "testView" "a")
```

Returns `win2`.

© 2020 All Rights Reserved.

geChangeEditMode

```
geChangeEditMode(
     t_newMode
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the mode of a design, allowing you to edit or view the design.

Arguments

t_newMode	New access mode. `r` (read), `a` (edit).
w_windowId	Database ID of window containing the design you want to change.

Value Returned

`t`	The mode is changed.
`nil`	The mode is not changed.

Example

```
geChangeEditMode( "r" win3 )
```

Changes the mode in `win3` to read.

© 2020 All Rights Reserved.

geDiscardEdits

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

Description

Discards all edits made to a design.

Arguments

w_windowId	Database ID of window whose edits are to be discarded. If not specified, the current window is used.

Value Returned

`t`	The edits are discarded.
`nil`	The edits are not discarded.

Example

```
geDiscardEdits( win3 )
```

Discards the edits in `win3`.

© 2020 All Rights Reserved.

geNewWindow

```
geNewWindow(
     [ w_windowId ]
     )
     => w_windowId / nil
```

Description

Makes a copy of a window.

The copy window views the same design and uses the same viewing parameters as the source window. You can change the pan, zoom, and other parameters of the new window independently of the old window.

Arguments

w_windowId	Database ID of the source window for the copy. If not specified, the current window is used.

Value Returned

w_windowId	The name of the new window is copied.
nil	The name of the new window is not copied.

Example

```
geNewWindow( win3 )
```

Copies `win3` (returns `win4` if it is successful).

© 2020 All Rights Reserved.

geReturn

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

Description

Returns to the previous (parent) level displayed in a window before a descend, edit in place, or switch was done.

Note: This function does not support fig groups.

Arguments

w_windowId	Database ID of the window containing the cellview.

Value Returned

`t`	The display returns to the previous level.
`nil`	The display does not return to the previous level.

Example

```
geReturn( getCurrentWindow( ) )
```

Returns to the previous level in the current window.

© 2020 All Rights Reserved.

geReturnToLevel

```
geReturnToLevel(
     w_windowId
     )
     => nil
```

Description

Displays a list box containing the levels to which you can return.

Arguments

w_windowId	Database ID of the window containing the design.

Value Returned

`nil`	Always returns `nil`.

Example

```
geReturnToLevel( getCurrentWindow( ) )
```

Displays a list box of the levels you can return to in the current window.

© 2020 All Rights Reserved.

geSave

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

Description

Saves a design.

Arguments

w_windowId	Database ID of the window containing the design to save. If not specified, the current window is used.

Value Returned

`t`	The cellview is saved.
`nil`	The cellview is not saved.

Example

```
geSave( win2 )
```

Saves the design in `win2`.

© 2020 All Rights Reserved.

geSaveAs

```
geSaveAs(
     [ w_windowId ]
     [ t_libName ]
     [ t_cellName ]
     [ t_viewName ]
     )
     => t / nil
```

Description

Saves a design under another name.

If *t_cellName*, *t_viewName*, and *t_libName* are valid strings, the design is saved under that name. If any of these arguments is not specified or is not a valid string, a form is displayed for you to enter new names or cancel. The form displays the current names as defaults. If you enter a new name in the form, click *Apply* to save the design under that name. If the filename already exists, you are warned and given a chance to cancel or save to a different name. The window continues to view the design with the old name; you must open the design with the new filename if you want to see the new copy in a window.

Arguments

w_windowId	Database ID of the window containing the design to save. If not specified, the current window is used.
t_libName	Library name into which the cellview is saved.
t_cellName	New cell name for the design.
t_viewName	New view name for the design.

Value Returned

`t`	The cellview is saved.
`nil`	The cellview is not saved.

Example

```
geSaveAs( win2 "newLibName" "newCellName" "newViewName" )
```


© 2020 All Rights Reserved.

Returns `t`. The data is saved in *`newLibName`* with cell name *`newCellName`* and view name *`newViewName`*. The design in *`win2`* retains the old name and library.

© 2020 All Rights Reserved.

geSaveHier

```
geSaveHier(
     d_cellviewId
     [ x_depth ]
     [ g_askForSave ]
     )
     => t
```

Description

Saves a design hierarchically starting from the edit cellview of the specified window to the specified depth.

Note: `geSaveHier` works only on maskLayout views.

Arguments

d_cellviewId	The cellview ID of the window containing the design to save. If not specified, the current window is used.
x_depth	The depth hierarchy upto which the design needs to be saved. If not specified, the maximum depth (32) is used.
g_askForSave	When set to t, prompts you to specify the cellviews that need to be saved. By default, `g_askForSave` is set to t.

Value Returned

`t`	The design is saved hierarchically.

Example

```
geSaveHier(geGetEditCellView()32 t)
```


© 2020 All Rights Reserved.

Automatic Menu Enabling SKILL Functions

The menu update system is a maintainable, extensible mechanism for specifying when a menu in the Virtuoso Layout Editor is enabled or disabled. Several applications such as the layout editor itself and Virtuoso® XL Layout Editor can define update policies simultaneously.

For each menu, you can define an enable predicate that specifies when the menu is to be enabled. The enable predicate is an integer (`fixnum`) whose bits correspond to status bits in a status mask defined by your application. When the Virtuoso tool updates the window menus, it Boolean ANDs each menu item's enable predicates with the corresponding window status. Each window can have more than one window status mask. For example, one layout window might have a status mask for the layout editor, a separate status mask for the XL layout editor, and a third user status mask. Likewise a menu item can have a separate enable predicate that corresponds to each window status mask. A menu item is enabled if all the status flags specified by all of the menu item's enable predicates are TRUE. That is determined by bitwise ANDing each enable predicate with the corresponding window status.

To define an update policy in your application, you

1. Decide what status flags your application needs.

   For example, the Layout editor defines the following flags:

   ```
   layoutIsEditable = 2
   layoutIsModified = 4
   shapeIsSelected = 8
   ```

   Note: The status flags must be powers of two so that each corresponds to one bit in the status number.

2. Set the enable predicate on one or more menus items. The enable predicate is a bit mask. The bitmask is formed by combining the status flags using the bitwise OR operator.

   For example, the `createShapeMenu` and the `deselectAllMenu` enable predicates are set as follows:

   ```
   geSetEnablePredicate(createShapeMenu "LayoutApp" layoutIsEditable)
   geSetEnablePredicate(deselectAllMenu "LayoutApp" shapeIsSelected)
   ```

   Note: You must pass the menu object, not the menu's symbolic name, to `geSetEnablePredicate`.

3. Use `geSetWindowStatus` to set the current status whenever your application's status changes. The Virtuoso tool enables or disables your menus according to the policy you define.

   ```
   currentStatus = geGetWindowStatus(windowId "LayoutApp")
   currentStatus = if(cellview~>modifiedButNotSaved
           then currentStatus | layoutIsModified
   ```


© 2020 All Rights Reserved.

```
        else currentStatus & ~layoutIsModified
)
geSetWindowStatus(windowId "LayoutApp" currentStatus)
```


© 2020 All Rights Reserved.

geGetDisabledMenus

```
geGetDisabledMenus(
     w_windowId
     [ l_menuList ]
     )
     => l_disabledMenus
```

Description

Given a *windowId*, returns all the disabled submenus for all banner menus on that window. If *menuList* is specified, only the disabled submenus on those banner menus are returned.

The returned list contains one sub-list for each banner menu specified. Each sublist contains the symbolic names of all the disabled submenus of the banner menu, or `nil` if the corresponding banner menu does not contain any disabled submenus. If a slider-menu item is disabled, the slider-menu's submenus are omitted from the sublist. Each sublist is sorted alphabetically by symbolic name.

`geGetDisabledMenus` is useful for testing a menu disabling policy such as that implemented by `geUpdateWindowMenus`.

Arguments

w_windowId	The ID of the window containing the menus you want.
l_menuList	A list of the symbolic names of banner menus or the symbolic name of one banner menu. If specified, only the disabled submenus of the menus in *l_menuList* are returned.

Value Returned

l_disabledMenus	A list that contains one sublist for each banner menu specified. Each sublist contains the symbolic names of all the disabled submenus of one banner menu. Each sublist is sorted alphabetically. If a banner menu does not have any disabled submenus, the corresponding sublist is `nil`.

Example

```
windowId = window(2)
=> window:2
deGetRootType(windowId)
```


© 2020 All Rights Reserved.

```
=> "maskLayout"
geGetDisabledMenus(windowId)
=> (nil
    (leDesignDescendItem leDesignEditInPlaceItem
     leDesignResetItem leDesignReturnItem
     leDesignReturnToLevelItem leDesignSaveItem
    )
    nil
    (leCreateConicsItem leCreateContactItem leCreateInstItem
     leCreateLabelItem leCreatePathItem leCreatePinItem
     leCreatePolygonItem leCreateRectItem leCreateSymDevItem
    )
    (leEditAttachItem leEditChopItem leEditConnectivityItem
     leEditConvertToPolygonItem leEditDeleteItem
     leEditDeselectAllItem leEditFlattenItem
     leEditMakeCellItem leEditMergeItem leEditReshapeItem
     leEditRotateFigItem leEditSizeItem leEditSplitItem
     leEditStretchItem leModifyCornerItem
    )
    (leVerifyDRCItem leVerifyExtractItem)
    (leMiscLayerGenItem leMiscMoveOriginItem
     leMiscYankPasteItem)
)
```

Note:

- The sub-lists correspond to the banner menus in the Virtuoso tool.
- Some sub-lists are `nil`.
- Sub-lists are sorted alphabetically.
- The *Create – Conics* menu containing *Circle*, *Ellipse*, and *Donut* is not included in the *Create* banner menu sublist because its parent menu, `leCreateConicsItem`, is disabled.

© 2020 All Rights Reserved.

geGetEnablePredicate

```
geGetEnablePredicate(
     r_menu
     t_appName
     )
     => n_enablePredicate
```

Description

Returns the enable predicate of the `menu` for the application named *`appName`* or `nil` if undefined.

Arguments

`r_menu`	The array, not the symbolic name, of a menu that is returned by `hiCreateMenuItem` or `hiCreateSliderMenuItem`.
`t_appName`	The text or symbolic name used to uniquely identify your application's menu update system.

Value Returned

`n_enablePredicate`	The enable predicate of the `menu` for the application *`appName`*.
`nil`	Undefined

Example

```
leCreateConicsItem
=> array[8]:108643592
geGetEnablePredicate(leCreateConicsItem 'default)
=> 2
```


© 2020 All Rights Reserved.

geGetWindowStatus

```
geGetWindowStatus(
     w_windowId
     t_appName
     )
     => n_status
```

Description

Returns the status for the application named *appName* on the window *windowId*.

Arguments

w_windowId	The ID of the window whose status you want.
t_appName	The text or symbolic name used to uniquely identify your application's menu update system.

Value Returned

n_status	The status for application named *appName* on window *windowId*.
nil	The status for *appName* is undefined.

Example

```
windowId = window(2)
=> window:2
geGetWindowStatus(windowId 'default)
=> 66
geGetWindowStatus(windowId "UndefinedApplication")
=> nil
```


© 2020 All Rights Reserved.

geSetEnablePredicate

```
geSetEnablePredicate(
     r_menu
     t_appName
     n_enablePredicate
     )
```

Description

Sets the enable predicate of the *r_menu* for *t_appName* to *n_enablePredicate*.

The enable predicate is a bit mask that specifies when menu should be enabled. Each bit in *n_enablePredicate* corresponds to an application-defined status, for example “window is editable” or “can create instances.”

Each application can set its own enable predicate, which means menu can have multiple enable predicates. For menu to be enabled, all the status bits for all enable predicates must be TRUE. When geUpdateWindowMenus is called, the enable predicates of menu are compared, logically ANDed, with the corresponding window status for *appName*. When all the status bits specified in the enable predicates for all applications are TRUE, menu is enabled. Otherwise, *menu* is disabled.

Arguments

r_menu	The menu whose enable predicate you want to set. This is an array name, not the symbolic name, of a menu that is returned by hiCreateMenuItem or hiCreateSliderMenuItem.
t_appName	The text or symbolic name of the application for which this enable predicate is set.
n_enablePredicate	An integer bit mask that specifies when *menu* needs to be enabled in *t_appName*.

Value Returned

None

Example

```
leCreateConicsItem
=> array[8]:108643592
```


© 2020 All Rights Reserved.

```
geSetEnablePredicate(leCreateConicsItem 'default lecLayoutIsEditable)
=> 2
```


© 2020 All Rights Reserved.

geSetWindowStatus

```
geSetWindowStatus(
     w_windowId
     t_appName
     n_status
     )
     => n_previousStatus
```

Description

Sets the status for *w_windowId* and *t_appName* to *n_status*.

n_status is a bit field that indicates the current status of the window. Each bit corresponds to a status such as “window is editable” or “can create instances.” Each application can define its own status field. When `geUpdateWindowMenus` is called, the enable predicate of each menu is compared with the corresponding status. If all the enable predicates on a menu are TRUE, the menu is enabled. Otherwise, the menu is disabled.

Arguments

w_windowId	The ID of the window whose status you want to set.
t_appName	The text or symbolic name of the application whose status you want to set.
n_status	A fixed number that specifies the current state of *t_appName* in *w_windowId*.

Value Returned

n_previousStatus	Previous status for *appName* on *w_windowId* or `~0` if the previous status was undefined.

Example

```
windowId = window(2)
=> window:2
geSetWindowStatus(windowId 'default 64)
=> 66
```

Note: The menus are not updated until `geUpdateWindowMenus` is called. `geUpdateWindowMenus` is automatically called by the layout editor.

© 2020 All Rights Reserved.

geUpdateWindowMenus

```
geUpdateWindowMenus(
     w_windowId
     )
```

Description

Updates the enabled status of all the menu items in the window's banner menu hierarchy and all of the window's fixed menu items.

Whether a menu item is enabled or disabled is determined by comparing that menu item's enable predicates with the corresponding application status in the window. The enable predicates are matched to the status value according to the *appName* specified with `geSetEnablePredicate` and `geSetWindowStatus`. The comparison used is

```
(enablePredicate & status) == enablePredicate
```

If all of the comparisons are TRUE, the menu item is enabled. Otherwise, the menu item is disabled.

If a menu item does not have an enable predicate, the menu item's enabled state is not changed. The menu is neither enabled or disabled. However, the menu item's submenus, if any, are updated.

If a menu item is disabled, the menu item's submenus are not updated. Otherwise, if the menu item is enabled, that menu's submenus are updated.

If an application's current status is undefined and `~0` is used for the application's status, all status bits are TRUE.

Note: Layout window menus are automatically updated after every HI event. So you do not need to explicitly call `geUpdateWindowMenus` to update the menus.

Arguments

w_windowId	The ID of the window whose menus you want to update.

Value Returned

Returns a list of all the menu update system applications that changed since the last call to `geUpdateWindowMenus`.

© 2020 All Rights Reserved.

Edit/Display Functions

A cellview is a specific view of a particular cell. There are many different ways of viewing a cell and manipulating the design. The following functions help you get the cellview you want, determine the instance hierarchy, and assign a cellview to a particular window.

geClearIgnoreProp

```
geClearIgnoreProp(
     l_inst | l_net
     )
     => l_obj
```

Description

Deletes the `ignoreCheck` property for all objects given in the list. The name of the property is `ignoreCheck*` where `*` is a numerical value.

Arguments

l_inst | *l_net*	List of instances or list of nets

Value Returned

l_obj	Returns a list of objects passed to the function.

Example

```
geClearIgnoreProp(geGetSelSet())
```

Clears the property on the selected instances.

```
geClearIgnoreProp(geGetEditCellView()~>nets)
```

Clear the property on all the nets in the current cellview.

© 2020 All Rights Reserved.

geComputeQueryBbox

```
geComputeQueryBbox(
     l_point
     [ w_windowID ]
     )
     => queryBbox / nil
```

Description

This function takes a point as parameter and returns a corresponding bounding box. The point that is specified is the center of the bounding box. The lower left and upper right corners of the bounding box are based on the selectionAperture.

If the window is a layout window, delta = selectionAperture/2.

If the window is a schematic window, delta = selectionAperture.

If *w_windowId* is not specified, the current windowId is used.

Arguments

l_point	Point to be used for bounding box center.
w_windowID	Optional argument specifies the window to be used.

Value Returned

t	Returns bounding box coordinates.
nil	Unable to process bounding box.

© 2020 All Rights Reserved.

geCopySelSet

```
geCopySelSet(
     [ w_windowId ]
     [ l_sourcePt ]
     [ l_destPt ]
     )
     => t / nil
```

Description

Procedural interface to copy the selected set.

Arguments

w_windowId	Window ID of the window to use. If not specified, the current window is used.
l_sourcePt	The starting point for the copy.
l_destPt	The destination point for the copy.

Value Returned

`t`	The selected objects are copied.
`nil`	The selected objects are not copied.

Example

```
geCopySelSet(hiGetCurrentWindow()list(456.7 234.5)list(678.9 567.8))
```

Copies objects in the selected set from the current window to the current window.

Interactive Function

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

Enter this function with only the window ID argument; the system prompts you for the starting and destination point that encloses the objects to be copied. If *w_windowId* is not specified, the current window is used.

© 2020 All Rights Reserved.

geDeleteSelSet

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

Description

Deletes all 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 to use. If not specified, the current window is used.

Value Returned

t	The selected objects are deleted.
nil	The selected objects are not deleted.

© 2020 All Rights Reserved.

geMenuForObjectsUnderPoint

```
geMenuForObjectsUnderPoint(
     w_windowId
     l_point
     )
     => t / nil
```

Description

Creates the *(De)Select Under Cursor* menu for the specified window and point.

Arguments

w_windowId	Window ID of the window for which the menu is created.
l_point	Objects at the point are added to the menu.

Value Returned

`t`	Menu has been created.
`nil`	Menu has not been created.

© 2020 All Rights Reserved.

geSelectNextInSelSet

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

Description

Selects an object from the selected set of objects in the specified cellview.

The objects are randomly selected and there is no predefined order. For example, if you select three objects, A B C. Then, calling the `geSelectNextInSelSet()` function will select A. When you next call the `geSelectNextInSelSet()` function, the object B is selected, and so on.

Arguments

d_cellviewId	(Optional) ID of the cellview containing the selected set of objects. If no cellview ID is specified, the edit cellview of the current window is used.

Value Returned

`t`	The object is selected
`nil`	The object is not selected

© 2020 All Rights Reserved.

geSelectPreviousInSelSet

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

Description

Selects the last selected object in the specified cellview.

Consider an example in which three objects, A B and C, are selected. Then, calling the `geSelectNextInSelSet()` function will select A. Calling the `geSelectNextInSelSet()` function again will select B. If you now call `geSelectPreviousInSelSet()` function, object A will be selected.

Arguments

d_cellviewId	(Optional)ID of the cellview containing the selected set of objects. If the cellview ID is not specified, the edit cellview of the current window is used.

Value Returned

`t`	The object is selected
`nil`	The object is not selected

© 2020 All Rights Reserved.

geSelectToggleObject

```
geSelectToggleObject(
     [ w_windowID ]
     )
     => t / nil
```

Description

Toggles the state of the selected objects in the specified window. This means that the function deselects the selected objects and selects the remaining objects.

Arguments

w_windowID	(Optional) Specifies the window containing the selected set of objects. If this is not specified, the objects in the current window are toggled.

Value Returned

`t`	The objects in the window are successfully toggled
`nil`	The objects in the window could not be toggled

© 2020 All Rights Reserved.

geDispWinEnvProps

```
geDispWinEnvProps(
     [ t_type ]
     )
     => t / nil
```

Description

Lists all the window environment variables that can be set using the `envSetVal` function, or stored inside the `.cdsenv` file.

Arguments

t_type	Optional argument that describes the type of environment variables.
	`all`
	`boolean`
	`integer`
	`float`
	`string`
	`list`
	Default Value: `all`

Value Returned

`t`	Lists of available variables.

Example

```
geDispWinEnvProps()
geDispWinEnvProps("all")
geDispWinEnvProps("boolean")
geDispWinEnvProps("integer")
```


© 2020 All Rights Reserved.

geEditSelSet

```
geEditSelSet(
     [ d_cellview ]
     )
     => t / nil
```

Description

Displays and edits the properties of each object in the selected set in a cellview.

Arguments

d_cellview	Database ID of the cellview containing the selected set. If not specified, the cellview from the current window is used.

Value Returned

`t`	The selected objects are found.
`nil`	The selected objects are not found.

Example

```
geEditSelSet( geGetEditCellView( ) )
```

Displays the Property List Editor form for the objects in the selected set in the current cellview.

© 2020 All Rights Reserved.

geEditSingleObject

```
geEditSingleObject(
     d_objectId
     )
     => t / nil
```

Description

Displays and edits the properties of an object.

Arguments

d_objectId	Database ID of the object to edit.

Value Returned

`t`	The object is found.
`nil`	The object is not found.

Example

```
figId = car( geGetSelectedSet( ) )
geEditSingleObject( figId )
```

Displays the Property List Editor form for the object whose database ID is `figId`, or for the first object in the selected set.

© 2020 All Rights Reserved.

geEditToWindowPoint

```
geEditToWindowPoint(
     w_windowId
     l_editCellViewPoint
     )
     => l_windowCellViewPoint
```

Description

Translates coordinate values when you are using edit in place. `geWindowToEditPoint` applies the top-level cellview's coordinates to the cellview being edited; `geEditToWindowPoint` does the reverse.

All SKILL functions apply to the cellview being edited in the current window. When you are editing in place, this cellview is not the same one as the top-level cellview being displayed in the window. Coordinate values passed to SKILL functions are always expressed with respect to the cellview being displayed, even when you are editing in place. Database access inquiries, however, always return coordinate values in terms of the coordinate system of the cellview being interrogated, regardless of the context that cellview is used in.

The role of `geWindowToEditPoint` and `geEditToWindowPoint` is to translate between the two coordinate systems.

`geEditToWindowPoint` expects an argument consisting of a coordinate pair list. It interprets the coordinates with respect to the coordinate system of the current *`l_editCellview`* and returns the corresponding list, taking all intermediate locations and transformations into account.

`geWindowToEditPoint` performs the translation in the opposite direction. Given a coordinate pair list, it interprets the coordinates with respect to the *window-cellview* and returns the corresponding list expressed in terms of the *edit-cellview*'s coordinate system. You can use this function, for example, to determine where an object placed at a given location in the *window-cellview* is situated in the *edit-cellview*.

© 2020 All Rights Reserved.

Arguments

w_windowId Database ID of the window containing the cellview being edited.

l_editCellViewPoint

Coordinate pair expressed in edit-cellview units (`geEditToWindowPoint`).

Value Returned

l_windowCellViewPoint

Coordinate pair expressed in window-cellview units (`geEditToWindowPoint`).

Example

Assume you are editing in place an instance placed at 12.5:10.25.

```
geWindowToEditPoint(window(4) 12.5:10.25)
```

Returns `0:0`, because this is the origin of the instance.

```
geEditToWindowPoint( window(4) 0:0 )
```

Returns `12.5:10.25`

© 2020 All Rights Reserved.

geEquivWindow

```
geEquivWindow(
     w_windowId
     )
     => w_windowEquiv / nil
```

Description

Returns the ID of a window that has the same edit cellview as the specified window.

If *w_windowId* is the only window with this edit cellview, *w_windowId* is returned.

Arguments

w_windowId	Database ID of the window whose equivalent you want to find. Defaults to the current window.

Value Returned

w_windowEquiv	The window with the same edit cellview as *w_windowId* if it exists.
`nil`	The window does not exist.

Example

Assume that window(2) and window(3) both are editing the same cellview.

```
geEquivWindow( window(2) )
```

Returns `window:3`.

© 2020 All Rights Reserved.

geEquivWindowSet

```
geEquivWindowSet(
     w_windowId
     )
     => l_windowList
```

Description

Returns a list of all windows that have the same edit cellview as the specified window.

If *w_windowId* is the only window with this edit cellview, a list containing only *w_windowId* is returned.

Arguments

w_windowId	Database ID of the window whose equivalent you want to find. Defaults to the current window.

Value Returned

l_windowList	List of windows with the same edit cellview as *w_windowId*.

Example

Assume that window(2) and window(3) both are editing the same cellview.

```
geEquivWindowSet(window(2))
```

Returns `(window:2 window:3)`.

© 2020 All Rights Reserved.

geFindPathToCellViewInst

```
geFindPathToCellViewInst(
     d_cellview
     d_masterId
     )
     => l_hierPath / nil
```

Description

Returns a hierarchical path of one of the references to a particular master in a cellview. This function returns the closest reference with the fewest levels of hierarchy.

Arguments

d_cellview	Database ID of the cellview.
d_masterId	Database ID of the master.

Value Returned

l_hierPath	The path to the master cellview.
nil	The cellview is not found.

© 2020 All Rights Reserved.

geClearNetNameDisplayFilter

```
geClearNetNameDisplayFilter(
     d_cellviewID
     )
     => t / nil
```

Description

If the specified cellview is editable, clears the net name filter setting currently saved for the cellview.

Arguments

d_cellviewID	ID of the cellview for which the net name filter settings are cleared.

Value Returned

`t`	The net name filter settings were cleared for the specified cellview ID.
`nil`	The command failed.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetNetNameDisplayFilter(cellView "Hide" list("VDD*" "VSS*"))
geSaveNetNameDisplayFilter(cellView)
geGetNetNameDisplayFilter(cellview t)
geClearNetNameDisplayFilter(cellview)
geGetNetNameDisplayFilter(cellview t)
```

Clears the filter applied to hide nets VDD and VSS.

© 2020 All Rights Reserved.

geGetNetNameDisplayFilter

```
geGetNetNameDisplayFilter(
     d_cellview
     [ g_fromDesign ]
     )
     => l_filterSettings / nil
```

Description

Returns the net name filtering settings currently in use for the given cellview.

Arguments

d_cellview	Database ID of the cellview.
g_fromDesign	When `t`, it returns the net name filtering settings that are currently saved in the design.

Value Returned

l_filterSettings	The net name filtering settings.
`nil`	The net name filtering settings were not found.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
netNameFilters = geGetNetNameDisplayFilter(cellView)
netNameFilters
(nil mode "Hide" netNames ("VDD" "VSS"))
netNameFilters->mode
"Hide"
netNameFilters->netNames
("VDD" "VSS")
```


© 2020 All Rights Reserved.

geIsNetNameDisplayActiveOnWindow

```
geIsNetNameDisplayActiveOnWindow(
     w_windowID
     )
     => t / nil
```

Description

Checks whether the Net Name Display functionality is supported for the given window ID.

Arguments

w_windowID	Window ID that needs to be checked for the Net Name Display functionality support.

Value Returned

`t`	The Net Name Display functionality is available because the specified window is open in Layout XL and the cellview supports the functionality. **Note:** The Net Name Display functionality being available does not mean that the functionality is enabled. Use geIsNetNameDisplayEnabled to check if the functionality is enabled for the specified window.
`nil`	The Net Name Display functionality is not available for the specified window.

Example

Evaluates `window 2` to check if the Net Name Display functionality is supported.

```
geIsNetNameDisplayActiveOnWindow(window(2))
```


© 2020 All Rights Reserved.

geIsNetNameDisplayEnabled

```
geIsNetNameDisplayEnabled(
     w_windowID
     )
     => t / nil
```

Description

Checks whether the Net Name Display functionality is enabled for the given window ID.

Arguments

w_windowID	Window ID for which the state of the Net Name Display functionality is evaluated.

Value Returned

`t`	The Net Name Display functionality is enabled for the given window ID.
`nil`	The Net Name Display functionality is not enabled for the given window ID.

Example

Evaluates `window 2` to check if the Net Name Display functionality is enabled.

```
geIsNetNameDisplayEnabled(window(2))
```


© 2020 All Rights Reserved.

geGetSortedSelectByLine

```
geGetSortedSelectByLine(
     [w_windowId]
     [l_linePoints]
     )
     => l_sortedObjects
```

Description

Returns the sorted list of the crossed objects by the line.

Arguments

w_windowId	Database ID of the cellview.
l_linePoints	Database ID of the master.

Value Returned

l_sortedObjects	List of sorted objects.

Example

```
line = list(-8.99:-3.45 -8.562:6.464 11.331:5.945 11.453:-2.336 -2.756:-2.214 -
2.8:3.3)
sorted = geGetSortedSelectByLine(hiGetCurrentWindow() line)
 ((db:0x2c07e01a)
     (db:0x2c07e11b)
     (db:0x2c07e11a)
     (db:0x2c07e49b))
```


© 2020 All Rights Reserved.

geGetCellViewWindow

```
geGetCellViewWindow(
     d_cellview
     )
     => w_windowId / nil
```

Description

Returns the ID of a window in which the specified cellview is displayed.

If the cellview is displayed in more than one window, the first window found is returned.

Arguments

d_cellview	Database ID of the cellview.

Value Returned

w_windowId	The database ID of the window if the cellview is displayed.
nil	The cellview is not displayed.

Example

```
geGetCellViewWindow( cell3 )
```

Returns `win3`.

© 2020 All Rights Reserved.

geGetEditCellView

```
geGetEditCellView(
     [ w_windowId ]
     )
     => d_cellview
```

Description

Returns the ID for the cellview being edited.

This value is normally the same as that returned by `geGetWindowCellView`. Only when you are editing a cell in place do the two functions return different results. If you know what window you are interested in, a shorthand equivalent to this function may be used: `window(n)->editCellView`.

Arguments

w_windowId	Database ID of the window of the cellview being edited.

Value Returned

d_cellview	Database ID of the cellview being edited in the window.

Example

```
geGetEditCellView( getCurrentWindow( ) )
```

Returns `db:654321`, which is the edited cellview.

© 2020 All Rights Reserved.

geGetEditCellViewWindow

```
geGetEditCellViewWindow(
     d_cellview
     )
     => w_windowId / nil
```

Description

Returns the ID of the window in which the specified cellview is the edit cellview.

Arguments

d_cellview	Database ID of the cellview.

Value Returned

w_windowId	Database ID of the window in which the specified cellview is the edit cellview.
nil	An edit cellview window was not found or an error occurred.

© 2020 All Rights Reserved.

geGetEditRep

```
geGetEditRep(
     [ w_windowId ]
     )
     => d_cellview
```

Description

An alias for `geGetEditCellView`. Returns the ID for the cellview being edited.

Arguments

w_windowId	Database ID of the window of the cellview being edited.

Value Returned

d_cellview	Database ID of the cellview being edited in the window.

© 2020 All Rights Reserved.

geGetHierMemInst

```
geGetHierMemInst(
     [ w_windowId ]
     )
     => l_list / nil
```

Description

Returns a list describing the hierarchy of instances that have been descended into or edited-in-place to arrive at the cellview in the window.

The list is in the form

```
dbInst1  3
dbInst2  4
```

Each element in the list represents the database ID of the instance and iteration number of the instance that was descended into or edited-in-place at a higher level of the window. If the cellview currently occupying the window is the top-level cellview—that is, no descends or edits-in-place have been done to arrive at it—then the list returned by `geGetHierMemInst` is `nil`.

Arguments

w_windowId	Database ID of the window from which the hierarchy information is wanted.

Value Returned

l_list	Instance hierarchy description for this window displayed in a list. The first element in the list is the instance ID. The second element in the list is the iteration number of the instance.
`nil`	The current window is empty or if the current window is undefined.

Example

```
geGetHierMemInst( getCurrentWindow( ) )
```

Returns

```
db:01234567  0
db:45678900  1
```


© 2020 All Rights Reserved.

geGetInstHier

```
geGetInstHier(
     [ w_windowId ]
     )
     => t_instance-hierarchy / nil
```

Description

Returns a string describing the hierarchy of instances that have been descended into or edited-in-place in the current window to arrive at the cellview in the window.

The string is in the form

```
/dbinst1/dbinst2/dbinst3/...
```

Slashes separate the string into components. Each component represents the name of the instance and possibly the iteration number of the instance that was descended into or edited-in-place at a higher level of the window. If the cellview currently occupying the window is the top-level cellview, that is, no descends or edits-in-place have been done to arrive at it, then the string returned by `geGetInstHier` is `/`.

The results of this function can be used in the Hierarchy item in the Open Design form.

Arguments

w_windowId	Database ID of the window from which the hierarchy information is wanted.

Value Returned

t_instance-hierarchy	Instance hierarchy description for this window.
`nil`	The current window is empty or the current window is undefined

Example

```
geGetInstHier( getCurrentWindow( ) )
```

Returns `/db:13240512/db:13405278`.

© 2020 All Rights Reserved.

geGetInstHierPath

```
geGetInstHierPath(
     w_windowId
     )
     => l_instance-hierarchy / nil
```

Description

Similar to `geGetInstHier` but returns additional information for mosaic instances.

This function returns a list of list describing the hierarchy of instances that have been descended into or edited-in-place in the current window to arrive at the cellview in the window. The list contains the following information for each instance:

(*dbinst memInst row column*)

where *dbinst* is the database name of the instance, *memInst* is the iteration number, *row* is the row position, and *column* is the column position of the instance.

If the cellview currently occupying the window is the top-level cellview, that is, no descends or edits-in-place have been done to arrive at it, then the string returned by `geGetInstHier` is `()`.

Arguments

w_windowId	Database ID of the window for which you want the hierarchy information.

Value Returned

l_instance-hierarchy

Returns a list of list (`dbInst memInst row column`).

`nil` The current window is empty or undefined.

Example

```
geGetInstHierPath( hiGetCurrentWindow( ) )
```

Returns `((db:13240512 0 0 0) (db:13402578 1 4 2))`.

© 2020 All Rights Reserved.

geGetObjectSelectedSet

```
geGetObjectSelectedSet(
     [ d_cellview ]
     )
     => list(
          list(d_selectedObject
               list(
                         list(d_instId x_memInst x_row x_column))
                         list(d_instId x_memInst x_row x_column))
                         ...
               )
          list(d_selectedObject
               list(
                         list(d_instId x_memInst x_row x_column))
                         list(d_instId x_memInst x_row x_column))
                         ...
               )
          ...
     )
     => l_selectedSet
```

Description

Returns a list of lists describing objects selected in the hierarchy of the given cellview. If the cellview is omitted, the current edit cellview of the current window is used. For each object in the selected set, the path to it consisting of the instance ID, the member instance, the row, and the column is given.

Arguments

d_cellview	Database ID of the top-level cellview in the window.

Value Returned

l_selectedSet	A list of lists describing objects selected in the hierarchy of the given cellview.

Example

Select a pin on an instance in a schematic.

© 2020 All Rights Reserved.

```
geGetObjectSelectedSet()
((db:3865864
        ((db:38370404 0 0 0)) ) )
```


© 2020 All Rights Reserved.

geGetTopLevelCellView

```
geGetTopLevelCellView(
     [ w_windowId ]
     )
     => d_cellview / nil
```

Description

Returns the ID of the top-level cellview displayed in a window.

This function returns `nil` if the current window is empty or if no current window is defined. Only if you descend in the hierarchy or do an edit in place would the top-level cellview be different from the edit and display cellviews. If you know the window you are interested in, a shorthand equivalent to this function can be used: `window(n)->topCellView`.

Arguments

w_windowId	Database ID of the window in which the cellview is at the top level.

Value Returned

d_cellview	Database ID of top-level cellview in the window.
`nil`	The current window is empty or no current window is defined.

Example

```
geGetTopLevelCellView( hiGetCurrentWindow( ) )
```

Returns `db:664422`, which is the top-level cellview.

© 2020 All Rights Reserved.

geGetTopCellViewWindow

```
geGetTopCellViewWindow(
     d_cellview
     )
     => w_windowId / nil
```

Description

Returns the ID of the window in which the specified cellview is the top cellview.

Arguments

d_cellview	Database ID of the cellview.

Value Returned

w_windowId	Database ID of the window in which the specified cellview is the top cellview.
nil	A top cellview window was not found or an error occurred.

© 2020 All Rights Reserved.

geGetWindowBox

```
geGetWindowBox(
     w_windowId
     )
     => l_bBox / nil
```

Description

Returns the lower left and upper right coordinates of the current window display.

Arguments

w_windowId	Database ID of the window.

Value Returned

l_bBox	The coordinates are found.
nil	The coordinates are not found.

Example

```
geGetWindowBox( window(2) )
```

Returns `(0:0 1200:1000)`.

© 2020 All Rights Reserved.

geGetWindowCellView

```
geGetWindowCellView(
     [ w_windowId ]
     )
     => d_cellview / nil
```

Description

Returns the ID of the cellview displayed in a window.

This function returns `nil` if the current window is empty or if no current window is defined. The cellview being displayed in the window can be different from the cellview being edited, as returned by `geGetEditCellView`, if you are currently editing a cell in place.

You can use the database identifier returned by `geGetWindowCellView` in subsequent function calls to access the properties of the cellview in the current window (as described in chapter two, Database Access. If you know what window you are interested in, a shorthand equivalent to this function can be used: `window(n)->displayCellView`.

Arguments

w_windowId	Database ID of window displaying the cellview whose object identifier you want.

Value Returned

d_cellview	Database ID of the object displayed in the window.
`nil`	The current window is empty or no current window is defined.

Example

```
geGetWindowCellView( hiGetCurrentWindow( ) )
```

Returns `db:123456`, which is the displayed cellview.

© 2020 All Rights Reserved.

geGetWindowRep

```
geGetWindowRep(
     [ w_windowId ]
     )
     => d_cellview / nil
```

Description

An alias for `geGetWindowCellView`. Returns the ID of the cellview displayed in a window.

This function returns `nil` if the current window is empty or if no current window is defined. The cellview being displayed in the window can be different from the cellview being edited, as returned by `geGetEditCellView`, if you are currently editing a cell in place.

You can use the database identifier returned by `geGetWindowCellView` in subsequent function calls to access the properties of the cellview in the current window (as described in chapter two, Database Access. If you know what window you are interested in, a shorthand equivalent to this function can be used: `window(n)->displayCellView`.

Arguments

w_windowId	Database ID of window displaying the cellview whose object identifier you want.

Value Returned

d_cellview	Database ID of the object displayed in the window.
`nil`	The current window is empty or if no current window is defined.

© 2020 All Rights Reserved.

geIsGraphicalWindow

```
geIsGraphicalWindow(
     [ d_windowId ]
     g_noWarn
     )
     => t / nil
```

Description

This function tests if the specified window (or by default, the current window) is a graphical window.

Arguments

d_windowId	Optional argument. The database ID of the window.
g_noWarn	When `t`, no warning is given if the window is a valid graphical window. When `nil`, a warning is given when the window is not valid.

Value Returned

`t`	Testing is set successfully.
`nil`	Returns `nil` when unable to set or determine window.

Example

```
geIsGraphicalWindow()
```

```
geIsGraphicalWindow(window(3) nil)
```


© 2020 All Rights Reserved.

geSaveNetNameDisplayFilter

```
geSaveNetNameDisplayFilter(
     d_cellview
     )
     => t / nil
```

Description

Saves the net name filtering settings that are currently applied on the given cellview after the geSetNetNameDisplayFilter function is called. After you call the `geSaveNetNameDisplayFilter`, the settings will be reloaded automatically when you open the cellview in a new Virtuoso session.

Note: You can use this function only if the cellview is editable.

Arguments

d_cellview	Database ID of the cellview.

Value Returned

`t`	The net name filtering is saved.
`nil`	The net name filtering is not saved.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetNetNameDisplayFilter(cellView "Hide" list("VDD" "VSS"))
geSaveNetNameDisplayFilter(cellView)
```


© 2020 All Rights Reserved.

geSetNetNameDisplayFilter

```
geSetNetNameDisplayFilter(
     d_cellview
     t_mode
     l_netNames
     )
     => t / nil
```

Description

Sets a net name filtering on the given cellview. You can control which nets are displayed or not by the net name display feature.

Arguments

d_cellview	Database ID of the cellview on which to set the filter settings.
t_mode	The mode, `Hide` or `Show`.
l_netNames	The list of net names handled by filtering. If *t_mode* is set to `Show`, the net name display only displays nets in the provided list of net names. If *t_mode* is set to Hide, the net name display shows any nets names except the net names provided in the list of net names.

Value Returned

`t`	The net name filtering is set.
`nil`	The net name filtering is not set.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetNetNameDisplayFilter(cellView "Show" list("VDD" "VSS"))
```


© 2020 All Rights Reserved.

geSetTCProbeDisplayStyle

```
geSetTCProbeDisplayStyle(
     d_probeId
     g_trueColorProbeVal
     g_probeHighlightVal
     )
     => t / nil
```

Description

This function sets the *trueColorProbeVal* and *probeHighlightVal* options of the specified *probeId*.

Arguments

d_probeId	The database ID of the specified probe.
g_trueColorProbeVal	Indicates whether the true color probe will be drawn or not.
g_probeHighlightVal	Indicates whether the probe highlight will be drawn or not.

Value Returned

t	The specified values have been successfully set to the specified *probeId*.
nil	The specified values have not been successfully set to the specified *probeId*.

Example

```
myProbe = geMakeProbeWithColor(?window win ?name "myProbe" ?object '("n1") ?color
list( "hilite"

"drawing7" ) ?probeType "net")

geSetTCProbeDisplayStyle(myProbe t t)
```


© 2020 All Rights Reserved.

geTransparencyOptions

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

Description

(ICADVM20.1) This form opens the Transparency Options form.

Arguments

None

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

```
geTransparencyOptions()
```


© 2020 All Rights Reserved.

geWindowToEditPoint

```
geWindowToEditPoint(
     w_windowId
     l_windowCellViewPoint
     )
     => l_editCellViewPoint
```

Description

Translates coordinate values when you are using edit in place. `geWindowToEditPoint` applies the top-level cellview's coordinates to the cellview being edited; `geEditToWindowPoint` does the reverse.

All SKILL functions apply to the cellview being edited in the current window. When you are editing in place, this cellview is not the same one as the top-level cellview being displayed in the window. Coordinate values passed to SKILL functions are always expressed with respect to the cellview being displayed, even when you are editing in place. Database access inquiries, however, always return coordinate values in terms of the coordinate system of the cellview being interrogated, regardless of the context that cellview is used in.

The role of `geWindowToEditPoint` and `geEditToWindowPoint` is to translate between the two coordinate systems.

`geWindowToEditPoint` expects an argument consisting of a coordinate pair list. It interprets the coordinates with respect to the coordinate system of the current *edit-cellview* and returns the corresponding list, taking all intermediate locations and transformations into account.

`geWindowToEditPoint` function performs the translation in the opposite direction. Given a coordinate pair list, it interprets the coordinates with respect to the *window-cellview* and returns the corresponding list expressed in terms of the *edit-cellview*'s coordinate system. You can use this function, for example, to determine where an object placed at a given location in the *window-cellview* is situated in the *edit-cellview*.

© 2020 All Rights Reserved.

Arguments

w_windowId Database ID of the window containing the cellview being edited.

l_windowCellviewPoint

Coordinate pair expressed in window-cellview units (`geWindowToEditPoint`).

Value Returned

l_editCellViewPoint

Coordinate pair expressed in edit-cellview units (`geWindowToEditPoint`).

Example

Assume you are editing in place an instance placed at 12.5:10.25.

```
geWindowToEditPoint(window(4) 12.5:10.25)
```

Returns `0:0`, because this is the origin of the instance.

```
geEditToWindowPoint( window(4) 0:0 )
```

Returns `12.5:10.25`.

© 2020 All Rights Reserved.

geEnableNetNameDisplay

```
geEnableNetNameDisplay(
     w_windowId
     g_state
     )
     => t / nil
```

Description

Enables the display of net name labels on shapes that are big enough to display a label.This SKILL function is supported only in Layout XL and higher tiers.

Arguments

w_windowId	ID of the window for which net name labels need to be displayed on shapes.
g_state	Enables or disables the display of net name labels for a specified window ID.

Value Returned

`t`	Net name label display for shapes in the specified window is enabled.
`nil`	Net name label display is disabled.

Example

```
window = hiGetCurrentWindow()
geEnableNetNameDisplay(window t)
```

Enables the net name label display for shapes in the specified window ID.

```
geEnableNetNameDisplay(window nil)
```

Disables the net name label display for shapes in the specified window ID.

Related Topics

Enabling Net Name Display

Net Name Display Options Form

© 2020 All Rights Reserved.

geNetNameDisplayOptionForm

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

Description

Opens the Net Name Display Options form. This SKILL function is supported only in Layout XL and higher tiers.

Arguments

w_windowId	ID of the window in which the Net Name Display Options form needs to be opened.

Value Returned

t	The Net Name Display Options form was opened in the specified window ID.
nil	The Net Name Display Options form could not be opened.

Related Topics

Enabling Net Name Display

© 2020 All Rights Reserved.

Refresh Functions

geRefresh

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

Description

Redraws all areas of all windows that have not been updated since the last redraw and need redrawing.

Arguments

None

Value Returned

`t`	The windows are redrawn.
`nil`	The windows are not redrawn.

Example

```
geRefresh( )
```

Redraws all areas of all windows that need updating.

© 2020 All Rights Reserved.

geRefreshCellView

```
geRefreshCellView(
     d_cvId
     l_Bbox
     )
     => t / nil
```

Description

Redraws a cellview when necessary. If a bounding box is specified, only the specified area is redrawn.

Arguments

d_cvId	ID of the cellview to be redrawn.
l_Bbox	List of points that specify an area to be redrawn.

Value Returned

`t`	Cellview or area is successfully refreshed.
`nil`	Cellview or area is not refreshed.

© 2020 All Rights Reserved.

geRefreshCellViews

```
geRefreshCellViews(
     d_cvId
     )
     => t / nil
```

Description

Displays a form that shows all cellviews from the specified cellview that have been modified by another process but not reread from disk. You can then select which ones should be refreshed. The top-level cellview is also checked for modification.

Arguments

d_cvId	Cellview ID of the cellview to be redrawn.

Value Returned

`t`	Cellview is successfully refreshed.
`nil`	Cellview is not refreshed.

Example

```
geRefreshCellViews( cell1 )
```

Redraws `cell1`.

© 2020 All Rights Reserved.

geRefreshWindow

```
geRefreshWindow(
     [ w_windowId ]
     [ l_bBox ]
     )
     => t / nil
```

Description

Refreshes a window. If a bounding box is specified, only that area is refreshed.

Arguments

w_windowId	Database ID of the window to be refreshed. If not specified, the current window is used.
l_bBox	Specifies the area to be refreshed.

Value Returned

`t`	The window is refreshed successfully;
`nil`	The window is not refreshed successfully.

Example

```
geRefreshWindow( win2 )
```

Redraws `win2`.

© 2020 All Rights Reserved.

geRefreshWorldView

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

Description

Refreshes the worldView window.

Arguments

w_windowId	The worldView window ID.

Value Returned

`t`	The window is refreshed.
`nil`	The window is not refreshed.

© 2020 All Rights Reserved.

View Level Functions

geDeleteAllAreaViewLevel

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

Description

Removes all area view levels from a window.

Arguments

w_windowId	Database ID of the window where you want to delete area view levels. If *w_windowId* is not specified, the current window is used.

Value Returned

`t`	Area view levels were deleted.
`nil`	No area view levels were deleted.

Example

```
geDeleteAllAreaViewLevel( window(3) )
```

Deletes all area view levels in `window(3)`.

© 2020 All Rights Reserved.

geDeleteAllInstViewLevel

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

Description

Removes all instance-specific view levels from a window.

Arguments

w_windowId	Database ID of the window from which you want to remove the instance view levels.

Value Returned

`t`	View levels are found and deleted.
`nil`	No view levels are found.

Example

```
geDeleteAllInstViewLevel( )
```

Deletes all the instance view levels in the current window.

© 2020 All Rights Reserved.

geDeleteAreaViewLevel

```
geDeleteAreaViewLevel(
     [ w_windowId ]
     l_bBox
     )
     => t / nil
```

Description

Removes from a window all area view levels located in *l_bBox*.

Arguments

w_windowId	Database ID of the window where you want to delete area view levels. If *w_windowId* is not specified, the current window is used.
l_bBox	Bounding box containing the view levels to delete.

Value Returned

`t`	An area view level is deleted.
`nil`	No area view level is deleted.

Example

```
geDeleteAreaViewLevel( window(3) list(0:0 400:400) )
```

Deletes the area view levels located in the bounding box specified by the coordinates 0:0 400:400.

© 2020 All Rights Reserved.

geDeleteInstViewLevel

```
geDeleteInstViewLevel(
     [ w_windowId ]
     d_instId
     )
     => t / nil
```

Description

Removes all instance-specific view levels associated with an instance ID.

Arguments

w_windowId	Database ID of the window from which you want the instance view levels removed. If *w_windowId* is not specified, the current window is use otherwise, it returned.
d_instId	Database ID of the instance that has its view levels removed.

Value Returned

`t`	A view level was found and deleted.
`nil`	A view level was not found and deleted.

Example

```
geDeleteInstViewLevel( instance4 )
```

Deletes the view levels associated with `instance4`. The instance is then displayed using the window start level and stop level.

© 2020 All Rights Reserved.

geGetAllAreaViewLevel

```
geGetAllAreaViewLevel(
     w_windowId
     )
     => l_bBoxStartStop / nil
```

Description

Retrieves all area view levels set for a window. The view levels are returned as a list with the start level and stop level for the areas.

Arguments

w_windowId	Database ID of the window for which you want the view levels.

Value Returned

l_bBoxStartStop	A list consisting of the bounding box for the area and the start level and stop level.
nil	No areas were found.

Example

```
geGetAllAreaViewLevel( window(3) )
```

Returns a list of the bounding boxes and start levels and stop levels for all area views in `window(3)`.

© 2020 All Rights Reserved.

geGetAllInstViewLevel

```
geGetAllInstViewLevel(
     w_windowId
     )
     => l_instStartStop / nil
```

Description

Retrieves the start level and stop level assigned within a window. The start level and stop level are returned as a list of integers. If no start level and stop level are defined, then `nil` is returned.

Arguments

`w_windowId`	Database ID of the window where you want the view levels.

Value Returned

`l_instStartStop`	If the instances in *`w_windowId`* have view levels associated with them, then a list with each instance and the start level and stop level of each instance view level is returned.
`nil`	There are no view levels available.

Example

```
geGetAllInstViewLevel( )
```

Lists all instances associated with a view level and the start level and stop level for each instances in the current window.

© 2020 All Rights Reserved.

geGetAreaViewLevel

```
geGetAreaViewLevel(
     [ w_windowId ]
     l_bBox
     )
     => l_startStop / nil
```

Description

Retrieves an area view level overlapping a bounding box. The view level is returned as a list with the start level and stop level for that area. If more than one area overlaps the given box, the first area found is used.

Arguments

w_windowId	Database ID of the window in which you want the view levels for an area. If *w_windowId* is not specified, the current window is used.
l_bBox	Bounding box in which to check for an area view level.

Value Returned

l_startStop	A list consisting of the start level and stop level for the first area encountered that is located in *l_bBox*.
nil	No area view level was found located in *l_bBox*.

Example

```
geGetAreaViewLevel( window(3) list(0:0 200:200) )
```

Returns `(0 20)`, the start level and stop level for the area found in the bounding box 0:0 200:200 in `window(3)`.

© 2020 All Rights Reserved.

geGetInstViewLevel

```
geGetInstViewLevel(
     w_windowId
     d_instId
     )
     => l_startStop / nil
```

Description

Retrieves the start level and stop level of an instance. The start level and stop level are returned as a list of integers. If start level and stop level are not defined for the chosen instance, `nil` is returned.

Arguments

w_windowId	Database ID of the window where you want the view levels for *d_instId*.
d_instId	Database ID of the instance for which you want the view levels.

Value Returned

l_startStop	If *d_instId* has view levels associated with it in *w_windowId*, then a list with the start level and stop level is returned.
`nil`	*d_instId* has no view levels associated with it in *w_windowId*.

Example

```
geGetInstViewLevel( )
```

Assuming that the instance has start level 0 and stop level 20, it returns `(0 20)`.

© 2020 All Rights Reserved.

geSetAreaViewLevel

```
geSetAreaViewLevel(
     [ w_windowId ]
     l_bBox
     [ n_start ]
     [ n_stop ]
     )
     => t / nil
```

Description

Sets display start and stop threshold levels for an area independently of the window levels.

The start level and stop level specified apply only to *l_bBox* and only in *w_windowId*. If *n_start* or *n_stop* is not specified, a form comes up asking you to supply values for these arguments.

Arguments

w_windowId	Database ID of the window in which you want the area view levels changed. If *w_windowId* is not specified, the current window is used.
l_bBox	Box that is to have its own view level.
n_start	Starting hierarchy level for display in *l_bBox*. Must be less than or equal to *n_stop*. Valid Values: between 0 and 20 inclusive.
n_stop	Stopping hierarchy level for display in *l_bBox*. Valid Values: between 0 and 20 inclusive.

Value Returned

`t`	The start level and stop level are set
`nil`	The start level and stop level are not set.

Example

```
geSetAreaViewLevel( window(3) list(0:0 120:120) 0 10 )
```


© 2020 All Rights Reserved.

Sets the display of the area bounded by the coordinates 0:0 and 120:120 to view 10 levels of hierarchy in `window(3)`.

© 2020 All Rights Reserved.

geSetInstViewLevel

```
geSetInstViewLevel(
     w_windowId
     d_instId
     n_start
     n_stop
     )
     => t / nil
```

Description

Sets display start and stop threshold levels for an instance independently of the window levels.

The start level and stop level specified apply only to *d_instId* and only in *w_windowId*.

Arguments

w_windowId	Database ID of the window in which you want the area view levels changed.
d_instId	Database ID of the instance that is to have its own view level.
n_start	Starting hierarchy level for display in *l_bBox*. Must be less than or equal to *n_stop*. Valid Values: between 0 and 20 inclusive.
n_stop	Stopping hierarchy level for display in *l_bBox*. Valid Values: between 0 and 20 inclusive.

Value Returned

`t`	The start level and stop level are set
`nil`	The start level and stop level are not set.

Example

```
geSetInstViewLevel(
  hiGetCurrentWindow()
  dbFindAnyInstByName(geGetEditCellView() "|I11|MP0.3")
  0 1
)
```


© 2020 All Rights Reserved.

Sets the display of the instance named "I11|MP0.3" to view levels from 0 to 1 of hierarchy.

© 2020 All Rights Reserved.

Viewport Functions

geRestoreWinView

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

Description

Restores the view saved with `geSaveWinView`.

Arguments

w_windowId	Database ID of the window to restore.

Value Returned

`t`	The window is restored.
`nil`	The window is not restored.

Example

```
geRestoreWinView( newwin )
```

Restores the window `newwin`.

© 2020 All Rights Reserved.

geSaveWinView

```
geSaveWinView(
     [ w_windowId ]
     [ t_name ]
     )
     => t / nil
```

Description

Saves the current view of *w_windowId* to *t_name*.

Arguments

w_windowId	Database ID of the window containing the view to save. If *w_windowId* is not specified, the current window is used.
t_name	Name of the saved window view. If *t_name* is not specified, a form is displayed where you can enter the name.

Value Returned

`t`	The window view is saved.
`nil`	The window view is not saved.

Example

```
geSaveWinView( win3 "newwin" )
```

Saves `win3` to `newwin`.

© 2020 All Rights Reserved.

Procedural Level Functions

geEditInPlace

```
geEditInPlace(
     w_windowId
     t_accessMode
     d_instId
     x_iteration
     x_row
     x_col
     [ g_cancel ]
     )
     => t / nil
```

Description

Edits an instance in place.

w_windowId must be a valid Graphics Editor window with at least one stack element, and *d_instId* must be an instance in the edit cellview in *w_windowId*.

- If *d_instId* is an iterated instance, *x_iteration* is the particular iteration that is edited in place.
- If *d_instId* is an arrayed instance, *x_row* and *x_col* indicate the particular element that is edited in place.

© 2020 All Rights Reserved.

Arguments

w_windowId	Database ID of the window in which to do the edit in place.
t_accessMode	Access mode used to open the instance master. Valid Values: `r` (read) or `a` (append)
d_instId	Database ID of the instance to edit in place.
x_iteration	Desired iteration of the given instance. Default: `0`
x_row	Row of the arrayed instance.
x_col	Column of the arrayed instance.
g_cancel	This optional argument is currently ignored, and will be removed in a future release.

Value Returned

`t`	The instance is edited in place.
`nil`	The instance is not edited in place.

Example

```
geEditInPlace( win3 "a" inst5 1 3 3 )
```

Opens the master of `inst5` in `win3` in edit mode before editing it in place. It opens iteration 1 in row 3, column 3.

© 2020 All Rights Reserved.

gePush

```
gePush(
     w_windowId
     t_accessMode
     d_instId
     x_iteration
     x_row
     x_col
     [ g_cancel ]
     )
     => t / nil
```

Description

Pushes into an instance.

w_windowId must be a valid Graphics Editor window with at least one stack element, and *d_instId* must be an instance in the edit cellview in *w_windowId*.

- If *d_instId* is an iterated instance, *x_iteration* is the particular iteration that is pushed.
- If *d_instId* is an arrayed instance, *x_row* and *x_col* indicate the particular element that is pushed.

© 2020 All Rights Reserved.

Arguments

w_windowId	Database ID of the window in which to do the push.
t_accessMode	Access mode used to open the instance master. Valid Values: `r` (read) or `a` (append)
d_instId	Database ID of the instance to push into.
x_iteration	Desired iteration of the given instance. Default: `0`
x_row	Row of the arrayed instance.
x_col	Column of the arrayed instance.
g_cancel	This optional argument is currently ignored, and will be removed in a future release.

Value Returned

`t`	The instance is pushed into.
`nil`	The instance is not pushed into.

Example

```
gePush( win3 "a" inst5 1 3 3 )
```

Opens the master of `inst5` in `win3` in edit mode before pushing it on the stack. Opens iteration 1 of the instance in row 3, column 3.

© 2020 All Rights Reserved.

geSwitch

```
geSwitch(
     w_windowId
     t_accessMode
     d_instId
     x_iteration
     x_row
     x_col
     [ g_cancel ]
     )
     => w_windowId / nil
```

Description

Switches into an instance.

w_windowId->viewNameList is used as a switch for the instance. *w_windowId* must be a valid Graphics Editor window with at least one stack element, and *d_instId* must be an instance in the edit cellview in *w_windowId*.

- If *d_instId* is an iterated instance, *x_iteration* is the particular iteration that is switched to.
- If *d_instId* is an arrayed instance, *x_row* and *x_col* indicate the particular element that is switched to.

© 2020 All Rights Reserved.

Arguments

w_windowId	Database ID of the window in which to do the switch.
t_accessMode	Access mode used to open the instance master.Valid Values: `r` (read) or `a` (append)
d_instId	Database ID of the instance to switch into.
x_iteration	Offset in the array of the instance (not the bit number). For example: ■ iteration 1 of I1<0:5> descends into I1<1> ■ iteration 2 of I1<1:6> descends into I1<3> ■ iteration 3 of I1<2:7> descends into I1<5> ■ iteration 4 of I1<3:8> descends into I1<7> ■ iteration 4 of I1<5:0> descends into I1<4> Default: `0`
x_row	Row of the arrayed instance.
x_col	Column of the arrayed instance.
g_cancel	This optional argument is currently ignored, and will be removed in a future release.

Value Returned

w_windowId	The database ID of the window that is switched into.
`nil`	The instance in not switched into.

Example

```
geSwitch( win3 "a" inst5 2 3 3 )
```

Opens the master of `inst5` in `win3` in edit mode and switches it to iteration 2 of the instance in row 3, column 3.

© 2020 All Rights Reserved.

geSwitchInContext

```
geSwitchInContext(
     w_windowId
     t_accessMode
     d_instId
     x_iteration
     x_row
     x_col
     [ verbose ]
     )
     => w_windowId / nil
```

Description

Switches into an instance by using the view name specified in the hierarchy manager configuration context. If the correct configuration context is not found, this function uses the geSwitch command to perform the switch.

Arguments

w_windowId	Database ID of the window in which to do the switch.
t_accessMode	Access mode used to open the instance master.Valid Values: `r` (read) and `a` (append)
d_instId	Database ID of the instance to switch into.
x_iteration	Offset in the array of the instance (not the bit number). For example: iteration 1 of I1<0:5> descends into I1<1> iteration 2 of I1<1:6> descends into I1<3> iteration 3 of I1<2:7> descends into I1<5> iteration 4 of I1<3:8> descends into I1<7> iteration 4 of I1<5:0> descends into I1<4>
x_row	Row of the arrayed instance.
x_col	Column of the arrayed instance.
verbose	This optional argument diplays a warning message when something is incorrect.

© 2020 All Rights Reserved.

Value Returned

w_windowId	The database ID of the window that is switched into.
nil	The instance in not switched into.

Example

In a valid configuration context, specify a view name to be used for an instance (`instId`) in hierarchy manager configuration, set the correct window as the current window and call the function below:

```
geSwitchInContext(hiGetCurrentWindow() "r" instId 0 0 0)
```


© 2020 All Rights Reserved.

Window Scrolling Functions

geCenter

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

Description

Pans the display so that the edit cellview is centered in the window.

Arguments

w_windowId	Database ID of the window in which to center the design. If *w_windowId* is not specified or `nil`, the current window is used.

Value Returned

`t`	The display is panned.
`nil`	The display is not panned.

Example

```
geCenter( window(3) )
```

Centers the design in `window(3)`.

© 2020 All Rights Reserved.

geGetScrollPercent

```
geGetScrollPercent(
     )
     => x_percent
```

Description

Obtains the percentage used by `geScroll` when shifting the drawing.

Arguments

None

Value Returned

x_percent	Percentage of the display to scroll.

Example

```
geGetScrollPercent( )
```

Returns the scroll percentage.

© 2020 All Rights Reserved.

geScroll

```
geScroll(
     [ w_windowId ]
     [ t_dir ]
     [ g_zoom ]
     )
     => t / nil
```

Description

Scrolls a view.

This percentage can be set and read using `geSetScrollPercent` and `geGetScrollPercent`.

Arguments

w_windowId	Database ID of the window in which to pan the design. If not specified, the current window is used.
t_dir	The map direction of the scroll. Valid Values: `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw`.
g_zoom	If `t`, pan all the way to the edge in the given direction. If `nil`, pan by the current scroll percentage. See `geSetScrollPercent`.

Value Returned

`t`	The display is panned.
`nil`	The display is not panned.

Example

```
geScroll( window(3) "ne" t )
```

Pans the design in `window(3)` so that the upper right hand corner is visible.

© 2020 All Rights Reserved.

geSetScrollPercent

```
geSetScrollPercent(
     n_percent
     )
     => n_percent
```

Description

Controls the percentage of the drawing currently visible that is shifted when `geScroll` is activated.

The value can be set between 1 percent and 100 percent.

Arguments

n_percent	New percentage of the display to scroll.

Value Returned

n_percent	New percentage of the display to scroll.

Example

```
geSetScrollPercent( 25 )
```

Returns `25` and sets the new percentage to 25 percent.

© 2020 All Rights Reserved.

geShift

```
geShift(
     [ w_windowId ]
     [ t_dir ]
     )
     => t / nil
```

Description

Shifts *w_windowId* in the direction *t_dir*.

`geshift()` is a wrapper function written on top of `geScroll()` in order to facilitate a slightly different use mode. `geshift()` uses the same mechanism as `geScroll()`, but gives you access to `e`, `w`, `n`, `s` directions in the form of `right`, `left`, `up`, and `down`.

Arguments

w_windowId	Database ID of the window to be shifted.
t_dir	Direction to shift the window. Valid Values: `right`, `left`, `up`, `down`

Value Returned

`t`	The window shifts.
`nil`	The window does not shift.

Example

```
geShift( win2 "up" )
```

Shifts the window `win2` one screen dimension in the *upward* direction.

© 2020 All Rights Reserved.

geZoomToGrid

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

Description

Zooms in on a subject up to the last point at which the major grid is still visible.

Arguments

w_windowId	Database ID of the window that displays the subject.

Value Returned

`t`	The subject is displayed successfully.
`nil`	The subject is not displayed successfully.

© 2020 All Rights Reserved.

Selection Functions

With the selection functions you can

- Point at specific objects to be selected
- Select all objects within a specified area
- Select all objects in a specified cellview

You can select objects individually (point selection) or in groups (area selection) and add them to other selected groups of objects. You can subtract objects from a selected set (sub mode) or define certain objects to be unselectable (selection filter).

Selected Set

A group of objects selected together, either by point selection or area selection, is a selected set.

Point Selection

Point selection is the most commonly used method of selection. The modes for point selection are

- Single (selects one object at a time)
- Add (adds one object at a time to the selected set)
- Sub (subtracts one object at a time from the selected set)

Each of these modes can be used to select entire objects or vertices of objects. All modes of point selection affect only one object at a time. The object selected is the object with the edge closest to the point of selection that also passes the selection filter. You can select edges and lines that are off-grid as long as the point of selection is on the closest snap grid location.

Selecting the vertices of objects is referred to as *partial* selection. For partial selection,

- Only the closest edge to the selection point is selected
- If the selection point is a vertex, only the vertex is selected
- Instances, mosaics, dots, and labels cannot be partially selected
- Selecting the vertices of this type of object selects the entire object

© 2020 All Rights Reserved.

Single Mode Point Selection

The *single* mode of selection is useful when you want to select a specific object. It selects the specified object (edge or vertex for partial selection) and deselects all previously selected objects. All previously selected objects are deselected even if they would not pass the selection filter.

Add Mode Point Selection

The *add* mode of selection is identical to the *single* mode, except that currently selected objects are not deselected. The *add* mode is useful when you are trying to select a group of specific objects.

Sub Mode Point Selection

The *sub* mode of selection is identical to the *add* mode, except that the specified object (edge or vertex for partial selection) is deselected. The *sub* mode is useful when you have selected a group of objects and you want to remove one of them.

Interactive Function

When a selection point (`l_point`) is specified as an argument in any point selection function, SKILL uses that point to evaluate the function and select the indicated object. If you do not specify a value for `l_point`, the function behaves interactively and prompts you to enter a selection point.

Area Selection

Like point selection, area selection supports three modes:

- Single (selects objects in one area at a time)
- Add (adds objects in an area to the selected set)
- Sub (subtracts objects in an area from the selected set)

Area selection is most useful when you want to select multiple objects at the same time. All objects that are completely enclosed within the selection area and also pass the selection filter are selected. Area selection can also be used to partially select shapes. If partial selection is enabled, all vertices of overlapped shapes are selected.

© 2020 All Rights Reserved.

Single Mode Area Selection

The *single* mode of selection is useful when you want to select a specific group of objects. It selects all completely overlapped objects (plus all overlapped vertices for partial selection) that pass the selection filter, and deselects all previously selected objects. The de-selection operation deselects all previously selected objects even if they would not pass the selection filter.

Add Mode Area Selection

The *add* mode of selection is identical to the *single* mode, except that currently selected objects remain selected. The *add* mode is useful when you are trying to select a group of specific objects.

Sub Mode Area Selection

The *sub* mode of selection is identical to the *add* mode, except that all completely overlapped objects (plus all overlapped vertices for partial selection) are deselected. The *sub* mode is useful when you want to remove specified objects from a selected set.

Interactive Function

When a selection area (`l_bBox`) is specified as an argument in any area selection function, SKILL uses the coordinates for that area to evaluate the function and select the enclosed objects. If you do not specify a value for `l_bBox`, the function behaves interactively and prompts you to enter coordinates for a selection area.

User Interface

Both selection by point and selection by area are supported directly using standard mouse bindings. Selection by point is accomplished by clicking the left mouse button, and selection by area is accomplished by drawing a selection rectangle with the left mouse button. The three modes of selection are supported using modifier keys. These bindings are described in the table.

Standard Selection Mouse Bindings

Modifier	Left mouse click	Left mouse drag
None	Single-Select Point	Single-Select Box

© 2020 All Rights Reserved.

Standard Selection Mouse Bindings

Modifier	Left mouse click	Left mouse drag
Shift	Add-Select Point	Add-Select Box
Ctrl	Sub-Select Point	Sub-Select Box

Partial selection is controlled via function key `F4`. This key toggles partial selection on and off. Both partial point selection and partial area selection are affected. This key binding toggles the window environment variable `partialSelect`.

© 2020 All Rights Reserved.

SKILL-Level Access

The following sections describe the functions available at the SKILL level for creating, maintaining, and accessing selected sets. There are four categories of selection functions.

Point Selection

These functions implement the interactive selection by point. These functions can be interactive if the point information is not passed as an argument.

```
geSingleSelectPoint
geAddSelectPoint
geSubSelectPoint
```

Area Selection

These functions implement the functions for interactive selection by area. These functions can be interactive if the area (bBox) information is not passed as an argument.

```
geSingleSelectBox
geAddSelectBox
geSubSelectBox
```

Figure Selection

These functions allow you to select and deselect objects and portions of objects. These functions are separate from the point and area selection functions and do not use any of the selection modes.

```
geSelectFig
geSelectAllFig
geDeselectAllFig
geDeselectArea
geDeselectFigNoFilter
geDeselectObject
geDeselectObjectPoint
geIsFigSelected
geIsFigPartiallySelected
geGetSelectedSet
geGetSelSetCount
geSelectFigPoint
geDeselectAll
geDeselectAllObject
geDeselectFig
geDeselectFigPoint
geDeselectObjectNoFilter
geDeselectPoint
geIsFigAllSelected
geGetSelectedSetFigPoint
geGetSelSet
```

Selection Queries

These functions allow you to query the contents of the selected set.

```
geIsFigSelected
geIsFigAllSelected
geIsFigPartiallySelected
gePointQuery
```


© 2020 All Rights Reserved.

geAdvWireSelection

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

Description

Allows the user to select route elements and the adjacent routes. Its behavior is based on the initial state of the selection and the number of consecutive calls to `geAdvWireSelection()`.

`geAdvWireSelection()` follows a cycle mechanism in three steps:
The two first calls allow to expand the routes elements selection.
The third one restarts the selection to the initial selection.
The behavior of first step depends of the initial route element selected.

The Advanced Partial Selection is a feature of the select mechanism only available on symbolic Routes object. This feature is activated through `geAdvWireSelection()`.

Arguments

w_windowId	Database ID of the window.

Value Returned

`t`	The selection is performed.
`nil`	The initial conditions are not correct.

© 2020 All Rights Reserved.

geAddSelectBox

```
geAddSelectBox(
     [ w_windowId ]
     [ g_partial ]
     [ l_bBox ]
     )
     => t / nil
```

Description

Implements the `add` mode of area selection. With no arguments given, it prompts you to enter the area to be selected in the current window. Partial selection is performed only when the window environment variable `partialSelect` is set and the `stretch` command is active. Partial selection lets you select an edge, vertex, or endpoint of an object to stretch.

Arguments

w_windowId	Database ID of the window containing the objects.
g_partial	Partial selection is used only in conjunction with the `stretch` command. In this case, this function indicates whether partial selection (`t`) or full selection (`nil`) should be performed. If you use partial selection with the `move` command, the selection mode is always set to full selection and cannot be overwritten. Valid Values: `t`, `nil`
l_bBox	The bBox to be used for selection. When no bBox is specified, this function operates interactively.

Value Returned

`t`	The selection is performed.
`nil`	The selection is not performed.

Example

```
geAddSelectBox( win1 nil list(0:0 25:25) )
```

Selects the objects in `win1` enclosed in the bBox described by the coordinates 0:0, 25:25 and adds them to the selected set. Partial selection is not performed.

© 2020 All Rights Reserved.

geAddSelectPoint

```
geAddSelectPoint(
     [ w_windowId ]
     [ g_partial ]
     [ l_point ]
     )
     => t / nil
```

Description

Implements the `add` mode of point selection.

With no arguments, it prompts you to point to the object to be selected in the current window. Partial selection is performed if the window environment variable `partialSelect` is set.

Arguments

w_windowId	Database ID of the window containing the object.
g_partial	Whether partial selection should be performed. Valid Values: `t`, `nil`
l_point	List of points to be used for selection. When no points are specified, this function operates interactively.

Value Returned

`t`	The selection is performed.
`nil`	The selection is not performed.

Example

```
geAddSelectPoint( win3 nil list(0:0 10:10) )
```

Selects the objects in `win3` located at the coordinates 0:0 and 10:10. Partial selection is not performed.

© 2020 All Rights Reserved.

geAdjustPoint

```
geAdjustPoint(
     l_point
     [ w_windowId ]
     )
     => l_point / nil
```

Description

Returns the point after applying the adjustment associated to the window Id provided. Adjustment is defined using gravity, the current snap grid, and any snap mode defined in the window interface.

Arguments

l_point	The point to be adjusted.
w_windowId	Database ID of the window.

Value Returned

l_point	The point that has been adjusted.
nil	The point was not adjusted.

© 2020 All Rights Reserved.

geCyclePotentialSelectNext

```
geCyclePotentialSelectNext(
     [ w_windowID ]
     [ l_point ]
     )
     => t / nil
```

Description

Cycles the potential selection to the next object.

Arguments

w_windowID	Optional argument that specifies the window to be used. The current window is used if *w_windowId* is not specified.
l_point	Optional argument that specifies the point to be potentially selected. The current point is used if *l_point* is not specified.

Value Returned

t	Next object is potential selected.
nil	Next object is not potential selected.

Example

```
win=hiGetCurrentWindow()
dbCreateRect(win~>cellView list("M2" "drawing") list(-5:-5 5:5))
dbCreateRect(win~>cellView list("M3" "drawing") list(-5:-4 3:4))
dbCreateRect(win~>cellView list("M5" "drawing") list(-5:-3 1:3))
dbCreateRect(win~>cellView list("M6" "drawing") list(-5:-2 1:2))

geSingleSelectPoint(win t 0:0)
geCyclePotentialSelectNext(win 0:0)
```


© 2020 All Rights Reserved.

geCycleSelectNext

```
geCycleSelectNext(
     [ w_windowID ]
     )
     => t / nil
```

Description

Cycles to the next object at the current selection point.

Arguments

w_windowID	Optional argument that specifies the window to be used.

Value Returned

`t`	Next object is selected.
`nil`	No other objects are selected.

© 2020 All Rights Reserved.

geDeselectAll

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

Description

Deselects everything in the specified window. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Database ID of the window containing the object.

Value Returned

t	Everything is deselected.
nil	Everything is not deselected.

© 2020 All Rights Reserved.

geDeselectAllFig

```
geDeselectAllFig(
     [ d_cellview ]
     )
     => t / nil
```

Description

Deselects all objects in a cellview.

Does not use the selection filter, so all objects are deselected.

Arguments

d_cellview	Database ID of the cellview containing the objects you want to deselect. If no cellview is specified, the current cellview is used.

Value Returned

t	The objects are deselected.
nil	The objects are not deselected.

© 2020 All Rights Reserved.

geDeselectAllObject

```
geDeselectAllObject(
     [ d_cellview ]
     )
     => t / nil
```

Description

Deselects everything in the specified cellview.

Arguments

d_cellview	Database ID of the cellview containing the objects you want to deselect. If no cellview is specified, the current cellview is used.

Value Returned

t	The objects are deselected.
nil	The objects are not deselected.

© 2020 All Rights Reserved.

geDeselectArea

```
geDeselectArea(
     [ w_windowId ]
     [ l_bBox ]
     )
     => t / nil
```

Description

Calls `geSubSelectBox` to deselect objects within the specified bounding box in the specified window.

Arguments

w_windowId	Window ID of the window to use. If not specified, the current window is used.
l_bBox	The area to deselect.

Value Returned

`t`	The objects within the bounding box were deselected.
`nil`	The objects were not deselected.

Example

```
geDeselectArea(hiGetCurrentWindow() list(list(0 0) list(100 100)))
```

Deselects objects within the specified area and returns `t`.

Interactive Function

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

Enter this function with only the window ID argument; the system prompts you to point to the area to deselect and returns `t`. If *w_windowId* is not specified, the current window is used.

© 2020 All Rights Reserved.

geDeselectFig

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

Description

Removes an object from the selected set if the object passes the selection filter.

Arguments

d_figId	Database ID of the object you want to deselect.

Value Returned

`t`	The object passes the selection filter and is deselected.
`nil`	The object does not pass the selection filter and is not.

Example

```
geDeselectFig( fig3 )
```

Deselects `fig3`.

© 2020 All Rights Reserved.

geDeselectFigs

```
geDeselectFigs(
     l_figList
     )
     => l_list / nil
```

Description

Deselects figures from a specified list of figures.

Arguments

l_figList	List of figure IDs.

Value Returned

l_list	List of figures to be deselected.
nil	An error occurred, the figures are not selected and the list of figures is not returned.

Example

```
geSelectAllFig()
sel = geGetSelSet()
geDeselectFigs(sel)
```


© 2020 All Rights Reserved.

geDeselectFigNoFilter

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

Description

Deselects the specified figure.

Arguments

d_figId	The figure to be deselected.

Value Returned

`t`	The figure is deselected.
`nil`	The figure is not deselected.

Example

```
geDeselectFigNoFilter( figId )
```

Deselects `figId`.

© 2020 All Rights Reserved.

geDeselectFigPoint

```
geDeselectFigPoint(
     d_figId
     l_vertex
     )
     => t / nil
```

Description

Removes vertices from the selected set if they pass the selection filter.

For vertex de-selection, rectangles are considered to have four vertices, starting with the lower left point and moving counterclockwise.

Instances, dots, and labels cannot be partially selected, so this function cannot be used on these types of objects.

Arguments

d_figId	Database ID of the object you want to select.
l_vertex	A list of Boolean values corresponding to the list of points for the object. Each Boolean specifies whether the corresponding vertex of the object should be deselected or not. If there are fewer points in *l_vertex* than in the object, they are applied to the first vertices in the object and the remaining vertices are unaffected.Valid Values: `t`, `nil`

Value Returned

`t`	The object passes the selection filter and the vertices are deselected.
`nil`	The function failed.

Example

```
geDeselectFigPoint( fig3 list(t nil t nil t nil) )
```

Deselects every other vertex of `fig3`.

© 2020 All Rights Reserved.

geDeselectObject

```
geDeselectObject(
     d_figId
     [ l_path ]
     )
     => t / nil
```

Description

Deselects the specified object at the specified hierarchical path, if it passes the selection filter.

Arguments

d_figId	The figure to be deselected.

l_path	Hierarchical path to *d_figId*, in the following format:
	`list(list(`*instId memInst x y*`)`
	`list(`*instId memInst x y*`)`
	.
	.
	.
	`)`
	x and *y* are integers defining the row and column in case *instId* is a mosaic.

Value Returned

`t`	The specified figures are deselected.
`nil`	The specified figures are not deselected.

Example

```
geDeselectobject( figId list((list(instId 7 0 0 )))
```

If the seventh member of `instId` passes the selection filter, deselects the object.

© 2020 All Rights Reserved.

geDeselectObjectNoFilter

```
geDeselectObjectNoFilter(
     d_figId
     [ l_path ]
     )
     => t / nil
```

Description

Deselects the specified object at the specified hierarchical path.

Arguments

d_figId	The figure to be deselected.

l_path	Hierarchical path to *d_figId*, in the following format:
	`list(list(`*instId memInst x y*`)`
	`list(`*instId memInst x y*`)`
	.
	.
	.
	`)`
	x and *y* are integers defining the row and column in case *instId* is a mosaic.

Value Returned

`t`	The specified figure is deselected.
`nil`	The specified figure is not deselected.

Example

```
geDeselectobject( figId list((list(instId 7 0 0 )))
```

Deselects the seventh member of the specified `instId`.

© 2020 All Rights Reserved.

geDeselectObjectPoint

```
geDeselectObjectPoint(
     d_figId
     l_path
     l_points
     )
     => t / nil
```

Description

Deselect the specified vertices of the specified figure if it passes the selection filter.

Arguments

d_figId	The figure to be deselected.

l_path	Hierarchical path to *d_figId*, in the following format:
	`list(list(`*instId memInst x y*`)`
	`list(`*instId memInst x y*`)`
	.
	.
	.
	`)`
	x and *y* are integers defining the row and column in case *instId* is a mosaic.

l_points	The list of the points to be deselected.

Value Returned

`t`	The specified vertices of the specified figure are deselected.
`nil`	The specified vertices of the specified figure are not deselected.

Example

```
geDeselectObjectPoint(fig list(list(inst 0 1 2)) -> list(1.0:2.0))
```

Deselects the object at x 1 y 2.

© 2020 All Rights Reserved.

geDeselectPoint

```
geDeselectPoint(
     [ w_windowId ]
     [ l_point ]
     )
     => t / nil
```

Description

Calls `geSubSelectPoint` to deselect the figure at the specified point in the specified window. This might deselect just a vertex or edge, or the whole figure, depending on the state of the `partialSelect` window environment variable. Use the `F4` key to toggle this variable. You can see the state of `partialSelect` in the window banner.

Arguments

w_windowId	Window ID of the window to use. If not specified, the current window is used.
l_point	The point to deselect.

Value Returned

`t`	The figure is deselected.
`nil`	The figure is not deselected.

Example

```
geDeselectPoint(hiGetCurrentWindow() list(34.7 12.6))
```

Deselects the figure at 34.7x 12.6y in the current window and returns `t`.

Interactive Function

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

Enter this function with only the window ID argument; the system prompts you to point to the figure to deselect and returns `t`. If *w_windowId* is not specified, the current window is used.

© 2020 All Rights Reserved.

geDeselectProtected

```
geDeselectProtected(
     [ d_cellview ]
     )
     => t / nil
```

Description

Deselects all the objects that are marked protected in the specified cellview.

Arguments

d_cellview	The cellview in which the protected objects are to be made deselected

Value Returned

`t`	Protected objects are deselected.
`nil`	Protected objects are not deselected.

Example

```
geDeselectProtected()
```

Deselects the protected objects in the edit cellview

© 2020 All Rights Reserved.

geGetPotentialSelection

```
geGetPotentialSelection(
     [ w_windowId ]
     [ g_ignoreSelect]
     [ l_point]
     )
     => ((d_figId l_path) g_isAllSelected [l_selArray]) / nil
```

Description

Returns the figure ID of the object which can be selected, hierarchical path of the figure, and indicates if the figure is fully or partially selected.

If the boolean *g_ignoreSelect* is set, the current selection state is ignored.

Arguments

w_windowId	Window ID of the window to use.
g_ignoreSelect	Boolean which indicates if the current selection state should be ignored.
l_point	The point to potentially select.

Value Returned

(d_figId l_path)	ID of the figure and hierarchical path of the figure at the specified point.
g_isAllSelected	When `t`, indicates if the figure is fully selected. `nil` indicates partially selected.
[l_selArray]	If the figure is partially selected, indicates which figure points are selected.
`nil`	The function failed.

Example

```
geGetPotentialSelection(window(3) t 10:40)
```


© 2020 All Rights Reserved.

geGetSelSet

```
geGetSelSet(
     [ w_windowId ]
     [ d_cellview ]
     )
     => l_selectedSet / nil
```

Description

Returns a list of the currently selected objects in a window or cellview.

If nothing is specified, the edit cellview of the current window is used. If a window is specified, its edit cellview is used.

Arguments

w_windowId	Window ID of the window to use.
d_cellview	The cellview ID.

Value Returned

l_selectedSet	A list of the selected objects.
nil	The function failed.

Example

```
geGetSelSet()
```

Returns a list of selected objects in the edit cellview of the current window.

© 2020 All Rights Reserved.

geGetSelectedSet

```
geGetSelectedSet(
     [ d_cellview ]
     )
     => l_figs / nil
```

Description

Returns a list of selected objects in a cellview.

Arguments

d_cellview	Database ID of the cellview you want to query. If no cellview is specified, the current cellview is used.

Value Returned

l_figs	A list containing all objects that are selected (completely or partially) in the specified cellview.
nil	No objects are selected in the cellview.

© 2020 All Rights Reserved.

geGetSelSetCount

```
geGetSelSetCount(
     [ d_cellview ]
     )
     => n_count / 0
```

Description

Counts selected objects in a cellview.

Arguments

d_cellview	Database ID of the cellview you want to query. If no cellview is specified, the current cellview is used.

Value Returned

n_count	The number of objects in the selected set.
0	No objects are in the selected set.

© 2020 All Rights Reserved.

geGetSelectedInstancesCount

```
geGetSelectedInstancesCount(
     [ d_cellviewId ]
     )
     => n_count
```

Description

Counts selected instances in a cellview.

Arguments

d_cellviewId	Database ID of the cellview you want to query. If no cellview is specified, the current cellview is used.

Value Returned

n_count	The number of objects in the selected set.

Example

```
instSelCount = geGetSelectedInstancesCount(cvId)
```


© 2020 All Rights Reserved.

geGetSelProtectedObjects

```
geGetSelProtectedObjects(
     [ d_cellview ]
     )
     => l_results / nil
```

Description

Returns a list of all the objects that are marked protected in the specified cellview.

Arguments

d_cellview	The cellview in which the protected objects are looked for. If not specified, the list of protected objects in the current cellview is returned.

Value Returned

l_results	Returns a list of protected objects
nil	No list is returned

Example

```
geGetSelProtectedObjects()
```

Returns a list of all the protected objects in the edit cellview

© 2020 All Rights Reserved.

geGetSelSetFigPoint

```
geGetSelSetFigPoint(
     d_figId
     )
     => l_selPoints / nil
```

Description

Returns a list indicating whether points are selected for an object.

The list contains Boolean values corresponding to the point list of object *d_figId*. Each Boolean is `t` if the corresponding object point is selected; otherwise, it is `nil`. Since instances, mosaics, labels, and dots cannot be partially selected, the list contains a single Boolean specifying whether the object is selected or not.

Note: For rectangular objects, this function returns a list of four Boolean values corresponding to the `lowerLeft`, `upperLeft`, `upperRight`, and `lowerRight` points.

Arguments

d_figId	Database ID of the object you want to select.

Value Returned

l_selPoints	A list containing Boolean values corresponding to the point list of object *d_figId*. Each Boolean is `t` if the corresponding object point is selected; otherwise, it is `nil`.
`nil`	No objects are selected.

© 2020 All Rights Reserved.

geGetWindowSelProtectionHighlight

```
geGetWindowSelProtectionHighlight(
     [ w_window ]
     )
     => t / nil
```

Description

Returns whether or not the protected objects in the specified window are highlighted.

Arguments

w_window	The window that is queried for the highlight state of the protected objects. If not specified, the highlight state of the protected objects in the current window is returned.

Value Returned

`t`	Returned if the protected objects in the specified window are highlighted
`nil`	Returned if the protected objects in the specified window are not highlighted

Example

```
geGetWindowSelProtectionHighlight()
=> t
```

Indicates that the protected objects in the current window are highlighted.

© 2020 All Rights Reserved.

geGetWindowSelProtectionSelectability

```
geGetWindowSelProtectionSelectability(
     [ w_window ]
     )
     => t / nil
```

Description

Returns whether or not the protected objects in the specified window can be selected.

Arguments

w_window	The window that is queried for the selectability state of the protected objects. If not specified, the selectable state of the protected objects in the current window is returned.

Value Returned

`t`	Returned if the protected objects in the specified window can be selected
`nil`	Returned if the protected objects in the specified window cannot be selected

Example

```
geGetWindowSelProtectionSelectability()
=> t
```

Indicates that the protected objects in the current window can be selected.

© 2020 All Rights Reserved.

geHiDragFig

```
geHiDragFig(
     w_windowID
     r_form
     t_srcPrompt
     t_dstPrompt
     t_cmdName
     t_SelSet
     g_useMultiWindow
     [ l_firstPoint ]
     )
```

Description

Used in conjunction with other SKILL functions to implement the dragging of selected database figures.

Arguments

w_windowID	Window ID.
r_form	Form data structure. Valid Values: `hiGetCurrentForm` or any other valid form of data structure if there is a form, `0` if there is no form.
t_srcPrompt	Prompt string for the origin or source point.
t_dstPrompt	Prompt string for the destination point.
t_cmdName	The command name depends on the dragging action you desire, such as copy, move, or stretch. The command name could be any string or empty string, and the behavior is the same. Valid Values: `Copy`, `Move`, `Stretch`, or any other string or empty string.
t_SelSet	Specifies which selected set or object's image is to be dragged. Its value could be a selected set, a selected object, or `nil`. In each case, the return value is the same. It prompts you to specify the first point with the mouse (source point in source cellview), then it prompts you to specify the second point with the mouse (destination point in the destination cellview).

© 2020 All Rights Reserved.

	The return value is a list: `((source_point_CV source_point_coordination)(destination_point_CV destination_point_coordination))` If the argument is `nil`, a dragging image does not appear on the screen; otherwise, a dragging image appears and can be moved with the mouse. You can use `geGetSelSet` to get the value of the selected set or object. It returns `nil` if nothing is selected. Valid Values: `geGetSelSet` or any valid selected set or `nil`.
`g_useMultiWindow`	Supports multiple windows. If you set this argument to `nil`, you cannot use snap mode. If you set this argument to `t`, the snap mode is only respected in the source window. Valid Values: `t` (supports multiple windows), `nil` (does not support multiple windows).
`l_firstPoint`	First source point specified as a list.

Example 1 with Returned Value:

```
selSet = geGetSelSet()
geHiDragFig(window(2) 0 "Enter Origin Point" "Enter Destination Point" "Copy"
selSet t)
```

The example above returns the value

```
((db:21429292 (-0.4375 0.8125))
(db:21429292 (-0.4375 -0.1875))
)
```

Example 2 with Returned Value:

```
selSet = geGetSelSet()
geHiDragFig(window(2) hiGetCurrentForm() "Enter Origin Point"
"Enter Destination Point" "Copy" selSet t)
```

The example above returns the value

```
((db:21429292 (-0.5 0.8125))
(db:21429292 (-0.4375 -0.3125))
)
```

Multi-Window Example:

```
selSet = geGetSelSet()
```


© 2020 All Rights Reserved.

```
geHiDragFig(window(2) hiGetCurrentForm() "Enter Origin Point"
"Enter Destination Point" "Move" selSet t)
nil
>
hiSetCurrentWindow(window(2))
t
mouseAddPt()
-0.25:0.875
t
hiSetCurrentWindow(window(3))
t
mouseAddPt()
1.5:-1.125
t
(
      (db:21267500
      (-0.25 0.875)
      )
      (db:21258284
      (1.5 -1.125)
      )
)
```

The image which is dragged into another window is just an image, not a real object. You cannot select or save it. The window is not edited. It disappears after you use the redraw or the fit command.

© 2020 All Rights Reserved.

geHiDisplaySelectionForm

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

Description

Opens a form that allows you to manage selection options.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The function failed

© 2020 All Rights Reserved.

geHighlightProtected

```
geHighlightProtected(
     [ w_window ]
     )
     => t / nil
```

Description

Displays a halo around objects that are protected from being selected in the specified cellview.

Arguments

w_window	The window in which the protected objects are to be highlighted. If no window Id is pecified, the protected objects in the current window are highlighted.

Value Returned

`t`	Protected objects are highlighted.
`nil`	Protected objects are not highlighted.

Example

```
geHighlightProtected()
```

Adds a halo around all the protected objects in the edit cellview.

© 2020 All Rights Reserved.

geHiSetSelProtectionEF

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

Description

Starts an enter function to set selection protection.

Arguments

None

Value Returned

`t`	The enter function starts.
`nil`	The enter function does not start.

Examples

```
geHiSetSelProtectionEF()
```

Starts the enter function to set selection protection.

© 2020 All Rights Reserved.

geHiUnsetSelProtectionEF

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

Description

Starts an enter function to remove selection protection.

Arguments

None

Value Returned

`t`	The enter function starts.
`nil`	The enter function does not start.

Examples

```
geHiUnsetSelProtectionEF()
```

Starts the enter function to remove selection protection.

© 2020 All Rights Reserved.

geIsFigAllSelected

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

Description

Checks whether an object is completely selected.

Arguments

d_figId	Database ID of the object you want to select.

Value Returned

t	*d_figId* is completely selected.
nil	*d_figId* is not completely selected.

© 2020 All Rights Reserved.

geIsFigPartiallySelected

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

Description

Checks whether an object is partially, but not completely, selected.

Arguments

d_figId	Database ID of the object you want to select.

Value Returned

t	*d_figId* is partially selected.
nil	*d_figId* is not partially selected.

© 2020 All Rights Reserved.

geIsFigSelected

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

Description

Checks whether an object is at least partially selected.

Arguments

d_figId	Database ID of the object you want to select.

Value Returned

t	Object *d_figId* is at least partially selected.
nil	Object *d_figId* is not at least partially selected.

© 2020 All Rights Reserved.

geIsObjectAllSelected

```
geIsObjectAllSelected(
     d_figId
     [ l_path ]
     )
     => t / nil
```

Description

Tests if the specified figure is fully selected.

Arguments

d_figId	The figure to be tested.
l_path	Hierarchical path to the *d_figId*, in the following format:

```
list(list(instId memInst x y)
         list(instId memInst x y)
                    .
                    .
                    .
     )
```

x and *y* are integers defining the row and column in case *instId* is a mosaic.

Value Returned

t	The specified figure is fully selected.
nil	The specified figure is not fully selected.

Example

```
geIsObjectAllSelected(figId)
```

Checks to see if `figId` is selected.

© 2020 All Rights Reserved.

geIsObjectPartiallySelected

```
geIsObjectPartiallySelected(
     d_figId
     [ l_path ]
     )
     => t / nil
```

Description

Tests if the specified figure is partially selected.

Arguments

d_figId	The figure to be tested.

l_path	Hierarchical path to the *d_figId*, in the following format:
	`list(list(instId memInst x y)`
	`list(instId memInst x y)`
	.
	.
	.
	`)`
	x and *y* are integers defining the row and column in case *instId* is a mosaic.

Value Returned

`t`	The specified figure is partially selected.
`nil`	The specified figure is not partially selected.

Example

```
geIsObjectPartiallySelected(figId)
```

Checks to see if `figId` is partially selected.

© 2020 All Rights Reserved.

geIsObjectSelected

```
geIsObjectSelected(
     d_figId
     [ l_path ]
     )
     => t / nil
```

Description

Tests if the specified figure is partially or fully selected.

Arguments

d_figId	The figure to be tested.

l_path	Hierarchical path to the *d_figId*, in the following format:
	`list(list(instId memInst x y)`
	`list(instId memInst x y)`
	`.`
	`.`
	`.`
	`)`
	x and *y* are integers defining the row and column in case *instId* is a mosaic.

Value Returned

`t`	The specified figure is partially or fully selected.
`nil`	The specified figure is neither partially nor fully selected.

Example

`geIsObjectSelected(figId)`

Checks to see if `figId` is partially or fully selected.

© 2020 All Rights Reserved.

geIsSelProtection

```
geIsSelProtection(
     w_window
     d_figId
     l_figId
     )
     => l_results
```

Description

Returns whether the specified object in the specified window is protected from being selected.

Arguments

w_window	The window in which the protection state of objects is being queried
d_figId	Object that is queried for its protection state
l_figId	List of objects that are queried for their protection state

Value Returned

l_results	List of objects along with their protection state, in the following format: `(figId1 t / nil figId2 t / nil...)`

Example

```
geIsSelProtection(winId figId)
```

Returns the protection state of the `figId` figure in the `win` window

© 2020 All Rights Reserved.

gePointQuery

```
gePointQuery(
     w_windowId
     l_point
     t_filterProc
     )
     => d_figId / nil
```

Description

Returns the ID of the object found at the point *l_point*.

This function searches the database of the cellview in the window *w_windowId* at the point *l_point*. It chooses the object with the edge closest to the point. The point must be inside the object. It is the same as `geSingleSelectPoint` except it doesn't modify the selected set.

Arguments

w_windowId	Database ID of the window containing the object.
l_point	Location of the point used for the search.
t_filterProc	The name of a user-defined procedure which accepts *d_figID*, and returns `nil` if *figID* is filtered.

Value Returned

d_figId	The ID of the object found in the query.
`nil`	No object is found.

Example

```
procedure(myFilterProc(objId)
if(objId~>objType=="rect" t nil)
)
gePointQuery( win3 1:1 "myFilterProc")
```

Returns the ID of the database object found at the point 1:1, only if the object is a `rect`

© 2020 All Rights Reserved.

gePointToFig

```
gePointToFig(
     [ w_windowId ]
     [ g_ignoreSelect ]
     [ l_list ]
     )
     => d_figure / nil
```

Description

Returns the ID of the object located at the coordinates given in *l_list*.

This function is like geSingleSelectPoint, except the object indicated is returned rather than added to the selection. If no figure satisfies the selection criteria, nil is returned. If set, the Boolean *g_ignoreSelect* tells the function to ignore whether objects are currently selected when searching the database.

Arguments

w_windowId	Database ID of the window containing the objects.
g_ignoreSelect	Specifies whether to ignore objects currently selected when searching for the object. Valid Values: t, nil
l_list	List of X and Y coordinates in user units to specify locations to look for objects.

Value Returned

d_figure	Database ID of the first object satisfying the selection criteria.
nil	No objects satisfy the selection criteria.

Example

```
gePointToFig( win2 list(0:0 10:10) )
```

Returns the object IDs of the objects located at the given coordinates.

© 2020 All Rights Reserved.

geSelectAll

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

Description

Selects all objects in the specified window.

Arguments

w_windowId	Database ID of the window containing the objects to be selected. If not specified, the current window is used.

Value Returned

`t`	All objects are selected.
`nil`	No object is selected.

© 2020 All Rights Reserved.

geSelectAllFig

```
geSelectAllFig(
     [ d_cellview ]
     )
     => t / nil
```

Description

Selects all objects in a cellview that pass the selection filter.

Arguments

d_cellview	Database ID of the cellview you want to select. If not specified, the current cellview is used.

Value Returned

`t`	The objects pass the selection filter and are selected.
`nil`	The objects do not pass the selection filter and are not selected.

Example

```
geSelectAllFig( cell3 )
```

Selects all objects in `cell3`.

© 2020 All Rights Reserved.

geSelectArea

```
geSelectArea(
     [ w_windowId ]
     [ l_points ]
     [ t_areaShapeType ]
     )
     => t / nil
```

Description

Adds the objects within the area defined by `t_areaShapeType` to the selection.

Note: You can use left mouse button to select area in enclosed mode and the middle mouse button to select area in enclosed and crossed mode.

Arguments

w_windowId	Window ID of the window containing the objects. If not specified, the current window is used.
l_points	List of points that define the area for selection.
t_areaShapeType	The shape of the area to use for selection. Valid Values: `rectangle`, `polygon`. The default shape is `rectangle`.

Value Returned

`t`	The objects within the specified area are added to the selection.
`nil`	The objects within the specified area are not added to the selection.

Example

```
geSelectArea(hiGetCurrentWindow() list(list(0 0) list(100 100) list(0 100))
"polygon")
```

Adds the objects within the specified polygon to the selection.

© 2020 All Rights Reserved.

Interactive Function

```
geSelectArea(
     [ w_windowId ]
     [ l_points ]
     [ g_areaShapeType ]
     )
     => t / nil
```

Enter this function with only the window ID argument; the system prompts you to point to the area to select and returns `t`. Press `Shift` and draw a box to add objects to the selected set. Press `Ctrl` and draw a box to remove objects from the selected set. If *`w_windowId`* is not specified, the current window is used.

When the system prompts you to select the area, you can press F3 to display the *Select Area Shape Type Form* to select a polygon. For more information, see *Select Area Shape Type Form* in Appendix A of *Virtuoso Layout Suite L User Guide*.

© 2020 All Rights Reserved.

geSelectByLine

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

Description

Enables you to select objects by drawing a line. Press `Shift` and draw a line to overlap the additional objects to be selected. Press `Ctrl` and draw a line to overlap the objects to be removed from the selected set. Double-click to end the line.

Arguments

None

Value Returned

`t`	Objects overlapping the line are selected.
`nil`	Objects overlapping the line are not selected.

Example

```
geSelectByLine()
```

Enables you to draw a line to overlap all objects you want to select.

© 2020 All Rights Reserved.

geSelectBy2PointsLine

```
geSelectBy2PointsLine(
     [ w_windowId ]
     )
     => t / nil
```

Description

Selects objects by drawing a line. Press `Shift` and draw a line to overlap the additional objects to be selected. Press `Ctrl` and draw a line to overlap the objects to be removed from the selected set. The selection is done when the second point is entered.

Arguments

w_windowId	Database ID of the window containing the objects to be selected. If not specified, the current window is used.

Value Returned

`t`	All objects are selected.
`nil`	No object is selected.

Example

```
geSelectBy2PointsLine(window(2))
```


© 2020 All Rights Reserved.

geSelectFig

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

Description

Selects an object in the layout.

Arguments

d_figId	Database ID of the object you want to select.

Value Returned

`t`	The object passes the selection filter and is selected.
`nil`	The object does not pass the selection filter and is not selected.

Example

```
geSelectFig( fig35 )
```

Selects `fig35`.

© 2020 All Rights Reserved.

geSelectFigs

```
geSelectFigs(
     l_figList
     )
     => l_list / nil
```

Description

Selects figures from a specified list of figures.

Arguments

l_figList	List of figure IDs.

Value Returned

l_list	List of selected figures.
nil	An error occurred, the figures are not selected and the list of figures is not returned.

Example

```
geSelectAllFig()
sel = geGetSelSet()
geSelectFigs(sel)
```


© 2020 All Rights Reserved.

geSelectFigNoFilter

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

Description

Selects the specified figure regardless of the selection filter.

Arguments

d_figId	The figure to be selected.

Value Returned

`t`	The specified figure is selected.
`nil`	The specified figure is not selected.

Example

```
geSelectFigNoFilter( figId )
```

Ignores all selection filters and selects `figId`.

© 2020 All Rights Reserved.

geSelectFigPoint

```
geSelectFigPoint(
     d_figId
     l_vertex
     )
     => t / nil
```

Description

Adds vertices of an object to the selected set if they pass the selection filter.

For vertex selection, rectangles are considered to have four vertices, starting with the lower left point and moving counterclockwise.

Instances, mosaics, dots, and labels cannot be partially selected, so this function cannot be used on these types of objects.

Arguments

d_figId	Database ID of the object you want to select.
l_vertex	A list of Boolean values corresponding to the list of points for the object. Each Boolean specifies whether the corresponding vertex of the object should be selected or not. If there are fewer points in *l_vertex* than in the object, they are applied to the first vertices in the object and the remaining vertices are unaffected. Valid Values: `t`, `nil`

Value Returned

`t`	The object passes the selection filter and is selected.
`nil`	The object does not pass the selection filter and is not selected.

Example

```
geSelectFigPoint( fig35 list(t nil t nil t nil t nil) )
```

Selects every other vertex on `fig35`.

© 2020 All Rights Reserved.

geSelectObject

```
geSelectObject(
     d_figId
     [ l_path ]
     )
     => t / nil
```

Description

Selects the specified figure at the specified hierarchical path, if it passes the selection criteria.

Arguments

d_figId	The database ID of the figure to be selected.
l_path	A path in the format `( (d_instId x_memInst x_row x_column)(d_instId x_memInst x y) ...)` *x_memInst* is used for an iterated instance. *x* and *y* are used for mosaic instances.

Value Returned

`t`	Selects the specified figure at the specified hierarchical path, if it passes the selection criteria.
`nil`	Does not select the specified figure.

© 2020 All Rights Reserved.

geSelectObjectNoFilter

```
geSelectObjectNoFilter(
     d_figId
     [ l_path ]
     )
     => t / nil
```

Description

Selects the specified figure at the specified hierarchical path.

Arguments

d_figId	Database ID of the figure to be selected.
l_path	The optional path in the format ((*d_instId x_memInst x_row x_column*)(*d_instId x_memInst x y*) ...) *x_memInst* is used for an iterated instance. *x* and *y* are used for mosaic instances.

Value Returned

t	Selects the specified figure at the specified hierarchical path.
nil	Does not select the specified figure.

© 2020 All Rights Reserved.

geSelectPoint

```
geSelectPoint(
     [ w_windowId ]
     [ l_point ]
     )
     => t / nil
```

Description

Uses the given point to select the object. If there is no object at that location, no object is selected and `nil` is returned.

It is recommended that `geAddSelectPoint` be used in place of this function.

Arguments

w_windowId	Window ID of the window to use. If not specified, the current window is used.
l_point	The location of the point to select. If not specified, the user will be prompted to specify one by clicking in the design window.

Value Returned

`t`	The point is used to select the object.
`nil`	The point is not used to select the object.

Example

```
geSelectPoint( hiGetCurrentWindow() list(70 70) )
```

Selects the object at 70x 70y in the current window.

Interactive Function

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


© 2020 All Rights Reserved.

Enter this function with only the window ID argument; the system prompts you to point to the object to add to the selection and returns `t`. If *`w_windowId`* is not specified, the current window is used.

© 2020 All Rights Reserved.

geSelectProtected

```
geSelectProtected(
     [ d_cellview ]
     )
     => t / nil
```

Description

Selects the objects that are protected from being selected in the specified cellview.

Arguments

d_cellview	The cellview in which the protected objects are to be selected. If not specified, the protected objects in the edit cellview are selected.

Value Returned

`t`	All the protected objects are selected.
`nil`	The protected objects are not selected.

Examples

```
geSelectProtected()
```

Selects all the protected objects in the edit cellview

© 2020 All Rights Reserved.

geSelectViaPile

```
geSelectViaPile(
     [ d_cellview ]
     )
     => t / nil
```

Description

Extends the selection set by adding via pile for each via selected. If the selection set does not contain vias, `nil` is returned.

Arguments

d_cellview	Optional argument specifies the cellview to process. If not specified, the current cellview is used.

Value Returned

`t`	An additional via has been added to the selection set.
`nil`	The selection set does not contain vias.

Example

```
geSelectViaPile( )
```

Selects vias associated with the via pile of the via selected in the current cellview.

© 2020 All Rights Reserved.

geSetEnterFunctionSelectionMode

```
geSetEnterFunctionSelectionMode(
     w_windowId
     x_mode
     )
     => t / nil
```

Description

Sets the selection mode for the enter function associated with the specified window.

Arguments

w_windowId	Window ID.
x_mode	Sets the selection mode as follows: 0 = no select option 1 = select override 1 = full select override 2 = partial select 3 = partial select override 4 = selection highlight

Value Returned

`t`	Default or if the windowID has an enter function and the mode was set
`nil`	The selection mode is not set.

Example

```
geSetEnterFunctionSelectionMode(hiGetCurrentWindow() 2)
geSetEnterFunctionSelectionMode(window(6) 2)
```

Both of these examples set the selection mode to 2 (partial select) for the enter function associated with the current window `(window 6)`.

© 2020 All Rights Reserved.

geSelObjectsPartiallySelected

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

Description

Fully selects the objects that are partially selected in the specified window. If all objects are already fully selected, `geSelObjectsPartiallySelected` extends the selection according to the current selection granularity.

Arguments

w_windowId	Database ID of the window containing the objects.

Value Returned

`t`	The objects in the window are fully selected
`nil`	The objects in the window are not fully selected.

Example

```
geSelObjectsPartiallySelected()
```


© 2020 All Rights Reserved.

geSetSelProtection

```
geSetSelProtection(
     [ d_cellview ] | [ d_figId ] | [ l_figId ]
     )
     => l_results / nil
```

Description

Sets the specified object as protected, which prevents it from being selected. If no *figId* is specified, the selected objects in the specified or in the current cellview are set as protected.

Arguments

[*d_cellview*] | [*d_figId*] | [*l_figId*]

Specify the cellview, object, or list of objects to be set as protected.

Value Returned

l_results	List of objects that are set as protected
nil	The specified object is not set as protected

Examples

```
geSetSelProtection(list(figId1 figId2))
```

Sets selection protection on the `figId1` and `figId1` objects

```
geSetSelProtection(cellViewA)
```

Sets selection protection on all the selected objects in the `cellViewA` cellview

```
geSetSelProtection()
```

Sets selection protection on all the selected objects in the edit cellview

© 2020 All Rights Reserved.

geSetSelProtectionAllObjects

```
geSetSelProtectionAllObjects(
     [ d_cellview ]
     )
     => t / nil
```

Description

Sets all the objects in the specified cellview as protected, which prevents them from being selected.

Arguments

d_cellview	The cellview in which all the objects object are to be set as protected. If not specified, all the objects in the edit cellview are set as protected.

Value Returned

`t`	All the objects are set as protected.
`nil`	The objects are not set as protected.

Examples

```
geSetSelProtectionAllObjects()
```

Sets all the objects in the edit cellview as protected

© 2020 All Rights Reserved.

geSetSelProtectionHighlightOptionForm

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

Description

Opens the *Selection Protection Highlight Options* form where you can configure the settings for highlighting protected objects.

Arguments

None

Value Returned

`t`	The *Selection Protection Highlight Options* form opens.
`nil`	The *Selection Protection Highlight Options* form does not open.

Examples

```
geSetSelProtectionHighlightOptionForm()
```

Opens the *Selection Protection Highlight Options* form.

© 2020 All Rights Reserved.

geSetSelProtectionHighlightOptions

```
geSetSelProtectionHighlightOptions(
     w_window
     d_cellview
     [ l_options ]
     )
     => t / nil
```

Description

Sets the highlight options for objects that are marked protected.

Arguments

None

Value Returned

w_window	The window in which the specified highlight options are set on the protected objects.
d_cellview	The cellview in which the protected objects are to be selected.
l_options	List of highlight settings to be used on the protected objects. If no options are specified, the default settings will be applied.

Examples

```
geSetSelProtectionHighlightOptions(windowId cellviewId list("fadeout" "under"
"thin" 40)))
```

Sets the following characteristics of the halo on the selection protected objects: *Halo Mode*: `fadeout`, *Level*: `under`, *Width*: `thin`, and *Transparency*: `40`.

© 2020 All Rights Reserved.

geSingleSelectBox

```
geSingleSelectBox(
     [ w_windowId ]
     [ g_partial ]
     [ l_point ]
     )
     => t / nil
```

Description

Implements the `single` mode of area selection.

With no arguments, it prompts you to enter the area to be selected in the current window. Partial selection is performed if the window environment variable `partialSelect` is set.

Arguments

w_windowId	Database ID of the window containing the objects.
g_partial	Whether partial selection should be performed. Valid Values: `t`, `nil`
l_point	The bounding box to be used for selection. When no bounding box is specified, this function operates interactively.

Value Returned

`t`	The objects are selected.
`nil`	The objects are not selected.

Example

```
geSingleSelectBox( win1 nil list(0:0 25:25) )
```

Selects the objects in `win1` enclosed in the bBox described by the coordinates 0:0, 25:25. Partial selection is not performed.

© 2020 All Rights Reserved.

geSingleSelectPoint

```
geSingleSelectPoint(
     [ w_windowId ]
     [ g_partial ]
     [ l_point ]
     )
     => t / nil
```

Description

Implements the `single` mode of point selection.

With no arguments specified, it prompts you to point at the object to be selected in the current window. Partial selection is performed if the window environment variable `partialSelect` is set.

Arguments

w_windowId	Database ID of the window containing the objects.
g_partial	Whether partial selection should be performed. Valid Values: `t`, `nil`
l_point	Point to be used for selection. When no point is specified, this function operates interactively.

Value Returned

`t`	The objects are selected.
`nil`	The objects are not selected.

Example

```
geSingleSelectPoint ( win3 nil (0:0) )
```

Selects the object in `win3` located at the coordinate 0:0. Partial selection is not performed.

© 2020 All Rights Reserved.

geSubSelectBox

```
geSubSelectBox(
     [ w_windowId ]
     [ g_partial ]
     [ l_bBox ]
     )
     => t / nil
```

Description

Implements the `sub` mode of area selection.

With no arguments, it prompts you to enter the de-selection area in the current window. Partial de-selection is performed if the window environment variable `partialSelect` is set.

Arguments

w_windowId	Window ID of the window containing the objects.
g_partial	Whether partial selection should be performed. Valid Values: `t`, `nil`
l_bBox	The bounding box to be used for de-selection. When no bounding box is specified, this function operates interactively.

Value Returned

`t`	The de-selection is performed.
`nil`	The de-selection is not performed.

Example

```
geSubSelectBox( win1 nil list(0:0 25:25) )
```

Deselects the objects in `win1` enclosed in the bBox described by the coordinates 0:0, 25:25. Partial selection is not performed.

© 2020 All Rights Reserved.

geSubSelectPoint

```
geSubSelectPoint(
     [ w_windowId ]
     [ g_partial ]
     [ l_point ]
     )
     => t / nil
```

Description

Implements the `sub` mode of point selection.

With no arguments, it prompts you to point at the object to be de-selected in the current window. Partial de-selection is performed if the window environment variable `partialSelect` is set.

Arguments

w_windowId	Database ID of the window containing the objects.
g_partial	Whether partial selection should be performed. Valid Values: `t`, `nil`
l_point	The point to be used for de-selection. When no point is specified, this function operates interactively.

Value Returned

`t`	The de-selection is performed.
`nil`	The de-selection is not performed.

Example

```
geSubSelectPoint( win3 nil list(0:0) )
```

Deselects the object in `win3` located at the coordinate 0:0. Partial selection is not performed.

© 2020 All Rights Reserved.

geToggleAreaSelectOption

```
geToggleAreaSelectOption(
     )
     => enclosed_crossed / enclosed / edgeOnly / vertexAndEdge
```

Description

Toggles area selection mode.

If full mode is active, `geToggleAreaSelectOption` toggles between `enclosed_crossed` and `enclosed`. If partial mode is active, it toggles between `edgeOnly` and `vertexAndEdge`

Arguments

None

Value Returned

enclosed_crossed	`enclosed_crossed` area selection mode is active.
enclosed	`enclosed` area selection mode is active.
edgeOnly	`edgeOnly` area selection mode is active.
vertexAndEdge	`vertexAndEdge` area selection mode is active.

© 2020 All Rights Reserved.

geToggleDisplayResolution

```
geToggleDisplayResolution( )
```

Description

Rotates the value of the graphic environment variable `displayResolution` between the values `Very High`, `High`, `Medium`, and `Low`. This function is bound to the `F9` bindkey in the default layout editor bindkeys.

Arguments

None

Value Returned

`t_message`	Returns an information message indicating which resolution is applied to the current window..

Example

```
geToggleDisplayResolution()
```

Returns the information message, `Display resolution set to 'Very High'`.

© 2020 All Rights Reserved.

geToggleFilterSize

```
geToggleFilterSize( )
```

Description

Rotates the value of the graphic environment variable `filterSize` between the values 0.0, 3.0, 10.0, 25.0, and 50.0. This function is bound to the `F9` bindkey in the default layout editor bindkeys.

`filterSize` is used to control the size of the smallest features displayed and to control instance masters read in when opening or displaying a design. Larger values give faster display or read-in but less detail.

Arguments

None

Value Returned

None

Example

```
geToggleFilterSize()
\i geToggleFilterSize
\t Set filterSize to 10.000000
\r 10.0
\p >
\i aa = geToggleFilterSize
\t Set filterSize to 25.00000
```


© 2020 All Rights Reserved.

geTogglePartialSelect

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

Description

Toggles partial selection off and on.

This is the equivalent of pressing the `F4` key.

Arguments

None

Value Returned

`t`	`partialSelect` is on.
`nil`	`partialSelect` is off.

© 2020 All Rights Reserved.

geToggleWindowSelProtectionHighlight

```
geToggleWindowSelProtectionHighlight(
     [ w_window ]
     )
     => t
```

Description

Toggles the highlight state of protected objects in the specified window.

Arguments

w_window	The window in which the highlighting of protected objects is to be toggled. If no window Id is specified, the highlighting on the protected objects in the current window is toggled.

Value Returned

`t`	Highlight state of the protected objects is reversed.

Example

```
geToggleWindowSelProtectionHighlight()
```

Unhighlights the protected objects in the edit cellview if they are highlighted or highlights the protected objects if they are not highlighted

© 2020 All Rights Reserved.

geToggleWindowSelProtectionSelectability

```
geToggleWindowSelProtectionSelectability(
     [ w_window ]
     )
     => t
```

Description

Toggles the selectability state of protected objects in the specified window.

Arguments

w_window	The window in which the selectability of protected objects is to be toggled. If no window Id is specified, the selectability of the protected objects in the current window is toggled.

Value Returned

`t`	Selectability of the protected objects is reversed.

Example

```
geToggleWindowSelProtectionSelectability()
```

Enables you to select the protected objects in the edit cellview if they are unselectable or makes the protected objects unselectable if they were selectable earlier

© 2020 All Rights Reserved.

geUnhighlightProtected

```
geUnhighlightProtected(
     [ w_window ]
     )
     => t / nil
```

Description

Removes the halo from the objects marked as protected in the specified cellview.

Arguments

w_window	The window in which protected objects are to be unhighlighted

Value Returned

`t`	Protected objects are unhighlighted
`nil`	Protected objects remain highlighted

Examples

```
geUnhighlightProtected()
```

Removes the halo from all the protected objects in the edit cellview

© 2020 All Rights Reserved.

geUnsetAllSelProtection

```
geUnsetAllSelProtection(
     [ d_cellview ]
     )
     => t / nil
```

Description

Removes the selection protection from all the protected objects in the specified cellview.

Arguments

d_cellview	The cellview in which all the protected objects are to be made unprotected

Value Returned

`t`	The selection protection of all the objects is removed.
`nil`	The selection protection of objects is not removed.

Example

```
geUnsetAllSelProtection()
```

Removes the selection protection from all the objects in the edit cellview

© 2020 All Rights Reserved.

geUnsetSelProtection

```
geUnsetSelProtection(
     [ d_cellview ] | [ d_figId ] | [ l_figId ]
     )
     => l_results / nil
```

Description

Removes the selection protection from the specified protected objects. If no *figId* is specified, protection of the selected objects in the specified or in the current cellview is removed.

Arguments

[*d_cellview*] | [*d_figId*] | [*l_figId*]	Specify the cellview, object, or list of objects from which the protection needs to be removed.

Value Returned

l_results	List of figures made unprotected, and therefore, selectable
nil	The protection of the specified object is not removed

Examples

```
geUnsetSelProtection(list(figId1 figId2))
```

Removes the selection protection from the `figId1` and `figId1` objects

```
geUnsetSelProtection(cellViewA)
```

Removes the selection protection from all the selected objects in the `cellViewA` cellview

```
geUnsetSelProtection()
```

Removes the selection protection from all the selected objects in the edit cellview

© 2020 All Rights Reserved.

DSA Function

geDSAFreeze

```
geDSAFreeze(
     w_windowId
     l_point
     )
     => t / nil
```

Description

Toggles between the freeze and unfreeze states of the Dynamic Selection Assistant (DSA).

Arguments

w_windowId	Database ID of the window for which the DSA is required to be frozen or unfrozen.
l_point	Position of the cursor (x and y coordinates) at which the SKILL function is called.

Value Returned

`t`	The DSA toggled between freeze and unfreeze state successfully.
`nil`	The DSA did not toggle between freeze and unfreeze state successfully.

Example

```
geDSAFreeze(window(2) 8.0:17.0)
```

With the DSA in unfreeze state, this function freezes the DSA and displays all the shapes at the cursor location at 8:17 in the DSA.

© 2020 All Rights Reserved.

Stored Selection

Selection set can be stored and restored using the following functions. The selection set consists of full or partial selection of figures.

geDeleteStoredSelectionSet

```
geDeleteStoredSelectionSet(
     t_setName
     [ d_cellviewId ]
     )
     => t / nil
```

Description

Removes the specified stored selection set from the manual store selection set of the given cellview.

Arguments

t_setName	The name of the stored selection set.
d_cellviewId	Optional argument. When not specified, the current edit cellview is used. There is one current selection set by cellview.

Value Returned

`t`	The removal of the set is successful.
`nil`	The stored selection set does not exist.

Example

```
geDeleteStoredSelectionSet("selset1")
```


© 2020 All Rights Reserved.

geDeselectStoredSelectionSet

```
geDeselectStoredSelectionSet(
     t_setName
     [ d_cellviewId ]
     )
     => t / nil
```

Description

Removes the elements of the specified selection set from the current selection set of the given cellview.

Arguments

t_setName	The name of the stored selection set.
d_cellviewId	Optional argument. When not specified, the current edit cellview is used. There is one current selection set by cellview.

Value Returned

`t`	De-selection of elements is successful.
`nil`	The stored selection set does not exist.

Example

```
geDeselectStoredSelectionSet("selset1")
```


© 2020 All Rights Reserved.

geGetStoredSelectionSets

```
geGetStoredSelectionSets(
     d_cellviewId
     )
     => t_nameList / nil
```

Description

Returns the list of names for the stored selection sets in the given cellview.

Arguments

h_hlObjectId	Database ID of the highlighted object.

Value Returned

t_nameList	Returns the list of names.
nil	The cellview ID is invalid.

Example

```
geGetStoredSelectionSets(d_cellviewId)
```


© 2020 All Rights Reserved.

geHiStoreSelectionForm

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

Description

Opens the *Save/Restore Selection Set* form.

Arguments

None.

Value Returned

`t`	The *Save/Restore Selection Set* form opens.
`nil`	An error occurred.

Example

```
geHiStoreSelectionForm()
```


© 2020 All Rights Reserved.

geIsAutoStoreSelectionOn

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

Description

Returns true if the auto store selection set feature is enable.

Arguments

d_cellviewId	Optional argument of the database ID of the cellview.

Value Returned

t	Auto store selection set is on.
nil	The cellview ID is invalid.

Example

```
geIsAutoStoreSelectionOn(cv)
```


© 2020 All Rights Reserved.

geSelectNextAutoStoreSet

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

Description

Sets the next stored auto selection set as the current selection set. This function ends on the last stored selection set.

Arguments

d_cellviewId	Optional argument of a cellview ID.

Value Returned

t	The next stored auto selection set is made the current selection set.
nil	The cellview ID is invalid or the next stored auto selection set does not exist.

Example

```
geSelectNextAutoStoreSet()
```


© 2020 All Rights Reserved.

geSelectPreviousAutoStoreSet

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

Description

Restores the previous stored auto selection set. This function ends on the first stored selection set.

Arguments

d_cellviewId	Optional argument of a cellview ID.

Value Returned

`t`	The previous stored auto selection set is restored.
`nil`	The cellview ID is invalid or a previously stored auto selection set does not exist.

Example

```
geSelectPreviousAutoStoreSet()
```


© 2020 All Rights Reserved.

geSelectStoredSelectionSet

```
geSelectStoredSelectionSet(
     t_setName
     [ d_cellviewId]
     )
     => t / nil
```

Description

Adds the elements of the specified selection set into the current selection set of the given cellview.

Arguments

t_setName	The name of the stored selection set.
d_cellviewId	Optional argument of a cellview ID.

Value Returned

`t`	Elements of the specified selection are added to the current selection set.
`nil`	The cellview ID is invalid or the stored selection set does not exist.

Example

```
geSelectStoredSelectionSet("selset1")
```


© 2020 All Rights Reserved.

geSetAutoStoreSelection

```
geSetAutoStoreSelection(
     g_autoStore
     [ d_cellviewId ]
     )
     => t / nil
```

Description

Enables the auto store selection set feature on a cellview. Stored selection sets are remembered as long as the cellview is open. Once the cellview is closed, the recorded information is lost.

Deleted and modified objects are removed from the store selection set. If all objects are deleted in a stored selection set, the set is dropped. Does not recognize the undo/redo commands.

Arguments

g_autoStore	Optional argument that specifies whether the auto stored set is enabled. Valid Values: t, nil
d_cellviewId	Optional argument of a cellview ID.

Value Returned

`t`	The auto store selection set feature is enabled.
`nil`	The auto store selection set feature is not enabled.

Example

```
geSetAutoStoreSelection(t)
```


© 2020 All Rights Reserved.

geStoreSelectionSet

```
geStoreSelectionSet(
     t_setName
     [ d_cellviewId ]
     )
     => t / nil
```

Description

Records the current selection set into the cellview manual store selection set list.

Arguments

t_setName	The name of the stored selection set.
d_cellviewId	Optional argument of a cellview ID.

Value Returned

`t`	The current selection set is stored.
`nil`	The selection set is not stored.

Example

```
geStoreSelectionSet(setName)
```


© 2020 All Rights Reserved.

Highlight Functions

You can use the highlight functions to highlight shapes on specific layers of a cellview. The highlight functions are cellview based and highlight shapes in all windows containing the cellview.

The highlight functions use a “highlight set” to determine which shapes are highlighted in a cellview. A highlight set can be global or non-global. Global highlight sets are automatically drawn in the cellview. Non-global highlight sets are drawn when you use `gePushHilightStack`.You can only draw one non-global highlight set at a time.

To highlight shapes in a cellview, follow these steps:

1. Create a highlight set using `geCreateHilightSet`. Indicate the cellview and layer on which to draw the shapes. You can also indicate that the set is non-global by setting *`g_notGlobal`* to `t`.
2. Add shapes to the highlight set with the `geAddHilight` functions. You can also remove shapes from a highlight set with the `geDeleteHilight` functions.
3. Enable the highlight set by setting the enable attribute with
   `hlSetId->enable = t`
4. If the highlight set is non-global, use `gePushHilightStack` to draw the highlight set and `gePopHilightStack` to remove the highlight set from cellview.
5. If you want to change the layer of a highlight set, set the attribute with
   `hlSetId->layer = list("layer_name" "layer_purpose")`
6. If you want to disable the highlight set, reset the enable attribute with
   `hlSetId->enable = nil`

© 2020 All Rights Reserved.

geAddHilightArc

```
geAddHilightArc(
     g_hlSetId
     l_ellipsebBox
     l_arcbBox
     )
     => g_hlObjectId / nil
```

Description

Adds an arc to a highlight set.

l_ellipsebBox and *l_arcbBox* specify the location of the arc.

Arguments

g_hlSetId	Database ID of the highlight set.
l_ellipsebBox	List of points used to describe the ellipse.
l_arcbBox	List of points used to describe the arc.

Value Returned

g_hlObjectId	The arc is added to the set.
`nil`	The arc is not added to the set.

Example

```
geAddHilightArc( set1 list(0:0 10:10) list(0:8 10:8) )
```

Adds an arc to the highlight set `set1` and returns the object ID of the arc.

© 2020 All Rights Reserved.

geAddHilightBlockage

```
geAddHilightBlockage(
     g_hlSetId
     d_blockageId
     )
     => g_hlObjectId / nil
```

Description

Adds a blockage to a highlight set.

Arguments

g_hlSetId	Database ID of the highlight set.
d_blockageId	The blockage Id.

Value Returned

g_hlObjectId	The blockage is added to the set.
nil	The blockage is not added to the set.

© 2020 All Rights Reserved.

geAddHilightCircle

```
geAddHilightCircle(
     g_hlSetId
     l_center
     n_radius
     )
     => g_hlObjectId / nil
```

Description

Adds a circle to a highlight set.

l_center specifies the location of the circle, and *n_radius* specifies the size of the circle.

Arguments

g_hlSetId	Database ID of the highlight set.
l_center	Specifies the center of the circle.
n_radius	Specifies the size of the circle.

Value Returned

g_hlObjectId	The circle is added to the set.
nil	The circle is not added to the set.

Example

```
geAddHilightCircle( set1 0:0 10 )
```

Adds a circle to the highlight set `set1` whose center point is 0:0 and whose radius is 10. Returns the object ID of the circle.

© 2020 All Rights Reserved.

geAddHilightCurvedPath

```
geAddHilightCurvedPath(
     g_hlSetId
     l_points
     n_width
     )
     => g_hlObjectId / nil
```

Description

(ICADVM20.1 Only) Adds a curved path to a highlight set.

The points in *l_points* specify the location of the curved path, and *n_width* specifies the width of the curved path.

Arguments

g_hlSetId	Database ID of the highlight set.
l_points	List of points used to describe the curved path.
n_width	Width of the curved path.

Value Returned

g_hlObjectId	The curved path is added to the set.
nil	The curved path is not added to the set.

Example

```
hSet = geCreateHilightSet(geGetEditCellView() list("meta1" "drawing")
hSet~>enable = t
geAddHilightCurvedPath(hSet  list(-11.1:5.8 -11.1:-8.9 18.2:-8.9 18.2:-14.1) 1)
```

Adds a path to the highlight set and returns the object ID of the curved path.

© 2020 All Rights Reserved.

geAddHilightCurvedPolygon

```
geAddHilightPolygon(
     g_hlSetId
     l_points
     )
     => g_hlObjectId / nil
```

Description

(ICADVM20.1 Only) Adds a curved polygon to a highlight set.

The points in *l_points* specify the location of the curved polygon.

Arguments

g_hlSetId	Database ID of the highlight set.
l_points	List of points used to describe the curved polygon.

Value Returned

g_hlObjectId	The curved polygon is added to the set.
nil	The curved polygon is not added to the set.

Example

```
hSet = geCreateHilightSet(geGetEditCellView() list("meta1" "drawing")
hSet~>enable = t
geAddHilightCurvedPolygon(hSet  list(10:10 10:15 15:15 list(16.5 12.5 "arc" "cw")
15:10) )
```

Adds a curved polygon to the highlight set and returns the object ID of the polygon.

© 2020 All Rights Reserved.

geAddHilightDistance

```
geAddHilightDistance(
     h_hlSetId
     l_points [l_points]
     )
     => h_hlObjectId
```

Description

Adds a distance (like ruler) to the given hilight Set or Group.

Arguments

h_hlSetId	Unique ID of a specific hilight Set or Group.
l_points	The hilight distance in terms of points.
[l_points]	Additional hilight distance in terms of points.

Value Returned

h_hlObjectId	Returns the ID of the created hilight object.

Example

```
geAddHilightDistance(hgroupId list(-5:-3.6 2:-3.6) list(-5:0 2:-3.6))
```


© 2020 All Rights Reserved.

geAddHilightDonut

```
geAddHilightDonut(
     g_hlSetId
     l_center
     n_outR
     n_holeR
     )
     => g_hlObjectId / nil
```

Description

Adds a donut to a highlight set.

l_center specifies the center of the donut. The outer radius and hole radius are specified by *n_outR* and *n_holeR*.

Arguments

g_hlSetId	Database ID of the highlight set.
l_center	Specifies the center of the donut.
n_outR	Specifies the outer radius of the donut.
n_holeR	Specifies the inner radius of the donut.

Value Returned

g_hlObjectId	The donut is added to the set.
nil	The donut is not added to the set.

Example

```
geAddHilightDonut( set1 0:0 10 2 )
```

Adds a donut to the highlight set `set1` whose center point is 0:0, whose outer radius is 10, and whose inner radius is 2. Returns the object ID of the donut.

© 2020 All Rights Reserved.

geAddHilightDot

```
geAddHilightDot(
     g_hlSetId
     l_point
     n_width
     n_height
     )
     => g_hlObjectId / nil
```

Description

Adds a dot to a highlight set.

l_point specifies the location of the dot. The display size of the dot is specified by *n_width* and *n_height*.

Arguments

g_hlSetId	Database ID of the highlight set.
l_point	Point used to locate the dot.
n_width	Width of the dot.
n_height	Height of the dot.

Value Returned

g_hlObjectId	The dot is added to the set.
nil	The dot is not added to the set.

Example

```
geAddHilightDot( set1 list(0:0) 5 5 )
```

Adds a 5x5 dot to the highlight set `set1` with the center point at 0:0, and returns the object ID of the dot.

© 2020 All Rights Reserved.

geAddHilightEllipse

```
geAddHilightEllipse(
     g_hlSetId
     l_bBox
     )
     => g_hlObjectId / nil
```

Description

Adds an ellipse to a highlight set.

The points in *l_bBox* specify the location of the ellipse.

Arguments

g_hlSetId	Database ID of the highlight set.
l_bBox	List of points used to describe the ellipse.

Value Returned

g_hlObjectId	The ellipse is added to the set.
nil	The ellipse is not added to the set.

Example

```
geAddHilightEllipse( set1 list(0:0 10:10) )
```

Adds an ellipse to the highlight set `set1` and returns the object ID of the ellipse.

© 2020 All Rights Reserved.

geAddHilightFig

```
geAddHilightFig(
     g_hlSetId
     d_figId
     [ g_drawAll ]
     [ l_hierPath ]
     [ l_vertex ]
     )
     => g_hlObjectId / nil
```

Description

Adds a figure to a highlight set.

If *g_drawAll* is `t`, and the figure is an instance, all figures in the master of the instance are drawn.

Arguments

g_hlSetId	Database ID of the highlight set.
d_figId	Database ID of the figure added to the highlight set.
g_drawAll	Specifies whether all figures in the instance are drawn. Valid Values: `t`, `nil`
l_hierPath	Specifies the hierarchal path of a figure. This option is used to highlight figures located inside the hierarchy. The format used for this option is a list of list (`dbInst memInst row column`).
l_vertex	Specifies the Boolean list of vertices of the figure to highlight. This option can be used to partially highlight a figure.

Value Returned

g_hlObjectId	The figure is added to the set.
`nil`	The figure is not added to the set.

Example

```
geAddHilightFig( set1 fig3 nil )
```


© 2020 All Rights Reserved.

Adds `fig3` to the highlight set `set1` and does not draw the objects. Returns the object ID of `fig3`.

```
geAddHilightFig(set1 instId t)
```

Adds `instId` to the highlight set `set1` and highlights the objects inside `instId`.

```
geAddHilightFig(set1 figId nil list(list(inst1 0 0 0) list(inst2 0 0 0)))
```

Adds `figId` to the highlight set `set1`, located in the hierarchy at the given hierpath `inst1/inst2`.

```
geAddHilightFig(set1 rect nil nil list(nil t t nil))
```

Adds `rect` to the highlight set `set1`, but only highlights vertices `2` and `3`.

© 2020 All Rights Reserved.

geAddHilightLabel

```
geAddHilightLabel(
     g_hlSetId
     l_origin
     t_label
     t_just
     t_orient
     t_font
     n_height
     g_draftingP
     [ g_overBar ]
     )
     => g_hlObjectId / nil
```

Description

Adds a label to a highlight set.

l_origin specifies the location of the label. Other attributes of the label are specified by *t_label*, *t_just*, *t_orient*, *t_font*, *n_height*, *g_draftingP*, and *g_overbar*.

Arguments

g_hlSetId	Database ID of the highlight set.
l_origin	Specifies the location of the label.
t_label	Specifies the text of the label.
t_just	Specifies the justification of the label. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`
t_orient	Specifies the orientation of the label. Valid Values: `R0` (no rotation), `R90` (rotate 90 degrees), `R180` (rotate 180), `R270` (rotate 270), `MX` (mirror over the x axis), `MYy` (mirror over the y axis), `MXR90` (mirror over the x axis and rotate 90 degrees), `MYR90` (mirror over the y axis and rotate 90 degrees)
t_font	Specifies the font style used for the label. Valid Values: `europe`, `gothic`, `roman`, `script`, `stick`
n_height	Specifies the height of the label.

© 2020 All Rights Reserved.

g_draftingP	Determines how you can rotate text. `t` allows the text to rotate 90 degrees, `nil` allows rotation in all directions.
g_overBar	Determines whether the label has an over-bar or not.Valid Values: `t`, `nil`Default: `nil`

Value Returned

g_hlObjectId	The label is added to the set.
`nil`	The label is not added to the set.

Example

```
geAddHilightLabel( set1 list(0:0) "mytext" "lowerLeft" "R0" "stick" 2 )
```

Adds a label to the highlight set `set1` with the origin at 0:0. The text is `mytext`, the justification is `lowerleft`, the font style is `stick`, and the height of the text is 2 user units. Returns the object ID of the label.

© 2020 All Rights Reserved.

geAddHilightLine

```
geAddHilightLine(
     g_hlSetId
     l_points
     )
     => g_hlObjectId / nil
```

Description

Adds a line to a highlight set.

The points in *l_points* specify the location of the line.

Arguments

g_hlSetId	Database ID of the highlight set.
l_points	List of points used to describe the line.

Value Returned

g_hlObjectId	The line is added to the set.
nil	The line is not added to the set.

Example

```
geAddHilightLine( set1 list(0:0 10:0 10:10) )
```

Adds a line to the highlight set `set1` and returns the object ID of the line.

© 2020 All Rights Reserved.

geAddHilightPath

```
geAddHilightPath(
     g_hlSetId
     l_points
     n_width
     )
     => g_hlObjectId / nil
```

Description

Adds a path to a highlight set.

The points in *l_points* specify the location of the path, and *n_width* specifies the width of the path.

Arguments

g_hlSetId	Database ID of the highlight set.
l_points	List of points used to describe the polygon.
n_width	Width of the path.

Value Returned

g_hlObjectId	The path is added to the set.
nil	The path is not added to the set.

Example

```
geAddHilightPath( set1 list(0:0 10:10 5:10 0:5) 5 )
```

Adds a path to the highlight set `set1` and returns the object ID of the path.

© 2020 All Rights Reserved.

geAddHilightPathSeg

```
geAddHilightPathSeg(
     g_hlSetId
     l_beginPt
     l_endPt
     n_width
     t_beginStyle
     t_endStyle
     [ l_extValue ]
     => g_hlObjectId / nil
```

Description

Adds a pathSeg to a highlight set.

Arguments

g_hlSetId	Database ID of the highlight set.
l_beginPt	The start point of the path segment.
l_endPt	The end point of the path segment.
n_width	Width of the path.
t_beginStyle	The style for the beginning of the path segment.
t_endStyle	The style for the end of the path segment.
l_extValue	Extension value of a pathSeg. User needs to specify the value set of *n_beginExt*, *n_beginLeftDiagExt*, *n_beginRightDiagExt*, *n_beginRightHalfWidth*, and/or *n_endExt*, *n_endLeftDiagExt*, *n_endRightDiagExt*, *n_endRightHalfWidth* in the argument *l_extValue.*

Value Returned

g_hlObjectId	The pathSeg is added to the set.
nil	The pathSeg is not added to the set.

Example

```
cv = geGetEditCellView()
```


© 2020 All Rights Reserved.

```
=> db:0x4522df92
hl = geCreateHilightSet(cv list("text" "drawing"))
=> hl:0x110a5018
hl->enable = t
=> t
geAddHilightPathSeg(hl list(0.01 4.975) list(2.005 4.975) 0.12 "extend" "extend")
=> hl:0x110a4018
```


© 2020 All Rights Reserved.

geAddHilightPolygon

```
geAddHilightPolygon(
     g_hlSetId
     l_points
     )
     => g_hlObjectId / nil
```

Description

Adds a polygon to a highlight set.

The points in *l_points* specify the location of the polygon.

Arguments

g_hlSetId	Database ID of the highlight set.
l_points	List of points used to describe the polygon.

Value Returned

g_hlObjectId	The polygon is added to the set.
nil	The polygon is not added to the set.

Example

```
geAddHilightPolygon( set1 list(0:0 10:10 5:10 0:5) )
```

Adds a polygon to the highlight set `set1` and returns the object ID of the polygon.

© 2020 All Rights Reserved.

geAddHilightRow

```
geAddHilightRow(
     g_hlSetId
     d_siteDef
     l_point
     x_numSites
     [ t_siteOrient ]
     [ t_rowOrient ]
     => g_hlObjectId / nil
```

Description

Adds a row to a highlight set.

Arguments

g_hlSetId	Database ID of the highlight set.
d_siteDef	The `siteDef` to be used for this row.
l_point	The origin point for the row.
x_numSites	The number of site placements in the row.
t_siteOrient	Specifies the orientation of the site placements of the row. Default value: `R0`
t_rowOrient	Specifies the orientation of the row. Default value: `R0`

Value Returned

g_hlObjectId	The row is added to the set.
`nil`	The row is not added to the set.

© 2020 All Rights Reserved.

geAddHilightRuler

```
geAddHilightRuler(
     g_hlSetId
     l_points
     => g_hlObjectId / nil
```

Description

Adds a ruler to a highlight set.

Arguments

g_hlSetId	Database ID of the highlight set.
l_points	List of points used to describe the ruler.

Value Returned

g_hlObjectId	The ruler is added to the set.
nil	The ruler is not added to the set.

Example

```
geAddHilightRuler(hset list(0:0 5:5) )
```

Adds a ruler to the highlight set `hset` and returns the object ID of the ruler.

© 2020 All Rights Reserved.

geAddHilightSlicedCircle

```
geAddHilightSlicedCircle(
     g_hlSetId
     l_center
     n_radius
     f_startAngle
     f_StopAngle
     )
     => g_hlObjectId / nil
```

Description

Adds a sliced circle to a highlight set.

Arguments

g_hlSetId	Database ID of the highlight set.
l_center	Specifies the center of the sliced circle.
n_radius	Specifies the size of the sliced circle.
f_startAngle	Specifies the start angle of the sliced circle.
f_StopAngle	Specifies the stop angle of the sliced circle.

Value Returned

g_hlObjectId	The sliced circle is added to the set.
nil	The sliced circle is not added to the set.

Example

```
hl = geCreateHilightSet(geGetEditCellView() list("y0" "drawing"))
hl->enable = t
geAddHilightSlicedCircle(hl (10:7) 6 45.0 125.0)
```


© 2020 All Rights Reserved.

geAddHilightSlicedDonut

```
geAddHilightSlicedDonut(
     g_hlSetId
     l_center
     n_radius
     f_startAngle
     f_StopAngle
     )
     => g_hlObjectId / nil
```

Description

Adds a sliced donut to a highlight set.

Arguments

g_hlSetId	Database ID of the highlight set.
l_center	Specifies the center of the sliced donut.
n_radius	Specifies the size of the sliced donut.
f_startAngle	Specifies the start angle of the sliced donut.
f_StopAngle	Specifies the stop angle of the sliced donut.

Value Returned

g_hlObjectId	The sliced donut is added to the set.
nil	The sliced donut is not added to the set.

Example

```
hl = geCreateHilightSet(geGetEditCellView() list("y0" "drawing"))
hl->enable = t
geAddHilightSlicedDonut(hl (1:-20) 6 3  45.0 190.0)
```


© 2020 All Rights Reserved.

geAddHilightTrackPattern

```
geAddHilightTrackPattern(
     g_hlSetId
     d_trackPatternId
     )
     => g_hlObjectId / nil
```

Description

Adds a track pattern to a highlight set.

Arguments

g_hlSetId	Database ID of the highlight set.
d_trackPatternId	The track pattern Id.

Value Returned

g_hlObjectId	The track pattern is added to the set.
nil	The track pattern is not added to the set.

Example

```
win = window(3)
cv=geGetEditCellView(win)
tp = nth(0 cv~>trackPatterns)
hilSet = geCreateHilightSet(cv '("hilite" "drawing")) |
hilSet~>enable=t
geAddHilightTrackPattern(hilSet tp)
```


© 2020 All Rights Reserved.

geAddHilightRectangle

```
geAddHilightRectangle(
     g_hlSetId
     l_bBox
     )
     => g_hlObjectId / nil
```

Description

Adds a rectangle to a highlight set.

The points in *l_bBox* specify the location of the rectangle.

Arguments

g_hlSetId	Database ID of the highlight set.
l_bBox	List of points used to describe the rectangle.

Value Returned

g_hlObjectId	The rectangle is added to the set.
nil	The rectangle is not added to the set.

Example

```
geAddHilightRectangle( set1 list(0:0 10:10) )
```

Adds a rectangle to the highlight set `set1` and returns the object ID of the rectangle.

© 2020 All Rights Reserved.

geChangeHilightPacketName

```
geChangeHilightPacketName(
     h_hlSetId
     g_packetName
     )
     => t / nil
```

Description

Changes the packet on which the given highlight set is drawn.

Arguments

h_hlSetId	The database ID of the highlight set.
g_packetName	The name of the packet.

Value Returned

`t`	The name of the packet is changed.
`nil`	The highlight set ID is invalid.

Example

```
geChangeHilightPacketName(hsetId "annotate")
```


© 2020 All Rights Reserved.

geCreateHilightGroup

```
geCreateHilightGroup(
     h_hlSetId
     [ l_hlObjects ]
     )
     => hlGroupId / nil
```

Description

Creates a group container in the given highlight set.

A group container is used to group together highlight objects of a specified highlight set. Optionally, a parameter list of highlight objects can be specified for the group container.

Arguments

h_hlSetId	The database ID of the highlight set.
l_hlObjects	Optional argument which list highlight objects to be placed in the group container being created.

Value Returned

hlGroupId	The database ID of the group container.
nil	The highlight set ID is invalid.

Example

```
geCreateHilightGroup( hsetId list(hlObject1 hlObject2))
```


© 2020 All Rights Reserved.

geCreateHilightSet

```
geCreateHilightSet(
     d_cellview
     l_layerPurposePair
     [ g_notGlobal ]
     )
     => g_hlSetId / nil
```

Description

Creates a highlight set.

The cellview specified by *d_cellview* must be displayed in a window. The objects will be drawn using the layer specified by *l_layerPurposePair*. The *g_notGlobal* argument is an optional Boolean argument; if it is not specified, the highlight set is added to the global list of the cellview. If the *g_notGlobal* argument is equal to `t`, the highlight is not added to the global list and it must be pushed into the cellview's highlight stack in order for it be drawn. When the highlight set is first created, it is disabled. The application must enable the set and refresh the window to see the highlight objects.

Arguments

d_cellview	Database ID of the cellview in which to create the highlight set.
l_layerPurposePair	Layer-purpose pair used to create the highlight objects.
g_notGlobal	Specifies whether the highlight set is added to the global list. Valid Values: `t`, `nil`

Value Returned

g_hlSetId	The highlight set is created.
`nil`	The highlight set is not created.

Example

```
geCreateHilightSet( cell3 list("layer1" "purpose1") nil )
```

Creates a highlight set in `cell3` on `layer1` with `purpose1` and adds the set to the global list.

© 2020 All Rights Reserved.

geCreateWindowHilightSet

```
geCreateWindowHilightSet(
     w_windowId
     l_layerPurposePair | t_packetName
     [ g_notGlobal ]
     )
     => h_hilightSetId / nil
```

Description

Creates an highlight set on a window.

The highlight set can be created by specifying a layer purpose pair (LPP) or a packet name. If `notGlobal` is set, the highlight set must be at the top of the window highlight stack and enabled in order to be drawn.

Arguments

w_windowId	The ID of the window.
l_layerPurposePair | *t_packetName*	The layer purpose pair (LPP) or a packet name.
g_notGlobal	Optional argument that specifies whether the highlight set is added to the global list. Valid Values: t, nil

Value Returned

h_hilightSetId	The database ID of the highlight set is returned.
`nil`	The LPP of packet name is invalid.

Example

```
geCreateWindowHilightSet( window(3) list("metal1" "drawing") nil)
```


© 2020 All Rights Reserved.

geDeleteAllHilightSet

```
geDeleteAllHilightSet(
     d_cellview
     )
     => t / nil
```

Description

Deletes all highlight sets from the specified window.

Arguments

d_cellview	Database ID of the highlight set.

Value Returned

`t`	All sets are deleted.
`nil`	Sets are not deleted.

© 2020 All Rights Reserved.

geDeleteAllWindowHilightSet

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

Description

Deletes all the window highlight sets registered on the given window.

Arguments

w_windowId	The ID of the window containing the cellview.

Value Returned

`t`	The window highlight sets are deleted.
`nil`	An error occurred and the highlight set are not deleted.

Example

```
geDeleteAllWindowHilightSet(window(3))
```


© 2020 All Rights Reserved.

geDeleteHilightGroup

```
geDeleteHilightGroup(
     hl_groupId
     [ g_deleteAll ]
     )
     => t / nil
```

Description

Deletes the specified highlight group.

Arguments

hl_groupId	The database ID of the highlight group.
g_deleteAll	Optional argument. When t, each highlight object in the group is also deleted unless it belongs to another group. When `nil`, each highlight object in the group is placed back in the highlight set object list unless it belongs to another group. The default is `nil`.

Value Returned

`t`	The specified highlight group is deleted.
`nil`	An error occurred and the highlight group is not deleted.

Example

```
geDeleteHilightGroup(hlGroupId t)
```


© 2020 All Rights Reserved.

geDeleteHilightObject

```
geDeleteHilightObject(
     g_hlObjectId
     )
     => t / nil
```

Description

Deletes the highlight object from the highlight set.

Arguments

g_hlObjectId	Database ID of the highlight object.

Value Returned

`t`	The object is deleted.
`nil`	The object is not deleted.

Example

```
geDeleteHilightObject( object2 )
```

Deletes `object2`.

Deletes `set2`.

© 2020 All Rights Reserved.

geDeleteHilightSet

```
geDeleteHilightSet(
     g_hlSetId
     )
     => t / nil
```

Description

Deletes a highlight set.

The highlight set is removed from any highlight stack that contains it before the set is deleted.

Arguments

g_hlSetId	Database ID of the highlight set.

Value Returned

t	The set is deleted.
nil	The set is not deleted.

Example

```
geDeleteHilightSet( set2 )
```


© 2020 All Rights Reserved.

geDeleteHilightSetInWindows

```
geDeleteHilightSetInWindows(
     hl_groupId
     [ l_windowList ]
     )
     => t / nil
```

Description

Deletes the given window highlight set in all windows where it is referenced, or in the given windows specified in the list.

Arguments

hl_groupId	The database ID of the highlight group.
l_windowList	List of ID for windows in which the highlight set is to be deleted.

Value Returned

`t`	The specified highlight group is deleted.
`nil`	An error occurred and the highlight group is not deleted.

Example

```
geDeleteHilightSetInWindows(hlSetId list(window(3) window(5)))
```


© 2020 All Rights Reserved.

geDrawHilightSet

```
geDrawHilightSet(
     [ w_windowId ]
     [ g_hlSetId ]
     )
     => t / nil
```

Description

Redraws a highlight set from a specified window ID.

If no highlight set is provided, the highlight set from the specified window ID is re-displayed. If no window ID is provided, the highlight set from the current window is re-displayed.

Arguments

w_windowId	The window ID.
g_hlSetId	Database ID of the highlight set.

Value Returned

`t`	The set is redrawn.
`nil`	The set is not redrawn.

© 2020 All Rights Reserved.

geGetCurrentHilightSet

```
geGetCurrentHilightSet(
     d_cellview
     )
     => g_hlSetId / nil
```

Description

Retrieves the current highlight set.

The highlight stack in the cellview is specified by *d_cellview*.

Arguments

d_cellview	Database ID of the cellview containing the highlight stack.

Value Returned

g_hlSetId	The highlight set is retrieved.
nil	The cellview is not valid or the stack is empty.

Example

```
geGetCurrentHilightSet( cell3 )
```

Returns the highlight set ID of the current highlight set in cell3.

© 2020 All Rights Reserved.

geGetCurrentWindowHilightSet

```
geGetCurrentWindowHilightSet(
     w_windowId
     )
     => d_hlSetId / nil
```

Description

Returns the window highlight set which is at the top of the window highlight stack.

Arguments

w_windowId	Database ID of the window.

Value Returned

d_hlSetId	The database ID of the current window highlight set is returned.
nil	The window ID is invalid.

Example

```
geGetCurrentWindowHilightSet(window(3))
```


© 2020 All Rights Reserved.

geGetFigHilightObjectId

```
geGetFigHilightObjectId(
     g_hlSetId
     d_figId
     )
     => g_hlObjectId / nil
```

Description

Returns the highlight object ID of the object *d_figId* in the highlight set *g_hlSetId*.

Arguments

g_hlSetId	Database ID for the highlight set.
d_figId	Database ID for the object.

Value Returned

g_hlObjectId	The highlight object ID for the specified object.
nil	The highlight object ID is not found.

Example

```
geGetFigHilightObjectId( set1 fig2 )
```

Returns the highlight object ID for the object `fig2` in the highlight set `set1`.

© 2020 All Rights Reserved.

geGetHilightObjectFigId

```
geGetHilightObjectFigId(
     h_hlObjectId
     )
     => d_figId / nil
```

Description

Returns the figure ID of the highlighted object.

Arguments

h_hlObjectId	Database ID of the highlighted object.

Value Returned

d_figId	The figure ID of the highlighted object is returned.
nil	The highlight object ID is invalid.

Example

```
geGetHilightObjectFigId(hlObjectId)
```


© 2020 All Rights Reserved.

geInsertHilightObjectsInGroup

```
geInsertHilightObjectsInGroup(
     h_hlGroupId
     [ l_hlObjectId ]
     )
     => t / nil
```

Description

Groups all given highlight objects together.

Arguments

h_hlGroupId	Database ID of the group in which to insert the highlight object.
l_hlObjectId	Optional argument. Database ID of the highlighted object.

Value Returned

`t`	The highlight objects are grouped together.
`nil`	An error occurred.

Example

```
geInsertHilightObjectsInGroup( hlGroup2 list(hlObject1 hlObject2 hlGroup1))
```


© 2020 All Rights Reserved.

geIsValidHilightGroup

```
geIsValidHilightGroup(
     h_hlGroupId
     )
     => t / nil
```

Description

Determines whether the highlighted group is valid.

Arguments

h_hlGroupId	The database ID of the highlight group.

Value Returned

`t`	The highlight group is valid.
`nil`	The highlight group is not valid.

Example

```
geIsValidHilightGroup(hlGroup)
```


© 2020 All Rights Reserved.

geIsValidHilightId

```
geIsValidHilightId(
     h_hlId
     )
     => t / nil
```

Description

This function determines whether a highlight ID is valid.

Arguments

h_hlId	The database ID of the highlight group, set, or object.

Value Returned

`t`	The highlight ID is valid.
`nil`	The highlight ID is not valid.

Example

```
geIsValidHilightId(hlSet)
geIsValidHilightId(hlGroup)
geIsValidHilightId(hlObject)
```


© 2020 All Rights Reserved.

geIsValidHilightObject

```
geIsValidHilightObject(
     h_hlObjectId
     )
     => t / nil
```

Description

Determines whether the highlighted object is valid.

A valid highlighted object has the format of the *g_hlObjectId* value returned from any of the `geAddHilight` functions.

Arguments

h_hlObjectId	Any highlighted object ID returned from any of the `geAddHilight` functions.

Value Returned

`t`	The highlighted object is valid.
`nil`	The highlighted object is not valid.

Examples

```
hlObjectId = geAddHilightPolygon( set1
list( 0:0 10:0 10:10 0:10 ) )
```

```
geIsValidHilightObject( hlObjectId )
```

Returns `t`.

```
geIsValidHilightObject( geGetEditCellView( ) )
```

Returns `nil`.

© 2020 All Rights Reserved.

geIsValidHilightSet

```
geIsValidHilightSet(
     h_hlSetId
     )
     => t / nil
```

Description

Determines whether a highlight set is valid.

A valid highlight set has the format of the *h_hlSetId* value returned from the `geCreateHilightSet` function.

Arguments

h_hlSetId	Any highlight set ID returned from `geCreateHilightSet`.

Value Returned

`t`	The highlight set is valid.
`nil`	The highlight set is not valid.

Examples

```
hlSetId = geCreateHilightSet( geGetEditCellView( )list( "hilite" "drawing" ) )
geIsValidHilightSet( hlSetId )
```

Returns `t`.

```
geIsValidHilightSet( geGetEditCellView( ) )
```

Returns `nil`.

© 2020 All Rights Reserved.

gePopHilightStack

```
gePopHilightStack(
     d_cellview
     )
     => nil
```

Description

Pops the current highlight set out of the highlight stack.

The next highlight set in the stack becomes current. If there are no more highlight sets in the stack, the current highlight set is `nil`.

Arguments

d_cellview	Database ID of the cellview containing the highlight stack.

Value Returned

`nil`	The highlight set is not popped out of the stack.

Example

```
gePopHilightStack( cell1 )
```

Pops the current highlight set out of the stack in `cell1`.

© 2020 All Rights Reserved.

gePopWindowHilightStack

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

Description

Pops the current window highlight set out of the window highlight stack.

The highlight set is removed from the window highlight stack, but not deleted. The highlight set can be pushed back into the stack at a later time.

Arguments

w_windowId	Window ID of the current window.

Value Returned

t	The window highlight set is popped out of the stack.
nil	The window highlight set is not popped out of the stack

Example

```
gePopWindowHilightStack(window(3))
```


© 2020 All Rights Reserved.

gePushHilightStack

```
gePushHilightStack(
     g_hlSetId
     )
     => t / nil
```

Description

Pushes a highlight set into the highlight stack of the cellview to which the highlight belongs.

The highlight set being pushed becomes the current highlight set. If a highlight set is already in the global list of the cellview, it cannot be pushed into the highlight stack.

Arguments

g_hlSetId	Database ID of the highlight set.

Value Returned

`t`	The highlight set is pushed into the highlight stack.
`nil`	The highlight set is not pushed into the highlight stack

Example

```
gePushHilightStack( set13 )
```

Pushes highlight set `set13` into the stack.

© 2020 All Rights Reserved.

geResetHilightSetAlphaBlendingParameters

```
geResetHilightSetAlphaBlendingParameters(
     g_hSetId
     )
     => t / nil
```

Description

Resets the alpha blending values to maximum values (1000) for the given highlight set.

Arguments

g_hSetId	The hilight set ID.

Value Returned

t	The function completed successfully.
nil	The function did not complete successfully.

Example

```
hSetId = geCreateHilightSet(geGetEditCellView() list("Metal1" "drawing"))
geAddHilightRectangle(hSetId list(0:0 5:5))
hSetId->enable=t
geSetHilightSetAlphaBlendingParameters(hSetId 650 650 650)
hiRedraw()
geResetHilightSetAlphaBlendingParameters(hSetId)
hiRedraw()
```

The above example resets the alpha blending values for a highlight set.

© 2020 All Rights Reserved.

geResetHilightSetHaloParameters

```
geResetHilightSetHaloParameters(
     h_hlSetId
     )
     => t / nil
```

Description

Resets the halo parameters set on a highlight set.

Arguments

h_hlSetId	The database ID of the highlight set.

Value Returned

t	Resets the halo parameters set on a highlight set.
nil	The window ID is invalid.

Example

```
geResetHilightSetHaloParameters(highlighSet)
```


© 2020 All Rights Reserved.

geSetHilightSetAlphaBlendingParameters

```
geSetHilightSetAlphaBlendingParameters(
     g_hSetId
     g_halo
     g_fill
     g_outLine
     )
     => t / nil
```

Description

This function sets the alpha blending values for the specified highlight set.

Note: Haloing objects on the screen is expensive and should not be performed on a large set of objects due to the computational and graphics system overhead. The geSetHilightSetHaloParameters and `geSetHilightSetAlphaBlendingParameters` functions provide a mechanism to halo objects on the screen and is intended for use by internal Cadence applications. They should be used sparingly. Do not halo a large number of objects on the screen as it affects performance.

Arguments

g_hSetId	The hilight set ID.
g_halo	The new halo value.
g_fill	The new fill value.
g_outLine	The new outline value.

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

```
hSetId = geCreateHilightSet(geGetEditCellView() list("Metal1" "drawing"))
geAddHilightRectangle(hSetId list(0:0 5:5))
hSetId->enable=t
geSetHilightSetAlphaBlendingParameters(hSetId 650 650 650)
hiRedraw()
```


© 2020 All Rights Reserved.

The above example changes the default alpha blending values for the specified highlight set.

© 2020 All Rights Reserved.

geSetHilightSetHaloParameters

```
geSetHilightSetHaloParameters(
     h_hlSetId
     g_haloPosition
     g_haloType
     g_haloThickNess
     g_haloTransparency
     [ g_drawStroke ]
     )
     => t / nil
```

Description

Sets the halos parameters on the given highlight set in order to draw the set with halos.

Note: Haloing objects on the screen is expensive and should not be performed on a large set of objects due to the computational and graphics system overhead. The `geSetHilightSetHaloParameters` and geSetHilightSetAlphaBlendingParameters functions provide a mechanism to halo objects on the screen and are intended for use by internal Cadence applications. They should be used sparingly. Do not halo a large number of objects on the screen as there will be a negative impact on performance.

© 2020 All Rights Reserved.

Arguments

h_hlSetId	The database ID of the highlight set.
g_haloPosition	The position of the halo. Valid values: `over`, `under`
g_haloType	The halo type. Valid values: `none`, `plain`, `smooth`, `fadeout`, `random`
g_haloThickNess	The halo thickness. Valid values: `normal`, `thin`, `thick`
g_haloTransparency	The transparency value. Valid values: `[0 100]`
g_drawStroke	Optional argument. The drawStroke parameter controls the visibility of the highlight shapes when they have a halo. Valid values: `t` or `nil`

Value Returned

`t`	Parameters of the halo are successfully set.
`nil`	The parameters are not set.

Example

```
geSetHilightSetHaloParameters(hlsetId "under" "smooth" "normal" 50
 nil )
```


© 2020 All Rights Reserved.

geShareWindowHilightSet

```
geShareWindowHilightSet(
     h_hlSetId
     l_windowList
     )
     => t / nil
```

Description

Shares the given window highlight set with other windows.

Arguments

h_hlSetId	The database ID of the highlight set.
l_windowList	List of windows ID in which to share the highlight set.

Value Returned

`t`	The window highlight set is shared with the specified list of windows.
`nil`	The highlight set ID or window IDs are invalid.

Example

```
geShareWindowHilightSet(hlSetId list(window(4) window(5)))
```


© 2020 All Rights Reserved.

Background Cellview Functions

geBackgroundCellView

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

Description

Opens the *Background CellView* form which is used to superimposed cellviews in the same window.

The background cellview is displayed under the current editable cellview. Any cellview from any library can be used as background cellview.

Arguments

None

Value Returned

`t`	Opens the *Background CellView* form.
`nil`	The specified cellview does not exist.

Example

```
geBackgroundCellView()
```


© 2020 All Rights Reserved.

geDisableBackgroundCellView

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

Description

Disables the display of the background cellview for the given window (current window by default).

Arguments

w_windowId	The ID of the window.

Value Returned

t	The display of the background cellview is disabled.
nil	The window ID is invalid.

Example

```
geDisableBackgroundCellView()
```


© 2020 All Rights Reserved.

geEnableBackgroundCellView

```
geEnableBackgroundCellView(
     w_windowId
     [ g_dimmed ]
     )
     => t / nil
```

Description

Enables the display of the background cellview in the specified window. The current window is used by default. If the dimmed parameter is set to `t`, the background cellview is displayed dimmed.

Arguments

w_windowId	The database ID of the window in which to display the background cellview.
g_dimmed	Specifies if the background cellview is displayed dimmed. Valid Values: `t`, `nil`

Value Returned

`t`	The display of the background cellview is successful.
`nil`	The window ID is invalid.

Example

```
geEnableBackgroundCellView(hiGetCurrentWindow() t)
```


© 2020 All Rights Reserved.

geGetBackgroundCellView

```
geGetBackgroundCellView(
     [ w_windowId ]
     )
     => l_cellViewId / l_transform
```

Description

Returns the cellview used for the background cellview.

Arguments

w_windowId	Optional argument, Window ID.

Value Returned

l_cellViewId l_transform	List containing the cellview ID and the transform used for the background.

Example

```
win = hiGetCurrentWindow()
geGetBackgroundCellView(win)
 (db:0x18b9d39a
     ((-25.0 -25.0) "R0") 1.0
 )
```


© 2020 All Rights Reserved.

geResetBackgroundCellView

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

Description

Resets the background cellview on the given window.

Arguments

w_windowId	Optional argument of the ID of the window.

Value Returned

`t`	Resets the background cellview on the given window.
`nil`	The window ID is invalid.

Example

```
geResetBackgroundCellView(window(3))
```


© 2020 All Rights Reserved.

geSetBackgroundCellView

```
geSetBackgroundCellView(
     w_windowId
     d_cellViewId | l_libName, cellName, viewName
     [ l_transform ]
     )
     => t / nil
```

Description

Sets a background cellview on specified window. A cellview ID or a list of *libName*, *cellName*, and *viewName* can be given.

Arguments

w_windowId	Window ID.
d_cellViewId | *l_libName, cellName, viewName*	Cellview ID or a list specifying the libName, cellName, or viewName.
l_transform	Optional argument that will place the background cellview at a specified location and rotation.

Value Returned

`t`	The background is set to the specified cellview.
`nil`	The cellview ID or specified list is invalid.

Example

```
geSetBackgroundCellView(window(3) list( "myLib" "myCell" "layout")
```

The following example places the background cellview at location `5:10` with a rotation `R90`

```
geSetBackgroundCellView(winId list(libname cellname viewname) list(5:10 "R0"))
```

or

```
geSetBackgroundCellView(winId cvId  list(5:10 "R0"))
```


© 2020 All Rights Reserved.

geZoomToBackgroundAndEditCV

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

Description

When a background cellview is set on the window, zooms to fit the background cellview and the current edit cellview. If a background cellview is not set on the window, zooms to fit the current edit cellview.

Arguments

w_windowId	Optional argument of the ID of the window.

Value Returned

t	Zooms to fit the background/edit cellview.
nil	The specified window does not exist.

Example

```
geZoomToBackgroundAndEditCV(window(3))
```


© 2020 All Rights Reserved.

geZoomToBackgroundCV

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

Description

When a background cellview is set on the window, zooms to fit the background cellview.

Arguments

w_windowId	Optional argument of the ID of the window.

Value Returned

`t`	Zooms to fit the background window.
`nil`	The specified window does not exist.

Example

```
geZoomToBackgroundCV(window(3))
```


© 2020 All Rights Reserved.

Marker Functions

geCreateMarker

```
geCreateMarker(
     d_dbobjectId
     t_severity
     t_owner
     t_reason
     t_displayText
     )
     => d_shapeId / nil
```

Description

Creates a box-shaped marker whose size and position are determined by the software.

The marker shape is placed on the marker layer with error purpose if *t_type* is `error`. Otherwise, the marker shape is placed on the marker layer with warning purpose. If *d_dbobjectId* is a cellview, the marker shape is a box at the lower left corner of the cellview. Otherwise, the marker shape is a box that surrounds the object *d_dbobjectId* as described below and glues the marker to the object.

figId	The bounding box of the shape is used for the marker.
instId	The bounding box of the instance is used for the marker.
instHeaderId	The bounding box of the first instance for the instance header is used for the marker.
instTermId	The bounding box of the figure representing the first pin found on the instance terminal is used. If no pin exists on the instance terminal, a generic box at the origin of the instance is used.
netId	The bounding box of the figures attached to the net is used for the marker. If no figures exist on the net, a generic box in the lower left corner of the design is used.
pinId	The bounding box of the figure representing the pin is used for the marker.
termId	The bounding box of the figure representing the first pin found on the terminal is used. If no pin exists on the terminal, a generic box in the lower left corner of the design is used.

© 2020 All Rights Reserved.

After the marker is created, the *t_owner*, *t_reason*, and *t_displayText* values are stored as properties on the marker. The *t_owner* value identifies the tool that identified the problem and is creating the marker. The *t_reason* value is the warning/error message that describes the problem causing the marker. The *t_displayText* value is a brief warning/error message that is displayed with the marker shape.

Arguments

d_dbobjectId	Database ID of the cellview to contain the marker.
t_severity	Severity of the marker. Valid Values: `warning`, `error`
t_owner	Owner of the marker. Valid Values: any valid owner name
t_reason	Reason for the violation.
t_displayText	Text associated with the marker.

Value Returned

d_shapeId	The shape representing the marker, if the marker is created.
`nil`	The marker is not created.

Example

```
geCreateMarker( cell2 "warning" "PDV" "physical rule violation" "mytext" )
```

Creates a marker in the cellview `cell2`. The marker severity is `warning` and the marker owner is *PDV*. `physical rule violation` is the reason for the marker. `Mytext` is displayed with the marker. Returns the database ID of the marker shape.

© 2020 All Rights Reserved.

geCreateMarkerByBBox

```
geCreateMarkerByBBox(
     d_cellview
     t_type
     t_owner
     t_reason
     t_displayText
     l_bBox
     )
     => d_shapeId / nil
```

Description

Creates a rectangular marker whose size you define with *l_bBox.*

Arguments

d_cellview	Database ID of the cellview to contain the marker.
t_type	Severity of the marker. Valid Values: `warning`, `error`
t_owner	Owner of the marker. Valid Values: `DLE`, `PDV`, or any marker owner
t_reason	Reason for violation.
t_displayText	Message displayed with the marker.
l_bBox	Bounding box that defines the marker shape.

Value Returned

d_shapeId	The shape representing the marker, if the marker is created.
`nil`	The marker is not created.

Example

```
geCreateMarkerByBBox( cell2 "warning" "PDV" "poly-poly spacing violation" "mytext"
list(12:10 20:18) )
```

Creates a marker in the cellview `cell2` using the coordinates in the list. The marker severity is `warning` and the marker owner is `PDV`. `Poly-poly spacing violation` is the reason

© 2020 All Rights Reserved.

for the marker. `Mytext` is displayed with the marker. Returns the shape ID of the marker shape.

© 2020 All Rights Reserved.

geCreateMarkerByPoints

```
geCreateMarkerByPoints(
     d_cellview
     t_type
     t_owner
     t_reason
     t_displayText
     l_points
     )
     => d_shapeId / nil
```

Description

Creates a polygon-shaped marker.

Arguments

d_cellview	Database ID of the cellview to contain the marker.
t_type	Severity of the marker. Valid Values: `warning`, `error`
t_owner	Owner of the marker. Valid Values: any marker owner
t_reason	Reason for violation.
t_displayText	Message displayed with the marker.
l_points	List of points defining the marker.

Value Returned

d_shapeId	The shape representing the marker, if the marker is created.
`nil`	The marker is not created.

Example

```
geCreateMarkerByPoints( cell2 "warning" "PDV" "metal1-metal1 spacing violation"
"mytext" list(12:10 20:10 20:18 10:18 10:14 12:14) )
```

Creates a marker in the cellview `cell2` using the coordinates in the list. The marker severity is `warning` and the marker owner is *PDV*. `Mytext` is displayed with the marker. Returns the shape ID of the marker shape.

© 2020 All Rights Reserved.

geDeleteAllMarker

```
geDeleteAllMarker(
     d_cellview
     [ l_owner ]
     [ l_severity ]
     [ g_traverse ]
     )
     => t / nil
```

Description

Deletes all markers in *d_cellview* that have owner *l_owner* and severity *l_severity*. If *l_owner* is specified, *l_severity* must be specified. All markers in the specified traversal hierarchy are deleted, unless *g_traverse* argument is specified as `nil`.

Arguments

d_cellview	Database ID of the cellview containing the markers.
l_owner	List of marker owners. Valid Values: any valid owner name
l_severity	List of marker severities. Valid Values: `warning`, `error`
g_traverse	Specifies the traversal hierarchy, starting from the cellview. Specify this value as `nil` if you do not want the function to traverse the hierarchy.

Value Returned

`t`	The markers are deleted.
`nil`	The markers are not deleted.

Example

```
geDeleteAllMarker( cell3 list("DLE" "PDV") "error" nil )
```

In this example, the function deletes all markers in `cell3` that are of severity `error` and owned by `DLE` or `PDV` but does not traverse the hierarchy because the last argument is *nil*.

```
geDeleteAllMarker( cvId nil nil nil)
```


© 2020 All Rights Reserved.

In this example, the function deletes the markers only in the specified cellview (`cvId`).

© 2020 All Rights Reserved.

geGetMarkerDisplayText

```
geGetMarkerDisplayText(
     d_figId
     )
     => t_string / nil
```

Description

Displays the text associated with a marker.

Arguments

d_figId	Database ID of the marker.

Value Returned

t_string	The text associated with the marker, if the marker text is found.
nil	The marker text is not found.

Example

```
geGetMarkerDisplayText( marker2 )
```

Returns the text associated with `marker2`.

© 2020 All Rights Reserved.

geGetMarkerOwner

```
geGetMarkerOwner(
     d_figId
     )
     => t_string / nil
```

Description

Displays the owner of a marker.

Arguments

d_figId	Database ID of the marker.

Value Returned

t_string	The name of the marker owner.
nil	The marker is not found.

Example

```
geGetMarkerOwner( d_figId )
```

Returns DLE, the name of the marker owner.

© 2020 All Rights Reserved.

geGetMarkerReason

```
geGetMarkerReason(
     d_dbobjectId
     )
     => t_reason / nil
```

Description

Displays the reason for a marker.

Arguments

d_dbobjectId	Database ID of the marker.

Value Returned

t_reason	The marker reason.
nil	The marker is not found.

Example

```
geGetMarkerReason( marker2 )
```

Returns the reason for `marker2`.

© 2020 All Rights Reserved.

geGetMarkerType

```
geGetMarkerType(
     d_figId
     )
     => t_type / nil
```

Description

Displays the type of a marker.

Arguments

d_figId	Database ID of the marker.

Value Returned

t_type	The marker type, either `warning` or `error`.
`nil`	The marker is not found.

Example

```
geGetMarkerType( marker2 )
```

Returns the marker type of `marker2`.

© 2020 All Rights Reserved.

geHasMarker

```
geHasMarker(
     d_cellview
     [ l_owner ]
     [ l_type ]
     )
     => t / nil
```

Description

Queries whether the cellview contains any markers with owner *l_owner* and type *l_type*.

Arguments

d_cellview	Database ID of the cellview queried.
l_owner	Owner of the markers to query. Valid Values: any marker owner
l_type	Type of markers to query. Valid Values: `warning`, `error`

Value Returned

`t`	The cellview contains the specified markers.
`nil`	The cellview does not contain the specified markers.

Example

```
geHasMarker( cell2 "PDV" "error" )
```

Checks whether `cell2` contains any markers owned by `PDV` of the type `error`.

© 2020 All Rights Reserved.

geHiCommonFindMarker

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

Description

Opens the Find Marker form and highlights the current marker in the current window.

Arguments

None

Value Returned

`t`	The form opens and gives marker status.
`nil`	The function failed.

© 2020 All Rights Reserved.

geHiDeleteAllMarker

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

Description

Displays a form that prompts you to select the markers to be deleted from the current cellview.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The function did not succeed.

© 2020 All Rights Reserved.

geHiDeleteMarker

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

Description

Deletes the current marker in the current window.

Arguments

None

Value Returned

`t`	The current marker is deleted.
`nil`	The current marker is not deleted.

© 2020 All Rights Reserved.

geHiExplainMarker

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

Description

Explains the current marker in the current cellview.

Arguments

None

Value Returned

`t`	The current marker is explained.
`nil`	The current marker is not explained.

© 2020 All Rights Reserved.

geHiFindMarker

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

Description

Highlights the current marker in the current window.

Arguments

None

Value Returned

`t`	The current marker is found.
`nil`	The current marker is not found.

© 2020 All Rights Reserved.

geIsMarkerShape

```
geIsMarkerShape(
     d_dbobjectId
     )
     => t / nil
```

Description

Checks whether a shape was created as a marker.

Arguments

d_dbobjectId	Database ID of the cellview containing the shape.

Value Returned

`t`	The shape is a marker.
`nil`	The shape is not a marker.

Example

```
geIsMarkerShape( shape2 )
```

Checks if `shape2` is a marker.

© 2020 All Rights Reserved.

Probing Functions

This section describes the concepts used in the probing functions. The probing functions available in the SKILL level are also described.

Probing allows you to find and highlight certain types of objects. These objects can be nets, terminals, and instances. When two terminals are probed together, the flight line (objects between the terminals) is shown as well. Probing is a way to show the logical connectivity between objects or to show objects that can have correspondence with other objects in a different representation of the same design.

The following is a graphic description of the logical data model.

© 2020 All Rights Reserved.

The path to the probe is an important concept. Two probes are the same if the *memInst* text string path to the probes are equal and if the top-level cellviews of each probe are the same.

A switch was done on C.

A, B, C, D, E are all layout views.

The probes are **not** the same since the top-level cellviews are not equal (views are different). The probes in the schematic and layout do correspond (useful for cross-probing applications).

Probe1 and Probe2 are not equal even though the same physical object was probed because the paths to the object are different.

The probe spans through the hierarchy. Thus, when a probe is done in a cell that has been descended into, the probe is still drawn at other levels of the hierarchy, for example, when you go up the hierarchy. Also, the probe shows in multiple windows even if the windows are at different hierarchical levels of the same top-level design. If the object probed is at a lower level

© 2020 All Rights Reserved.

than you are currently displaying, the instance containing the object probed is highlighted so you know which instances contain probes.

© 2020 All Rights Reserved.

Probe Types

There are four types of probes: net, instance, path, and terminal.

Net Probe

The *net probe* draws a user-specified net. There are two display modes associated with net probes. The flight lines of the net can be displayed so that a spanning tree contains all the terminals in the net. The other display mode is to display the objects in this net.

Net Probe with Flight line Display

Net Probe with Object Display

© 2020 All Rights Reserved.

Instance Probe

The *instance probe* highlights the user-specified instance. When you are at a level above this instance, the instance(s) in all equivalent windows that containing the chosen instance are highlighted to show that they contain a probe.

Instance Probe

Path Probe

The *path probe* draws a flight line between two user specified terminals. These terminals do not have to be on the same net. This also means that the terminals do not have to be physically connected.

Path Probe

© 2020 All Rights Reserved.

Terminal Probe

The *terminal probe* draws the user specified terminal. When you descend into or edit in place the instance containing this terminal, all the corresponding pins of the master are highlighted.

Terminal Probe

The *probe object* holds all the necessary information for finding the probe in the hierarchy as well as information about displaying the probe. The probe is a combination of the probe object and the display arguments or attributes.

Each probe has only one display argument associated with it. A *multiple probe* is needed if you want more than one display argument. The multiple probe is created when you specify a probe with the same path as one previously specified. You only need to specify the display arguments needed. With the possibility of multiple probes in the system, a unique probe is identified by the *pathname* to the probe and the probe name and the probe argument type combination in the display arguments.

This *probe object* is logically described below.

Probe Object
pathList
pathObjectName
objectId
probeType
probeDisplayStyle (only for net probes)
displayArgs

DisplayArgs Properties
probeName
probeArgType

Choose one of the following:

probeColor
probeLabel
probeLabelName
probeLabelColor
probeLabelOffset

© 2020 All Rights Reserved.

probeLabelFont
probeLabelFontSize
probeLabelJust
probeLabelOrient
probeOnColorWhenDisplay

The *pathname* of the probe is a list of strings that describe how to get to the probed object from the top level. The *pathname* is derived by combining the *pathList* and *pathObjectName* attributes of the *probe object*. The *pathList* is a list of strings with each string being the instance name (in *memInst* format) of the instance used to get to the *probe object* and not including the name of the object itself. The *memInst* format is "instance_name" for instances, "instance_name<iteration_number>" for iterated instances, and "instance_name(row,column)" for mosaics.

The *pathObjectName* is the name of the *probe object* at the level it was probed. The *pathObjectName* is useful for describing an object that can be represented by different database IDs in different representations.

The *objectId* is the database ID of the object in the representation being probed.

The *probeType* can be a path, net, terminal, or instance. These types have been described earlier.

The *probeDisplayStyle* is either the *flight* or *object* display style and has meaning only for net probes. *Flight* display style displays the flight lines between terminals. For net probes, the flight lines that span all the terminals of a net once are displayed. Path probes have a similar concept to flight display style because a flight line is drawn between the two terminals (which do not have to be on the same net). For *object* display style (which can apply to all the types of probes), the physical objects in the probe are highlighted.

The *displayArgs* is a *logical* property list containing the various display attributes associated with the probe object. Listed here are the properties that are provided.

The *probeName* is an attribute that gives a name to a display argument of the probe. This name is used to identify what the display argument does. Examples of such names are "resistance" and "capacitance."

The *probeArgType* identifies the type of display argument used. There are three possibilities: "label," "color," and "onColor."

The *probeColor* is the color (or layer-purpose pair) that the probed objects are to be displayed on. All net objects, terminals, and instances of the probe are highlighted in this color.

© 2020 All Rights Reserved.

The *probeLabel* is a label that is to be displayed when the probe is displayed. One of the important attributes of the *probeLabel* is the *probeLabelOffset*, which is the offset from the default location that the *probeLabelName* is to be drawn. If no offset is specified, the default location is calculated with an algorithm based on the type of probe. The default setting for instance probes is in the center of the instance bounding box. For net probes, the label displays by default at the center of the longest flight line for *flight probeDisplayStyle* or at the center of the largest segment of the net for *object probeDisplayStyle*. The terminal probe has the label at the center of the terminal. Path probes display by default at the center of the flight line. The *probeLabelColor*, *probeLabelFont*, *probeLabelFontSize*, *probeLabelJust*, and *probeLabelOrient* attributes all describe how the *probeLabelName* displays.

The *probeOnColorWhenDisplay* is a color (or layer-purpose pair) turned on for visibility so that preexisting icons in this color can be displayed when an instance probe is displayed.

SKILL Interface

This section describes the SKILL interface to the probing functionality available in the Graphics Editor. Probing is a mechanism for showing items that are logically equivalent. For example, all the objects in a net are highlighted because they are all logically connected.

Most of the attributes of a probe object are accessible and can be set through SKILL by using the -> access operator. For example,

```
probeList = geGetAllProbe(window(3))
foreach(p probeList
println(p—>pathObjectName) ;print probe obj names
)
```

The attributes that can be set by the SKILL access operator -> are restricted to *probeName* *probeColor*, *probeLabel* and other *probeLabel* attributes, *probeDisplayStyle*, and *probeOnColorWhenDisplay* because the other attributes are fixed.

You can add your own properties on the probe by using a new property and setting it to a value. For example, `p->myProperty = "critical"` adds the property *myProperty* to the probe *p* and sets it to the string “critical.” Property names with the prefix probe are reserved since there might be additional attributes available in the future.

When `println(p->pathObjectName)` is done in the example above, all the accessible attributes are listed with their values. All of these attributes vary with the types of probe:

```
probeName
pathList
pathObjectName
topLevelCellViewId
objectId
probeType
```


© 2020 All Rights Reserved.

```
probeArgType
probeColor
probeLabel
probeLabelColor
probeLabelOffset
probeLabelFont
probeLabelFontSize
probeLabelJust
probeLabelOrient
probeOnColorWhenDisplay
probeDisplayStyle
```


© 2020 All Rights Reserved.

geAddInstProbe

```
geAddInstProbe(
     [ w_windowId ]
     [ l_lpp ]
     [ lt_probeSpec ]
     )
     => o_probe
```

Description

Adds an instance probe to a window.

The instance probe is displayed using the attributes of layer-purpose pair *l_lpp*. If *l_lpp* is defaulted, a layer-purpose pair is chosen from the reserved layers y0 through y9 with purpose "drawing."

These default probing layers are cycled through so that successive probes can be highlighted in different colors. Any of the geAdd*Probe functions causes the layer to cycle to the next one. geSetProbeUserLPP overrides this layer rotation. See the description of that function.

lt_probeSpec is either a point or a string. If a point is specified, the instance at that location is probed. If a string is specified, the instance with that hierarchical path name is probed. The string should be in the format of "*/instname/instname/ .../objectName*." If *lt_probeSpec* is not specified, you are prompted to select an instance to probe.

© 2020 All Rights Reserved.

Arguments

w_windowId	Database ID of the window in which to add the probe. The current window is used if *w_windowId* is defaulted.
l_lpp	Layer-purpose pair on which to display the probe.
lt_probeSpec	Location in user units to probe at, or a string describing the probed object.

Value Returned

o_probe	The probe object created.

Example

```
m = geAddInstProbe( window(3) list( "y0" "drawing" ) )
```

Prompts you for a point, and creates an instance probe in `window(3)` using layer `(y0 drawing)`.

© 2020 All Rights Reserved.

geEnterAddInstProbe

```
geEnterAddInstProbe(
     [ w_windowId ]
     [ l_lpp ]
     [ lt_probeSpec ]
     [ g_repeatCmd ]
     )
     => o_probe / l_probes
```

Description

Adds an instance probe to a window. The instance probe is displayed using the attributes of layer-purpose pair *l_lpp*.

If *g_repeatCmd* is `true`, then all instances of the current selection set are probed and highlighted with a different color. If no selection set exists then multiple post-selection is enabled to probe instances. If however, *g_repeatCmd* is `nil` a single probe is added on an instance.

Arguments

w_windowId	Database ID of the window in which to add the probe. The current window is used if *w_windowId* is defaulted.
l_lpp	Layer-purpose pair on which to display the probe.
lt_probeSpec	Location in user units to probe at, or a string describing the probed object.
g_repeatCmd	SKILL expression specifying the probe selection mode (single select versus multiple post-select.)

Value Returned

o_probe	The probe object created.
l_probes	A list of all the probes in *w_windowId*.

Example

```
m = geEnterAddInstProbe( hiGetCurrentWindow() list( "y0" "drawing" ) nil t )
```


© 2020 All Rights Reserved.

Prompts you for a point, and creates an instance probe in the current window using layer `(y0 drawing)`

© 2020 All Rights Reserved.

geGetAdjustedPath

```
geGetAdjustedPath(
     w_winId
     t_pathList
     [ d_cellview ]
     [ t_viewnameList ]
     )
     => t_adjustedPathList
```

Description

Reduces the given hierarchical net path to the shortest hierarchical name that is equivalent to this net.

Arguments

w_winId	Window ID used to find the top-most point of the hierarchy for the name reduction.
t_pathList	A single concatenated string for the instance hierarchy with `"/"` as the hierarchy separator in the string in version 4.4.3 of design framework II. (In version 4.3.4, this was a list of strings of instance names for the instance hierarchy.)
d_cellview	Database Id of a cellview to be used as the top-level cellview specified by *w_winId*. If this value is specified, the minimized name is relative to this cellview, rather than the topmost cellview. Specifying the value returned by `geGetTopCellViewName` as this third argument is the same as omitting the third argument.
t_viewnameList	A string specifying views to switch into when descending the hierarchy specified in *t_pathList*. If not specified, the default value of "`schematic`" is used.

Value Returned

t_adjustedPathList

The reduced net name. If the net is local to this cellview only, the reduced net name is the same as the provided net name.

© 2020 All Rights Reserved.

Example

```
netpath = buildString(foreach(mapcar i
geGetHierMemInst(hiGetCurrentWindow()) car(i)->name) "/" )
netPath = strcat("/" netPath "/" "gnd")
geGetAdjustedPath( hiGetCurrentWindow() netPath)
=> "/gnd"
```

Assumes the currently displayed cellview has a net called `"gnd"`. This code reduces this name to the smallest hierarchically correct name to reach this net. Since the `gnd` net is connected from this point up to the top level of hierarchy, the return value is `"/gnd"`.

© 2020 All Rights Reserved.

geAddNetProbe

```
geAddNetProbe(
     [ w_windowId ]
     [ l_lpp ]
     [ lt_probeSpec ]
     )
     => o_probe
```

Description

Adds a net probe to a window.

The net probe is displayed using the attributes of layer-purpose pair *l_lpp*. If *l_lpp* is defaulted, a layer-purpose pair is chosen from the reserved layers y0 through y9 with purpose drawing. These default probing layers are cycled through so that successive probes can be highlighted in different colors. Any of the `geAdd*Probe` functions causes the layer to cycle to the next one. `geSetProbeUserLPP` overrides this layer rotation. See the description of that function.

lt_probeSpec is either a point or a string. If a point is specified, the instance at that location is probed. If a string is specified, the instance having that hierarchical path name is probed. The string should be in the format of "*/instname/instname/ .../objectName*."

If *lt_probeSpec* is defaulted, you are prompted to select an instance to probe. This the display of the net probe is affected by the net stop level.

Note: You can select the specific bit of a bus by specifying the `probePromptBusBit` environment variable.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is defaulted.
l_lpp	Layer-purpose pair on which to display the probe.
lt_probeSpec	Location in user units to probe at, or a string describing the probed object.

Value Returned

o_probe	Probe object created.

© 2020 All Rights Reserved.

Example

```
m = geAddNetProbe( window(3) list( "y0" "drawing" ) )
```

Prompts for a point, and creates a net probe in `window(3)` using layer `(y0 drawing)`.

© 2020 All Rights Reserved.

geEnterAddNetProbe

```
geEnterAddNetProbe(
     [ w_windowId ]
     [ l_lpp ]
     [ lt_probeSpec ]
     [ g_repeatCmd ]
     )
     => o_probe
```

Description

Adds a net probe to a window.

The net probe is displayed using the attributes of layer-purpose pair *l_lpp*. If *l_lpp* is defaulted, a layer-purpose pair is chosen from the reserved layers y0 through y9 with purpose drawing. These default probing layers are cycled through so that successive probes can be highlighted in different colors.

lt_probeSpec is either a point or a string. If a point is specified, the instance at that location is probed. If a string is specified, the instance having that hierarchical path name is probed. The string should be in the format of "*/instname/instname/ .../objectName*."

If *lt_probeSpec* is defaulted, you are prompted to select an instance to probe.

Note: The display of the net probe is affected by the net stop level.

If *g_repeatCmd* is `true`, then all nets of the current selection set are probed and highlighted with a different color. If no selection set exists, then multiple post-selection is enabled to probe nets. If however, *g_repeatCmd* is `nil`, a single probe is added on a net.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is defaulted.
l_lpp	Layer-purpose pair on which to display the probe.
lt_probeSpec	Location in user units to probe at, or a string describing the probed object.
g_repeatCmd	SKILL expression specifying the probe selection mode (single select versus multiple post-select.)

© 2020 All Rights Reserved.

Value Returned

o_probe	Probe object created.
l_probes	A list of all the probes in *w_windowId*.

Example

```
m = geEnterAddNetProbe( hiGetCurrentWindow() list( "y0" "drawing" ) nil t )
```


© 2020 All Rights Reserved.

geAddPathProbe

```
geAddPathProbe(
     [ w_windowId ]
     [ l_lpp ]
     [ lt_probeSpec1 ]
     [ lt_probeSpec2 ]
     )
     => o_probe
```

Description

Adds a path probe to a window.

The path probe is displayed using the attributes of layer-purpose pair *l_lpp*. If *l_lpp* is defaulted, a layer-purpose pair is chosen from the reserved layers y0 through y9 with purpose drawing. These default probing layers are cycled through so that successive probes can be highlighted in different colors. Any of the `geAdd*Probe` functions causes the layer to cycle to the next one. `geSetProbeUserLPP` will override this layer rotation.

lt_probeSpec1 is either a point or a string. If a point is specified, then the terminal at that location becomes the first of the two terminals in the path probe. If a string is specified, then the terminal having that hierarchical path name is the first terminal. The string should be in the format of "*/instname/instname/ .../objectName*." If *lt_probeSpec1* is defaulted, you are prompted to select the first terminal to probe. Similarly, *lt_probeSpec2* either selects the second terminal, or if defaulted, causes you to be prompted to point at the second terminal.

© 2020 All Rights Reserved.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is defaulted.
l_lpp	Layer-purpose pair on which to display the probe.
lt_probeSpec1	Location in user units to look for the first terminal of the path at, or a string describing the first terminal.
lt_probeSpec2	Location in user units at which to look for the second terminal of the path, or a string describing the second terminal.

Value Returned

o_probe	Probe object created.

Example

```
m = geAddPathProbe( window(3) )
```

Prompts for two points, and creates a path probe between the two terminals in `window(3)` using the next layer in the probe layer rotation sequence.

© 2020 All Rights Reserved.

geAddTermProbe

```
geAddTermProbe(
     [ w_windowId ]
     [ l_lpp ]
     [ lt_probeSpec ]
     )
     => o_probe
```

Description

Adds a terminal probe to a window.

The terminal probe is displayed using the attributes of layer-purpose pair *l_lpp*. If *l_lpp* is defaulted, a layer-purpose pair is chosen from the reserved layers y0 through y9 with purpose drawing. These default probing layers are cycled through so that successive probes can be highlighted in different colors. Any of the `geAdd*Probe` functions causes the layer to cycle to the next one. `geSetProbeUserLPP` overrides this layer rotation. See the description of that function.

lt_probeSpec is either a point or a string. If a point is specified, then the instance at that location is probed. If a string is specified, then the instance having that hierarchical path name is probed. The string should be in the format of "*/instname/instname/ .../objectName*." If *lt_probeSpec* is not specified, you are prompted to select a terminal to probe.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
l_lpp	Layer-purpose pair on which to display the probe.
lt_probeSpec	Location in user units to probe at, or a string describing the probed object.

Value Returned

o_probe	The probe object created.

Example

```
m = geAddTermProbe( window(3) )
```


© 2020 All Rights Reserved.

Prompts for a point, and creates a terminal probe in `window(3)` using the next layer in the probe layer rotation sequence.

© 2020 All Rights Reserved.

geEnterAddTermProbe

```
geEnterAddTermProbe(
     [ w_windowId ]
     [ l_lpp ]
     [ lt_probeSpec ]
     [ g_repeatCmd ]
     )
     => o_probe
```

Description

Adds a terminal probe to a window.

The terminal probe is displayed using the attributes of layer-purpose pair *l_lpp*.

If *l_lpp* is defaulted, a layer-purpose pair is chosen from the reserved layers y0 through y9 with purpose drawing. These default probing layers are cycled through so that successive probes can be highlighted in different colors.

lt_probeSpec is either a point or a string. If a point is specified, then the instance at that location is probed. If a string is specified, then the instance having that hierarchical path name is probed. The string should be in the format of "*/instname/instname/ .../objectName*." If *lt_probeSpec* is not specified, you are prompted to select a terminal to probe.

If *g_repeatCmd* is `true`, then all terminals of the current selection set are probed and highlighted with a different color. If no selection set exists then multiple post-selection is enabled to probe terminals. If however, *g_repeatCmd* is `nil` a single probe is added on a terminal.

© 2020 All Rights Reserved.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
l_lpp	Layer-purpose pair on which to display the probe.
lt_probeSpec	Location in user units to probe at, or a string describing the probed object.
g_repeatCmd	SKILL expression specifying the probe selection mode (single select versus multiple post-select.)

Value Returned

o_probe	Probe object created.
l_probes	A list of all the probes in *w_windowId*.

Example

```
m = geEnterAddTermProbe( hiGetCurrentWindow() list( "y0" "drawing" ) nil t )
```


© 2020 All Rights Reserved.

geDeleteAllProbe

```
geDeleteAllProbe(
     [ w_windowId ]
     [ g_crossProbeFlag ]
     )
     => t / nil
```

Description

Removes all the probes from a window.

If *g_crossProbeFlag* is not `nil`, all the cross probes associated with the probes in *w_windowId* are deleted.

Arguments

w_windowId	Database ID of the window from which to remove probes.
g_crossProbeFlag	If set, all cross probes associated with the probes in a window are also deleted.

Value Returned

`t`	The probes are deleted.
`nil`	The probes are not deleted.

Example

```
geDeleteAllProbe( hiGetCurrentWindow( ) )
```

Deletes all probes in the current window.

© 2020 All Rights Reserved.

geDeleteInstProbe

```
geDeleteInstProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     )
     => t / nil
```

Description

Deletes an instance probe from a window.

lt_probeSpec is the same type of probe specification used for adding a probe with the geAdd*Probe functions.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for an instance probe to delete, or a string describing the instance probe to delete.

Value Returned

t	An instance probe was found and deleted.
nil	An instance probe was not found and deleted.

Example

```
m = geDeleteInstProbe( window(3) )
```

Prompts for a point and deletes the instance probe under that point.

© 2020 All Rights Reserved.

geEnterDeleteInstProbe

```
geEnterDeleteInstProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     [ g_repeatCmd ]
     )
     => t / nil
```

Description

Deletes an instance probe from a window.

lt_probeSpec is the same type of probe specification used for adding a probe with the `geEnterAdd*Probe` functions.

If *g_repeatCmd* is `true`, then delete all instances of the current selection set and highlight with a different color. If no selection set exists, then enable multiple post-selection to delete probe on instances. If however, *g_repeatCmd* is `nil` then delete a probe on an instance.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for an instance probe to delete, or a string describing the instance probe to delete.
g_repeatCmd	SKILL expression specifying the probe selection mode (single select versus multiple post-select.)

Value Returned

`t`	An instance probe was found and deleted.
`nil`	An instance probe was not found and deleted.

Example

```
m = geEnterDeleteInstProbe( hiGetCurrentWindow() nil t)
```

Prompts for a point and deletes the instance probe under that point.

© 2020 All Rights Reserved.

geDeleteNetProbe

```
geDeleteNetProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     )
     => t / nil
```

Description

Deletes a net probe from a window.

lt_probeSpec is the same type of probe specification used for adding a probe with the geAdd*Probe functions.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for a net probe to delete, or a string describing a net probe to delete.

Value Returned

t	A net probe was found and deleted.
nil	A net probe was not found and deleted.

Example

```
m = geDeleteNetProbe( window(3) "/I1/NET10" )
```

Deletes the net probe associated with NET10 in instance I1 of the top level of window(3).

© 2020 All Rights Reserved.

geEnterDeleteNetProbe

```
geEnterDeleteNetProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     [ g_repeatCmd ]
     )
     => t / nil
```

Description

Deletes a net probe from a window.

lt_probeSpec is the same type of probe specification used for adding a probe with the `geAdd*Probe` functions.

If *g_repeatCmd* is `true`, then delete all nets of the current selection set and highlight with a different color. If no selection set exists, then enable multiple post-selection to delete probe on nets. If however, *g_repeatCmd* is `nil` then delete a probe on a net.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for a net probe to delete, or a string describing a net probe to delete.
g_repeatCmd	SKILL expression specifying the probe selection mode (single select versus multiple post-select.)

Value Returned

`t`	A net probe was found and deleted.
`nil`	A net probe was not found and deleted.

Example

```
m = geEnterDeleteNetProbe( hiGetCurrentWindow() nil t )
```


© 2020 All Rights Reserved.

geDeletePathProbe

```
geDeletePathProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     )
     => t / nil
```

Description

Deletes a path probe from a window.

lt_probeSpec is the same type of probe specification as that used for adding a probe with the `geAdd*Probe` functions.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for a path probe to delete, or a string describing a path probe to delete.

Value Returned

`t`	A probe was found and deleted.
`nil`	A probe was not found and deleted.

Example

```
m = geDeletePathProbe( window(3) )
```

Prompts for a point and deletes the path probe under that point in `window(3)`.

© 2020 All Rights Reserved.

geDeleteProbe

```
geDeleteProbe(
     w_windowId
     o_userType
     [ g_crossProbeFlag ]
     )
     => t / nil
```

Description

Removes a probe from a set of probes in a window.

If *g_crossProbeFlag* is not `nil`, all the cross probes associated with the probe are deleted.

Arguments

w_windowId	Database ID of the window from which to remove the probe.
o_userType	Handle to the probe object.
g_crossProbeFlag	Indicates if the crossProbes associated with the probe should also be deleted. Valid Values: `t`, `nil` Default Value: `nil`

Value Returned

`t`	The probe is removed.
`nil`	The probe is not removed.

Example

```
geDeleteProbe( getCurrentWindow( ) probe )
```

The probe is deleted from the current window.

© 2020 All Rights Reserved.

geDeleteTermProbe

```
geDeleteTermProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     )
     => t / nil
```

Description

Deletes a terminal probe from a window.

lt_probeSpec is the same type of probe specification as that used for adding a probe with the `geAdd*Probe` functions.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for a terminal probe to delete, or a string describing a terminal probe to delete.

Value Returned

`t`	A probe was found and deleted.
`nil`	A probe was not found and deleted.

Example

```
geDeleteTermProbe( window(3) )
```

Prompts for a point and deletes the term probe under that point in `window(3)`.

© 2020 All Rights Reserved.

geEnterDeleteTermProbe

```
geDeleteTermProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     )
     => t / nil
```

Description

Deletes a terminal probe from a window.

lt_probeSpec is the same type of probe specification as that used for adding a probe with the `geAdd*Probe` functions.

If *g_repeatCmd* is `true`, then delete all terminals of the current selection set and highlight with a different color. If no selection set exists, then enable multiple post-selection to delete probe on terminals. If however, *g_repeatCmd* is `nil` then delete a probe on a terminal.

Arguments

w_windowId	Database ID of the window to probe. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for a terminal probe to delete, or a string describing a terminal probe to delete.
g_repeatCmd	SKILL expression specifying the probe selection mode (single select versus multiple post-select.)

Value Returned

`t`	A probe was found and deleted.
`nil`	A probe was not found and deleted.

Example

```
geEnterDeleteTermProbe( hiGetCurrentWindow() nil t )
```


© 2020 All Rights Reserved.

geExplainProbe

```
geExplainProbe(
     [ w_windowId ]
     [ lt_probeSpec ]
     )
     => t / nil
```

Description

Explains the probes in a window.

A view file window with information on each of the probes described by *lt_probeSpec* is displayed for you to view. *lt_probeSpec* is the same kind of specification used for the `geAdd*Probe` functions. If a point is specified, all probes within the selection aperture of the specified point are explained.

Note: The information for the file shown is put into a temporary file in `/tmp`.

Arguments

w_windowId	Database ID of the window containing the probes you want explained. The current window is used if *w_windowId* is not specified.
lt_probeSpec	Location in user units to look at for probes to explain, or a string describing a probe to explain.

Value Returned

`t`	A probe was found and explained.
`nil`	A probe was not found and explained.

Example

```
geExplainProbe( window(3) 10:10 )
```

Explains any probes under the point 10:10 in `window(3)`.

© 2020 All Rights Reserved.

geGetAllProbe

```
geGetAllProbe(
     [ w_windowId ]
     )
     => l_probes / nil
```

Description

Gets a list of all the probes in a window.

Arguments

w_windowId	Database ID of the window containing the probes.

Value Returned

l_probes	A list of all the probes in *w_windowId*.
nil	There are no probes in the window.

Example

```
probe_list = geGetAllProbe( GetCurrentWindow( ) )
```


© 2020 All Rights Reserved.

geSetMaxCreatedProbe

```
geSetMaxCreatedProbe(
     x_maxCreateProbe
     )
     => t / nil
```

Description

Sets the maximum number of created probes.

Arguments

x_maxCreateProbe	Maximum number of probes.

Value Returned

t	The maximum limit of probes has been set.
nil	There maximum number of probes could not be set

Example

```
geSetMaxCreatedProbe(1000)
```


© 2020 All Rights Reserved.

geGetCrossProbeValue

```
geGetCrossProbeValue(
     t_viewTypeName
     )
     => g_value / nil
```

Description

Queries the view type to see if it is set.

If this function is called before the cross probe form is created, a list of defaults are accumulated. During the creation of the cross probe form, the default list is checked to see if the value on the form defaults to true.

Arguments

t_viewTypeName	Name of a view type to cross probe. The cross probe form is initialized with this value if the form is not created yet, or the form has a value corresponding to this value set on the existing cross probe form. Valid Values: `all`, `none`

Value Returned

g_value	The cross probe value.
`nil`	The cross probe value is not set.

Example

```
geGetCrossProbeValue( "graphic" )
```

Returns `t`.

© 2020 All Rights Reserved.

geGetProbeNetStopLevel

```
geGetProbeNetStopLevel(
     )
     => x_netDisplayStopLevel
```

Description

Gets the display stop level of net probes.

Arguments

None

Value Returned

x_netDisplayStopLevel

Display stop level of net probes.

Example

```
myValue = geGetProbeNetStopLevel( )
```

Returns `20`.

© 2020 All Rights Reserved.

geGetProbeUserLPP

```
geGetProbeUserLPP(
     )
     => l_lpp / nil
```

Description

Gets the layer-purpose pair to be used by the `geAdd*Probe` functions if no layer-purpose pair is passed into these functions.

Arguments

None

Value Returned

l_lpp	The layer-purpose pair to be used for probing operations.
`nil`	No layer-purpose pair exists.

Example

```
geGetProbeUserLPP( )
```

Returns `(“marker“ “error“)`

© 2020 All Rights Reserved.

geAddNetProbeFromTo

```
geAddNetProbeFromTo(
     [ w_windowId ]
     [ l_lpp ]
     )
     => t / nil
```

Description

Adds a probe between two points where there is a connection between two different nets, such as nets separated by feedthrough or nets on different hierarchy but physically connected by pins.

The figure below depicts the impact on inverser defined as feedthrough to traverse them during probing.

After the first click:

After the second click on a net that has been probed by the first click:

The resulting probe is between the two selected nets.

© 2020 All Rights Reserved.

Arguments

w_windowId	Database ID of the window in which to add the probe.
l_lpp	Layer-purpose pair on which to display the probe.

Value Returned

`t`	The probe was created.
`nil`	The probe was not created.

Example

```
geAddNetProbeFromTo()
```

When you select a net on the window, if the point is correctly associated to a net, the net and a probe showing the net are printed on the CIW.

When you select a second point on the same net to limit the probe between the two points, if the net is a part of the path, the probe will be created and printed on the CIW.

© 2020 All Rights Reserved.

geSetLimitedProbe

```
geSetLimitedProbe(
     o_probeObject
     t_dir
     )
     => t / nil
```

Description

Sets the specified probe as a directed probe.

Arguments

o_probeObject	Database ID of the probe.
t_dir	An integer representing the direction. Valid Values: `0-inputOutput; 1-input; 2-output`

Value Returned

`t`	The probe is set as a directed probe.
`nil`	The probe is not set as a directed probe.

Example

The following example shows how to probe the first net within the current window, edited cellview and define this probe as a limited probe.

```
myProbe = nil
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    when(net = car(cellView~>nets)
     myProbe = geMakeProbeWithColor(?window window ?name "myProbeName" ?object net
                                    ?color list( "y0" "drawing") ?probeType "net")
      when(myProbe
    geSetLimitedProbe(myProbe t)
      )
    )
  )
)
```


© 2020 All Rights Reserved.

geSetProbeDirection

```
geSetProbeDirection(
     o_probeObject
     t_dir
     )
     => t / nil
```

Description

Sets the specified probe as a directed probe.

Arguments

o_probeObject	Database ID of the probe.
t_dir	An integer representing the direction. Valid Values: `0-inputOutput; 1-input; 2-output`

Value Returned

`t`	The probe is set as a directed probe.
`nil`	The probe is not set as a directed probe.

Example

The following example depicts how to probe the first net within the current window, edited cellview and define this probe as a FadeIn probe. This probes all nets driven by this net.

```
myProbe = nil
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    when(net = car(cellView~>nets)
     myProbe = geMakeProbeWithColor(?window window ?name "myProbeName" ?object net
                                    ?color list( "y0" "drawing") ?probeType "net")
      when(myProbe
    geSetProbeDirection(myProbe 1)
      )
    )
  )
)
```


© 2020 All Rights Reserved.

geClearProbeNetFilter

```
geClearProbeNetFilter (
     d_cellview
     )
     => t / nil
```

Description

If the specified cellview is editable, clears the probe net filtering settings currently saved for the cellview.

Arguments

d_cellview	Database ID of the cellview for which the probe net filter settings will be cleared.

Value Returned

t	The probe net name filter settings were cleared for the specified cellview ID.
nil	The command failed.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetProbeNetFilter(cellView "Hide" list("VDD*" "VSS*"))
geSaveProbeNetFilter(cellView)
geGetProbeNetFilter(cellview t)
geClearProbeNetFilter(cellview)
geGetProbeNetFilter(cellview t)
```

Clears the probe net filter previously saved in the current cellview.

© 2020 All Rights Reserved.

geGetProbeNetFilter

```
geGetProbeNetFilter(
     d_cellview
     [ g_fromDesign ]
     )
     => l_filterSettings / nil
```

Description

Returns the net filtering settings of the Highlight Options form currently in use for the given cellview.

Arguments

d_cellview	Database ID of the cellview.
g_fromDesign	When `t`, it returns the net filtering settings for probe that are currently saved in the design.

Value Returned

l_filterSettings	The net filtering settings of the Highlight Options form.
`nil`	The net name filtering settings were not found.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
netNameFilters = geGetProbeNetFilter(cellView)
netNameFilters
(nil mode "Hide" netNames ("VDD" "VSS"))
netNameFilters->mode
"Hide"
netNameFilters->netNames
("net1")
```

In the above example, only `net1` is specified to be filtered in the Highlight Options form.

© 2020 All Rights Reserved.

geSaveProbeNetFilter

```
geSaveProbeNetFilter (
     d_cellview
     )
     => t / nil
```

Description

Saves the probe net filtering settings that are currently applied on the given cellview after the `geSetProbeNetFilter` function is called. After you call the `geSaveProbeNetFilter`, the settings are reloaded automatically when you open the cellview in a new Virtuoso session.

Note: This function can be used only if the cellview is editable.

Arguments

d_cellview	Database ID of the cellview.

Value Returned

`t`	The net filtering is saved.
`nil`	The net filtering is not saved.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetProbeNetFilter(cellView "Hide" list("VDD" "VSS"))
geSaveProbeNetFilter(cellView)
```

In the above example, let us assume you do not want to see probe highlights for `VDD` and `VSS` nets.

© 2020 All Rights Reserved.

geSetProbeNetFilter

```
geSetProbeNetFilter(
     d_cellview
     t_mode
     l_netNames
     )
     => t / nil
```

Description

Sets a net filtering for probe on the given cellview.

Arguments

d_cellview	Database ID of the cellview on which to set the filter settings.
t_mode	The mode, `Hide` or `Show`.
l_netNames	The list of net names handled by Highlight Options filtering. If the *t_mode* is set to `Show`, during the next redraw only net probes that have nets in the provided list of net names are displayed. If *t_mode* is set to Hide, on the next redraw corresponding net probes will not be displayed.

Value Returned

`t`	The probe net filtering is set.
`nil`	The probe net filtering is not set.

Example

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetProbeNetFilter(cellView "Show" list("VDD" "VSS"))
```

In the above example, assume the you want to to display probes for only `VDD` and `VSS` nets.

© 2020 All Rights Reserved.

Probe File Format

This section describes the probe file format written and read by the `geSaveProbe` and `geLoadProbe` functions, respectively. In addition, an application you integrate into the Cadence framework can use this format to graphically annotate the design with information about certain aspects. For example, a simulator could output error messages using this file format, and you can later request a display of the errors found. This simplifies the process of locating and correcting errors found by analysis tools.

Each line in the file specifies the property list of one probe. This list must contain property name and property value pairs:

```
(propName propValue propName propValue...)
```

The following is a list of the possible property names you can store in the probe file:

```
explain
probeName
pathList
pathObjectName
probeType
probeArgType
probeColor
probeLabel
probeLabelColor
probeLabelFontSize
probeOnColorWhenDisplay
```

The property name value pairs can be in any order, except for the *explain* property, which must come first if specified. In addition to these predefined property names, user-specified properties can also be stored in the probe list. User-defined properties must appear before any of the predefined property names. The minimal probe list must include the type of the object to be probed in the *probeType* property and the name of the object to probe in the *pathObjectName* property. Following is an example of the minimum information required in each line of the probe file:

```
(probeType "instance" pathObjectName "Inst3")
```

The following is an example of a complete probe file.

```
(probeType "instance" pathObjectName "I3")

(explain "Instance Inst3 input A is not driven"
 probeType "instance" pathObjectName "Inst9")

(explain "Line 1 message\nLine 2 message"
 probeType "instance" pathObjectName "Inst4")

(probeType "instance" pathList "/I0" pathObjectName "I2"
 probeArgType "probeColor")

(probeType "net" pathObjectName "I0.Y")

(probeType "net" pathObjectName "I1.Y" probeArgType
 "probeColor" probeColor ("y4" "drawing"))
```


© 2020 All Rights Reserved.

```
(probeType "net" pathList "/I3" pathObjectName "Y"
 probeArgType "probeColor" probeColor ("y4" "drawing"))

(probeType "terminal" pathObjectName "select")

(probeType "terminal" pathObjectName "/I3/D" probeArgType
 "probeColor")

(probeType "path" pathList "/I1" pathObjectName "A"
 otherPathList "/I3" otherObjectName "Y")

(probeType "path" pathObjectName "/I3/B"
 otherObjectName "/I1/Y")

(probeType "instance" pathObjectName "I2" probeArgType
 "probeLabel" probeLabel "myName")

(probeType "instance" pathObjectName "I1" probeArgType
 "probeLabel" probeLabel "newName"
  probeLabelFontSize 0.1)
```


© 2020 All Rights Reserved.

geLoadNextProbe

```
geLoadNextProbe(
     [ w_window ]
     [ p_port ]
     [ s_prop ]
     [ s_cond ]
     [ g_value ]
     [ s_loadCount ]
     )
     => t / nil
```

Description

Loads a probe into a window. Identical to `geLoadProbe` except that *`s_loadCount`* is a symbol containing the index of the probe to be loaded.

`s_loadCount` is incremental if successful.

Arguments

`w_windowId`	Database ID of the window containing the probes you want loaded.
`p_port`	Port to load the probe information through.
`s_prop`	Property to be examined to decide whether to load probe.
`s_cond`	Condition that must be satisfied if the probe is to be loaded.
`g_value`	Value of the property tested against the condition to decide whether to load the probe.
`s_loadCount`	Variable containing the index of the probe to load.

Value Returned

`t`	A probe was found and loaded.
`nil`	A probe was not found and loaded.

Example

```
m = infile( "/tmp/myProbes.out" )
ct = 0
```


© 2020 All Rights Reserved.

```
geLoadNextProbe( window(3) m 'myProp 'equal "critical" 'ct)
```

Loads the first probe in file `/tmp/myProbes.out` that has the user-defined property `myProp` equal to the string `critical`, then increments `ct` to `1`.

© 2020 All Rights Reserved.

geLoadProbe

```
geLoadProbe(
     [ w_windowId ]
     [ p_port ]
     [ s_prop ]
     [ s_cond ]
     [ g_value ]
     )
     => t / nil
```

Description

Loads probes into a window.

The probes are recreated by reading the necessary information in the property list saved from a `geSaveProbe` call. This property list also allows the restoring of any user-defined or application-specific properties that might have been put on the probes.

The optional parameters *s_prop*, *s_cond*, and *g_value* conditionally load probes. *s_prop* is a symbol representing a property name. *s_cond* is a SKILL symbol representing the condition. *g_value* is the reference value. The probe is loaded only if *s_prop* is related to *g_value* by *s_cond*.

Arguments

w_windowId	Database ID of the window containing the probes you want loaded. If not specified, the current window is used.
p_port	Port to load the probe information through. If *p_port* is not specified, the probes are read from the file `probe.out`.
s_prop	Property to be examined in deciding whether to load probe.
s_cond	Condition that must be satisfied if the probe is to be loaded.
g_value	Value of the property tested against the condition to decide whether to load the probe.

Value Returned

`t`	A probe was found and loaded.
`nil`	A probe was not found and loaded.

© 2020 All Rights Reserved.

Example

```
m = infile( "/tmp/myProbes.out" )
geLoadProbe( window(3) m 'myProperty 'equal "critical")
```

Loads all probes in file `/tmp/myProbes.out` that have the user-defined property `myProperty` equal to the string `critical`.

© 2020 All Rights Reserved.

geMakeProbeWithColor

```
geMakeProbeWithColor(
     [ ?window w_windowId ]
     [ ?name t_name ]
     [ ?object g_object ]
     [ ?otherObject g_otherObject ]
     [ ?color g_color ]
     [ ?probeType t_probeType ]
     [ ?displayStyle t_displayStyle ]
     [ ?viewTypeList l_viewTypesToCrossProbe ]
     [ ?message t_messageString ]
     )
     => o_probeObject
```

Description

Creates a probe of the specified type and gives it the specified color.

The object is highlighted when found using the display attributes of layer *g_color*.

The probe type can be inferred from *g_object* if it is an object Id and is not a terminal. A terminal can be part of either a terminal probe or a net probe, so the probe type cannot be derived in that case. If *g_object* is a path list, *t_probeType* must be specified since the type cannot be derived from the name.

A single probe can have only one of the three display arguments (`color`, `label`, or `onColor`), and this function sets the `color` attribute. A multiple probe (same object and different display argument) can specify multiple display arguments. If *g_object* has already been probed, the probe becomes a multiple probe.

Arguments

?window *w_windowId*	Database ID of the window to contain the probe.
?name *t_name*	Name of the probe.

© 2020 All Rights Reserved.

`?object` *`g_object`*

Object to be probed. It can be an instance, a net, or a terminal, and is specified either by the database ID of the object or by a list of strings describing a path to the object. The strings are in *`memInst`* format, which is needed for iterated instances and mosaics.

If the database ID of the object is used, the path is derived from the hierarchical path from the top level of *`w_windowId`* to the object.

`?otherObject` *`g_otherObject`*

Second object specified only for path probes, which require two terminal objects.

`?color` *`g_color`*

Layer on which to display the probe.

`?probeType` *`t_probeType`*

Type of the probe.

Valid Values: `net`, `path`, `terminal`, `instance`.

`?displayStyle` *`t_displayStyle`*

Used to determine whether to display the probe using flight lines or objects. Display style `object` highlights the physical objects associated with the probe. Display style `flight` highlights the terminals and draws straight flight lines between the terminals of the net. If *`t_probe_type`* = *`net`*, this is the way the net is displayed.

Valid Values: `object`, `flight`

Default: `object`

`?viewTypeList` *`l_viewTypesToCrossProbe`*

A list of strings that indicates the view types that should be crossProbed. If `nil`, no cross-probes. If a list of strings, the list of view types to cross-probe. If `all`, cross-probe all possible view types.

`?message` *`t_messageString`*

String passed to the C-level cross probe functions.

© 2020 All Rights Reserved.

Value Returned

o_probeObject	Database ID of the new probe object.

Examples

```
probe = geMakeProbeWithColor(?name "myProbe"
?color list( "y0" "drawing")
?object list( "I1" "I2")
?probeType "instance")
```

Probes instance I2 contained within instance I1 within the top level of the current window. The probe is drawn on layer `(y0 drawing)`.

```
probe = geMakeProbeWithColor(?name "myProbe"
    ?color list( "marker" "error")
    ?object list( "/I1/I5/NET10")
    ?probeType "net")
```

Probes net NET10 contained within instance I5 within instance I1 within the top level of the current window. The probe is drawn on layer `(marker error)`.

© 2020 All Rights Reserved.

geMakeProbeWithoutColor

```
geMakeProbeWithoutColor(
     [ ?window w_windowId ]
     [ ?name t_name ]
     [ ?object g_object ]
     [ ?otherObject g_otherObject ]
     [ ?probeType t_probeType ]
     [ ?viewTypeList l_viewTypesToCrossProbe ]
     [ ?message t_messageString ]
     )
     => o_probeObject
```

Description

Creates a probe of the specified type without specifying any colors. The object is highlighted when it is found by specifying the color defined in the `geProbeOptionForm`. This form can accessed from *Option - Probe*.

The type of probe — an object ID or a terminal — can be inferred from *`g_object`*. A terminal can be part of either a terminal probe or a net probe, so the probe type cannot be derived in that case. If *`g_object`* is a path list, *`t_probeType`* must be specified since the type cannot be derived from the name.

Arguments

`?window` *`w_windowId`*	Database ID of the window to contain the probe.
`?name` *`t_name`*	Name of the probe.
`?object` *`g_object`*	Object to be probed. It can be an instance, a net, or a terminal, and is specified either by the database ID of the object or by a list of strings describing a path to the object. The strings are in *`memInst`* format, which is needed for iterated instances and mosaics. If the database ID of the object is used, the path is derived from the hierarchical path from the top level of *`w_windowId`* to the object.
`?otherObject` *`g_otherObject`*	Second object specified only for path probes, which require two terminal objects.
`?probeType` *`t_probeType`*	Type of the probe. Valid Values: `net`, `path`, `terminal`, `instance`.

© 2020 All Rights Reserved.

`?viewTypeList` *`l_viewTypesToCrossProbe`*

A list of strings that indicates the view types that should be crossProbed.
If it is `nil`, no cross probing is done.
If a list of strings is specified, it represents the list of view types to cross-probe.
If it is `all`, all possible view types are cross probed.

`?message` *`t_messageString`*

String passed to the C-level cross probe functions.

Value Returned

`o_probeObject` Database ID of the new probe object.

Examples

```
probe = geMakeProbeWithoutColor(
?name "myProbe"
?object list( "I1" "I2")
?probeType "instance")
```

Probes instance I2 contained within instance I1 within the top-level of the current window. The probe is drawn on layer `(y0 drawing)`.

© 2020 All Rights Reserved.

geMakeProbeWithLabel

```
geMakeProbeWithLabel(
     [ ?window w_windowId ]
     [ ?name t_name ]
     [ ?object g_object ]
     [ ?otherObject g_otherObject ]
     [ ?label g_label ]
     [ ?probeType t_probeType ]
     [ ?displayStyle t_displayStyle ]
     [ ?viewTypeList l_viewTypesToCrossProbe ]
     [ ?message t_messageString ]
     [ ?labelName t_labelName ]
     [ ?labelColor l_labelColor ]
     [ ?labelOffset l_labelOffset ]
     [ ?labelFont t_labelFont ]
     [ ?labelFontSize f_labelFontSize ]
     [ ?labelJust t_labelJust ]
     [ ?labelOrient t_labelOrient ]
     )
     => o_probeObjectId
```

Description

Creates a probe and gives it the specified label.

The object is highlighted when found using the display attributes of the layer specified in *g_label* or *l_labelColor*. *t_probeType* can be `net`, `path`, `terminal`, or `instance`. The probe type can be inferred from *g_object* if it is an object ID and is not a terminal.

A terminal can be part of either a terminal probe or a net probe, so the probe type can't be derived in that case. If *g_object* is a path list, *t_probeType* must be specified since the type cannot be derived from the name. For net probes, *t_displayStyle* is used to determine whether to display the probe using flight lines or objects. Display style `object` highlights the physical objects associated with the probe. Display style `flight` highlights the terminals and draws straight flight lines between the terminals of the net. The default display style is `object`.

The probe label may be specified either as *g_label*, which is a list of label attributes, or by specifying specific label attributes as arguments to `geMakeProbeWithLabel`. The latter is usually more convenient if the predefined defaults are acceptable.

A single probe can have only one of the three display arguments (`color`, `label`, or `onColor`), and this function sets the `label` attribute. A multiple probe (same object and

© 2020 All Rights Reserved.

different display argument) can specify multiple display arguments. If *g_object* has already been probed, the probe becomes a multiple probe.

Arguments

?window *w_windowId*

Database ID of the window that is to contain the probe.

?name *t_name*

Name of the probe.

?object *g_object*

Object to be probed. If a list of strings is used to specify an instance terminal name, the list must have two elements for each terminal name. The first element is a string of the instance name. The second element is a string of the terminal name.

?otherObject *g_otherObject*

Second object specified only for path probes, which require two terminal objects.

?label *g_label*

Label of the probe.Valid Values: *t_labelName*, *l_labelColor*, *l_labelOffset*, *t_labelFont*, *f_labelFontSize*, *t_labelJust*, *t_labelOrient*. These attributes are the same as those that can be specified directly, and are described below. The default location of the label is calculated based on the type of probe.

?probeType *t_probeType*

Type of the probe.Valid Values: `instance`, `net`, `path`, `terminal`

?displayStyle *t_displayStyle*

If *t_probe_type* = `net`, then this is the way the net is displayed. Valid Values: `object` and `flight`.

?viewTypeList *l_viewTypesToCrossProbe*

A list of strings that indicates the view types that should be cross probed.If `nil`, no cross-probes. If list of strings, the list of view types to cross-probe. If `all`, then cross-probe all possible view types.

?message *t_messageString*

String passed to the C-level *crossProbe* functions.

?labelName *t_labelName*

© 2020 All Rights Reserved.

Text of probe label.

`?labelColor` *`l_labelColor`*

Layer-purpose pair to be used for displaying probe label.

`?labelOffset` *`l_labelOffset`*

Offset in user units from the default location to place the label.

`?labelFont t_labelFont`

Name of font to use for label.

`?labelFontSize` *`f_labelFontSize`*

Height in user units for the probe label.

`?labelJust` *`t_labelJust`*

Justification of probe label.

`?labelOrient` *`t_labelOrient`*

Orientation of the probe label.

Value Returned

`o_probeObjectId` Database ID of the new probe object.

Example

```
probe = geMakeProbeWithLabel(
?window getCurrentWindow( )
?name "probe2"
?object list("I2" "A1")
?labelName "labelName"
?labelColor list("y0" "drawing")
?probeType "terminal"
?displayStyle "object"
)
```

Probes terminal `A1` on instance `I2` of the current window.

```
probe = geMakeProbeWithLabel(?name "myProbe"
    ?labelName "myProbeLabel"
    ?labelColor list("y0" "drawing")
    ?labelOffset 10:10
    ?labelFont "gothic"
    ?labelFontSize 100.
    ?labelJust "centerCenter"
    ?labelOrient "R90"
    ?object list( "I1" "I2")
    ?probeType "instance")
```


© 2020 All Rights Reserved.

Has the same effect as the previous example.

© 2020 All Rights Reserved.

geMakeProbeWithOnColor

```
geMakeProbeWithOnColor(
     [ ?window w_windowId ]
     [ ?name t_name ]
     [ ?object g_object ]
     [ ?otherObject g_otherObject ]
     [ ?onColor g_onColor ]
     [ ?probeType t_probeType ]
     [ ?displayStyle t_displayStyle ]
     [ ?viewTypeList g_viewTypesToCrossProbe ]
     [ ?message t_messageString ]
     )
     => o_probeObjectId
```

Description

Creates an instance probe and gives it the specified color.

The object is highlighted when found using the display attributes of layer *g_onColor*. This function only works with instance probes. You can specify other types of probes, but they will not display. *t_probeType* can be net, path, terminal, or instance. The probe type can be inferred from *g_object* if it is an object ID and is not a terminal.

A terminal can be part of either a terminal probe or a net probe, so the probe type can't be derived in that case. If *g_object* is a path list, *t_probeType* must be specified because the type cannot be derived from the name. For net probes, *t_displayStyle* is used to determine whether to display the probe using flight lines or objects. Display style object highlights the physical objects associated with the probe. Display style flight highlights the terminals and draws straight flight lines between the terminals of the net. The default display style is object.

g_viewTypesToCrossProbe is a list of strings that indicates the view types that should be cross probed. *t_messageString* is an optional string to be passed to C-level probe functions.

A single probe can have only one of the three display arguments (color, label, or onColor), and this function sets the onColor attribute. A multiple probe (same object and different display argument) can specify multiple display arguments. If *g_object* has already been probed, the probe becomes a multiple probe.

© 2020 All Rights Reserved.

Arguments

?window w_windowId

Database ID of the window that is to contain the probe.

?name *t_name*

Name of the probe.

?object *g_object*

Object to be probed. It can be an instance, a net, or a terminal, and is specified either by the database ID of the object or by a list of strings describing a path to the object. The strings are in *memInst* format, which is needed for iterated instances and mosaics. If the database ID of the object is used, the path is derived from the hierarchical path from the top level of *w_windowId* to the object.

?otherObject *g_otherObject*

Second object specified only for path probes, which require two terminal objects.

?onColor *g_onColor*

Layer on which to display the probe.

?probeType *t_probeType*

Type of the probe.

Valid Values: instance, net, path, terminal

?displayStyle *t_displayStyle*

If *t_probe_type* = net, then this is the way the net is displayed.

Valid Values: object or flight.

viewTypeList *g_viewTypesToCrossProbe*

If nil, no cross-probes.

If list of strings, the list of view types to cross-probe.

If all, then cross-probe all possible view types.

?message *t_messageString*

String passed to the C-level crossProbe functions.

© 2020 All Rights Reserved.

Value Returned

o_probe_objectId Database ID of the new probe object.

Example

```
probe = geMakeProbeWithOnColor(?name "myProbe"
?onColor list( "y0" "drawing")
?object list( "I1" "I2")
?probeType "instance")
```

Probes instance I2 contained within instance I1 within the top level of the current window. The probe is drawn on layer `(y0 drawing)`.

© 2020 All Rights Reserved.

gePrintAllProbe

```
gePrintAllProbe(
     [ w_windowId ]
     [ p_port ]
     )
     => t / nil
```

Description

Prints a list of the probes in a window.

All the attributes of the probe are printed.

Arguments

w_windowId	Database ID of the window from which you want a list of probes. The default is the current window.
p_port	SKILL port printed out to. Default: *p_port*

Value Returned

`t`	The list of probes is created,.
`nil`	There are no probes in the window.

Example

```
port = outfile( "myFile")
gePrintAllProbe( hiGetCurrentWindow( ) port)
```

The attributes of all probes in the current window are printed in a text file called `myFile` in the current directory.

© 2020 All Rights Reserved.

geProbeP

```
geProbeP(
     o_probe
     )
     => t / nil
```

Description

Checks if the value passed in is a probe.

Arguments

o_probe	Handle to a probe object.

Value Returned

`t`	The object passed in is a probe.
`nil`	The object passed is not a probe.

Example

```
if( geProbeP( myObject) then
warn( "Object %L is not a probe\n" myObject ) )
```


© 2020 All Rights Reserved.

geRegCrossProbeTrigger

```
geRegCrossProbeTrigger(
     g_ilFunc
     x_priority
     t_name
     )
     => t / nil
```

Description

Registers a trigger function for cross probing.

Arguments

g_ilFunc	The symbol of the client's cross probe trigger function.
x_priority	An integer representing the trigger priority.
t_name	A string used for identification only.

Value Returned

`t`	Registers a trigger function for cross probing.
`nil`	Does not register a trigger function.

Example:

```
geRegCrossProbeTrigger('myXProbeTrigger 1 "DEMO")
```

Registers the trigger function `myXProbeTrigger` as priority 1 for cross probing.

© 2020 All Rights Reserved.

geSaveProbe

```
geSaveProbe(
     [ w_windowId ]
     [ p_port ]
     )
     => t / nil
```

Description

Saves probes to a port.

The probes are saved in the file by dumping out the entire property list associated with the probe. This enables the probe to be restored because the necessary information is in the property list. This also allows the restoring of any user defined or application-specific properties that might have been put on the probes.

Arguments

w_windowId	Database ID of the window containing the probes you want saved. If not specified, the current window is used.
p_port	Port to save the probe information through. If *p_port* is not specified, the probes are stored into the file `probe.out`.

Value Returned

`t`	A probe was found and saved.
`nil`	A probe was not found and saved.

Example

```
m = outfile( "/tmp/myProbes.out" )
geSaveProbe( window(3) m )
```

Writes out all probes in `window(3)` to the file `/tmp/myProbes.out`.

© 2020 All Rights Reserved.

geSetCrossProbeValue

```
geSetCrossProbeValue(
     t_viewTypeName
     [ g_value ]
     )
     => t / nil
```

Description

Sets the view types to cross-probe.

The crossProbe form is created only on the first *addProbe* menu command. If this function is called before the crossProbe form is created, then a list of defaults is accumulated. During the creation of the crossProbe form, the default list is checked to see if the value on the form defaults to true.

Arguments

t_viewTypeName	Name of a view type to cross-probe. The crossProbe form is initialized with this value if the form is not created yet or if the form has a value corresponding to this value set on the existing crossProbe form. This argument can be set to *all* or *none*.
g_value	Value assigned to crossProbe.

Value Returned

`t`	Returns `t` by default.
`nil`	If *t_viewTypeName* is `all` or `none`, this value is ignored.

Example

```
geSetCrossProbeValue( "graphic" t )
```

Set to cross probe `graphic` view types by default.

© 2020 All Rights Reserved.

geSetProbeNetStopLevel

```
geSetProbeNetStopLevel(
     x_netDisplayStopLevel
     )
     => t / nil
```

Description

Sets the display stop level of net probes.

Since a net can traverse many levels of hierarchy, this function controls how many levels the net probe should traverse. This stop level is relative to the current level of the hierarchy. Thus, the default setting of 0 only shows the net probe in the current level. This stop level is a global variable, and thus affects all net probes in all windows.

Arguments

x_netDisplayStopLevel

Display stop level of net probes relative to the current level of hierarchy.

Value Returned

`t`	The display stop level is set.
`nil`	The display stop level is not set.

Example

```
geSetProbeNetStopLevel( 20 )
```

Causes all net probes to be displayed to the lowest level of hierarchy.

© 2020 All Rights Reserved.

geSetProbeUserLPP

```
geSetProbeUserLPP(
     l_lpp
     )
     => t / nil
```

Description

Sets the layer-purpose pair to be used by the `geAdd*Probe` functions if no layer-purpose pair is passed in to these functions.

This function overrides the cycling of layers y0 through y9 with purpose drawing. This is a global variable.

Arguments

l_lpp	Layer-purpose pair to use for successive probing operations.

Value Returned

`t`	The layer was set.
`nil`	The layer was not set.

Example

```
geSetProbeUserLPP( list( "marker" "error" ) )
```

Sets the default probe layer to be the `error` `marker` layer.

© 2020 All Rights Reserved.

geiLoadProbe

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

Description

Displays the *Load Probes* form.

The probe file name is specified in this form. All probes saved in this file are loaded in the window.

Arguments

w_windowId	Database ID of the window.

Value Returned

t	The *Load Probes* form is displayed.

© 2020 All Rights Reserved.

geiSaveProbe

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

Description

Displays the *Save Probes* form.

The probe file name is specified in this form. All probes in the window are saved in this file.

Arguments

w_windowId	Database ID of the window.

Value Returned

t	The *Save Probes* form is displayed.

© 2020 All Rights Reserved.

Icon Creation and Loading Functions

geCellViewToDlist

```
geCellViewToDlist(
     d_cellview
     )
     => l_dlist / nil
```

Description

Returns a display list of all the objects in a cellview.

The display list contains a display function for each object in *d_cellview* and in all instances contained within its hierarchy.

Arguments

d_cellview	Database ID of the cellview whose contents are to be represented as a list.

Value Returned

l_dlist	Returns a list representing the contents of *d_cellview.*
nil	Returns nil if the cellview contains no objects.

Example

```
geCellViewToDlist( window(2)->editCellView )
```

Returns a list made from the objects in the cellview being edited in window(2).

© 2020 All Rights Reserved.

geLoadIcon

```
geLoadIcon(
     [ t_iconName ]
     [ t_fileName ]
     [ x_width ]
     [ x_height ]
     [ x_bgColor ]
     )
     => t / nil
```

Description

Creates an icon from a SKILL file.

This function loads SKILL code in *t_fileName* and takes the resulting *Dlist*, which must have the name *t_iconName*, and creates an icon with dimensions *x_width* and *x_height*, and using pixel value *x_bgColor* for the background. If any of the arguments are defaulted, a form is displayed prompting you to enter those values. The background color of the icon depends on which color (if any) is mapped to pixel value *x_bgColor* in the current technology file. The newly created icon is assigned to the symbol *t_iconName*. This icon can be used in any user interface function that takes an icon as an argument.

Arguments

t_iconName	Name given to the icon.
t_fileName	File in which the icon data is to be found.
x_width	Pixel width of the new icon.
x_height	Pixel height of the new icon.
x_bgColor	Background pixel value of the new icon.

Value Returned

`t`	The icon was successfully loaded.
`nil`	The icon was not successfully loaded.

Example

```
geLoadIcon( "myIcon" "myIconFile" 40 40 2 )
```


© 2020 All Rights Reserved.

Loads icon data from the file `myIconFile`, creates an icon named `myIcon` which is 40 pixels square and has pixel value 2 as its background color.

© 2020 All Rights Reserved.

geSaveIcon

```
geSaveIcon(
     [ w_windowId ]
     [ t_iconName ]
     [ t_fileName ]
     )
     => t / nil
```

Description

Saves a cellview as a SKILL icon file.

This function creates a *Dlist* from `w_windowId->editCellView`, and stores the *Dlist* in `t_fileName`. The file contains SKILL code which recreates the *Dlist* when loaded. The SKILL code assigns the recreated *Dlist* to a symbol with `t_iconName` as its print name. If `w_windowId` is defaulted, the current window is used. If `t_iconName` or `t_fileName` is defaulted, a form is displayed allowing you to enter these values.

Arguments

`w_windowId`	Database ID of the window containing the cellview to be saved as an icon file.
`t_iconName`	Name given to the icon.
`t_fileName`	File in which to save the icon data.

Value Returned

`t`	The icon was successfully written.
`nil`	The icon was not successfully written.

Example

```
geSaveIcon( window(2) "myIcon" "myIconFile" )
```

Writes icon data derived from the edit cellview in `window(2)` to the file `myIconFile` and names the icon `myIcon`.

© 2020 All Rights Reserved.

geUnRegCrossProbeTrigger

```
geUnRegCrossProbeTrigger(
     s_ilFunc
     )
     => t / nil
```

Description

Removes the cross probe trigger, if one is registered.

Arguments

s_ilFunc	The symbol of the client's cross probe trigger function.

Value Returned

t	The cross probe trigger is removed.
nil	The cross probe trigger is not removed.

© 2020 All Rights Reserved.

Utility Functions

geAskOverwriteMode

```
geAskOverwriteMode(
     d_cellviewId
     )
     => t / nil
```

Description

Prompts the user to determine whether or not to overwrite the already open cellview.

Arguments

d_cellviewId	Database ID of the cellview.

Value Returned

`t`	The cellview is overwritten.
`nil`	The cellview is not overwritten.

Example

```
geAskOverwriteMode( cell3 )
```

Asks the user whether or not to overwrite the cellview cell3 when the cellview is already open.

© 2020 All Rights Reserved.

geClosePolygon

```
geClosePolygon(
     d_cellview
     l_points
     t_snapMode
     )
     => l_pointsPolygon / nil
```

Description

Returns a list of points that describe a closed polygon if a polygon could be made from the specified points. Otherwise, returns `nil`.

Arguments

d_cellview	Database ID of the cellview to contain the polygon.
l_points	List of points used to create the polygon.
t_snapMode	Specifies how the cursor snaps when a polygon is created. For example, some possible snap modes are “orthogonal”, “diagonal”, or “L90YFirst”.

Value Returned

l_pointsPolygon	Returns a list of points that describe a resulting closed polygon.
`nil`	Returns `nil` when a closed polygon could not be created.

Example

```
geClosePolygon(geGetEditCellView() list(0:5 1:6 1:6) "orthogonal")
=> ((1.0 6.0)
      (1.0 5.0)
      (0.0 5.0)
    )
```


© 2020 All Rights Reserved.

geCompressPointArray

```
geCompressPointArray(
     d_cellview
     l_points
     g_closedList
     )
     => l_pointList / nil
```

Description

Searches a point list in a cellview for collinear points and deletes the intermediate collinear points.

Arguments

d_cellview	Database ID of the cellview to search.
l_points	List of points to search.
g_closedList	Either `t` or `nil` to specify whether the list is a closed list or not.

Value Returned

l_pointList	A list of coordinates representing the compressed point array.
`nil`	The intermediate collinear points are not deleted.

Example

```
geCompressPointArray( geGetEditCellView( ) list(0:0 10:0 20:0 20:20 0:20 0:0) t )
=> ((0.0 0.0)(20.0 0.0)(20.0 20.0)(0.0 20.0))
```

Deletes the vertex at the point 10:0 from the closed list in the cellview returned by `geGetEditCellView`.

© 2020 All Rights Reserved.

geCoordp

```
geCoordp(
     coord
     )
     => t / nil
```

Description

Determines whether the data supplied by *coord* is a point coordinate.

Arguments

coord	Data to check. *coord* can be of any type.

Value Returned

`t`	The data is the coordinates of a point.
`nil`	The data is not the coordinates of the point.

Examples

```
geCoordp(10:10)
```

Returns `t`.

```
geCoordp("myCell")
```

Returns `nil`.

© 2020 All Rights Reserved.

geCreatePacket

```
geCreatePacket(
     w_window
     t_name
     t_stippleName
     t_lineStyleName
     t_fillColorName
     t_outlineColorName
     [ d_lineThickness ]
     [ g_reuse ]
     [ t_fillStyleName ]
     )
     => t / nil
```

Description

Creates or updates a DFII display packet. The `reuse` flag indicates whether to use an existing display packet with the given name or to update it with given parameters.

© 2020 All Rights Reserved.

Arguments

w_window	Window used to get the display.
t_name	The name of the display packet to be created.
t_stippleName	The stipple name of the packet.
t_lineStyleName	The line style name of the packet.
t_fillColorName	The fill color name of the packet.
t_outlineColorName	The outline color name of the packet.
d_lineThickness	The line thickness of the packet.
g_reuse	When false, erases and re-creates an existing packet.
t_fillStyleName	The fill style name of the packet.

Value Returned

`t`	The display packet was created or updated.
`nil`	Otherwise.

Example

```
geCreatePacket(window(3) "pkt1" "solid" "solid" "green" "green" 4 nil)
```


© 2020 All Rights Reserved.

geGetInstTransform

```
geGetInstTransform(
     d_InstId
     )
     => list( list( xOffset yOffset ) rotation magnification )
```

or

```
geGetInstTransform(
     list( dbInstId dbInstId ...)
     )
     => list( list( xOffset yOffset ) rotation magnification )
```

Description

Evaluates a path to an instance returned by `dbGetTrueOverlaps` and concatenates the overall offset, transform, and magnification of the instance placed within the design hierarchy. The value returned by `geGetInstTransform` is a three member list containing the values of the three variables mentioned above. This transform list can be passed to the functions `dbCopyFig` or `dbMoveFig` as their third argument to copy or move the shape to the *`cellView`* passed to `dbGetTrueOverlaps` as its first argument.

Note: Cadence recommends you to use the `dbGetInstTransform` function instead.

Arguments

`d_InstId`	This can be a single *`instId`*, or a list of *`instId`*s in order from top to bottom which describes a hierarchical path to an instance at a lower level.

Value Returned

`xOffset yOffset`	The float values of the x and y offsets.
`rotation`	A string that is the value of the rotation.
`magnification`	The amount of magnification as a float.

Example

```
geGetInstTransform(
list(topInstId middleInstId bottomInstId))
=> ((-3.1 0.9) "R0" 1.0)
```


© 2020 All Rights Reserved.

geGetLockInfo

```
geGetLockInfo(
     t_libName
     t_cellName
     t_viewName
     )
     => t_userHostInfo / no data on lock available
```

Description

Returns the user and host from a `.cdslck` file.

Arguments

t_libName	String value, library name.
t_cellName	String value, cell name.
t_viewName	String value, view name.

Value Returned

t_userHostInfo	Returns the user and host information successful.
`no data on lock available`	No data is found.

Example

```
geGetLockInfo("myLib" "myCell" "myView")
"user : yourName\nhost : trnsoph6"
```


© 2020 All Rights Reserved.

geInstIdToFileSpec

```
geInstIdToFileSpec(
     d_instId
     t_mode
     )
     => l_fileSpec
```

Description

Creates a *fileSpec* (disassociated property list) from the data in an instance and the given mode.

© 2020 All Rights Reserved.

Arguments

d_instId	Database ID of the instance.
t_mode	Mode used in the *fileSpec*.Valid Values: `r` (read), `a` (append), `w` (overwrite).

Value Returned

l_fileSpec The filled-out *fileSpec*. *fileSpec* contains the following information:

```
list(nil
'libName            instId~>libName
'libId              instId~>master~>lib
'viewName           instId~>viewName
'viewType
          instId~>master~>cellViewType
'viewId             instId~>master~>view
'cellName           instId~>cellName

'configName         instId~>master~>lib~
>openedConfig~>name
'viewNameList       nil
'accessMode         mode
'instId             instId
'hierarchy          "")
)
```

Example

```
geInstIdToFileSpec( inst3 "a" )
```

Creates a *fileSpec* of the instance `inst3` with `append` mode.

© 2020 All Rights Reserved.

geIsPathReentrant

```
geIsPathReentrant(
     d_cellview
     l_points
     n_width
     [ t_style ]
     [ n_beginExt ]
     [ n_endExt ]
     )
     => t / nil
```

Description

Determines whether a path is self-intersecting.

In a self-intersecting path, one part of the path overlaps another part of the path. The system calculates the outline of the path using the other arguments.

Arguments

d_cellview	Database ID of the cellview containing the path to check.
l_points	List of points making up the path.
n_width	Width of the path.
t_style	String defining an optional path style. If no style is given, the default is *truncateExtend*. Valid Values: `truncateExtend`, `extendExtend`, `roundRound`, `varExtendExtend`
n_beginExt	Beginning path extension. Default: `0.0`
n_endExt	Ending path extension. Default: `0.0`

Value Returned

`t`	The path is self-intersecting.
`nil`	The path is not self-intersecting.

© 2020 All Rights Reserved.

Examples

```
geIsPathReentrant( geGetEditCellView( ) list( 0:0 10:10 0:10 10:0 ) 3.0
"truncateExtend" 1.0 0 )
```

Returns `t`.

```
geIsPathReentrant( geGetEditCellView( ) list( 0:0 10:10 0:10 3.0 )
```

Returns `nil`.

© 2020 All Rights Reserved.

geIsPolygonReentrant

```
geIsPolygonReentrant(
     d_cellview
     l_points
     )
     => t / nil
```

Description

Determines whether a polygon is self-intersecting.

In a self-intersecting polygon, one part of the polygon overlaps another part of the polygon. Self-touching is not considered self-intersecting.

Arguments

d_cellview	Database ID of the cellview containing the polygon to check.
l_points	List of polygon points.

Value Returned

`t`	The polygon is self-intersecting.
`nil`	The polygon is not self-intersecting.

Example

```
geIsPolygonReentrant( geGetEditCellView( ) list( 0:0 10:0 10:5 5:5 5:0 0:5 ) )
```

This polygon is self-touching, but not self-intersecting. Returns `nil`.

```
geIsPolygonReentrant( geGetEditCellView( ) list( 0:0 10:0
    10:5 5:5 5:-5 0:-5 ) )
```

This polygon is self-intersecting. Returns `t`.

© 2020 All Rights Reserved.

geOpen

```
geOpen(
     [ ?window w_windowId ]
     [ ?lib t_lib ]
     [ ?cell t_cell ]
     [ ?view t_view ]
     [ ?viewType t_viewType ]
     [ ?mode t_mode ]
     )
     => w_windowId / nil
```

Description

Opens a design in a new window.

The cell has the library *t_lib*, cell name *t_cell*, and view name *t_view*. Version *t_version* of the cell is opened in access mode *t_mode*.

Arguments

?window *w_windowId*	Database ID of the window in which to open the cell.
?lib *t_lib*	Name of the library containing the cell.
?cell *t_cell*	Name of the cell.
?view *t_view*	View of the cell. For example: `layout`, `symbol`, or `schematic`.
?viewType *t_viewType*	Type of data to open. For example: `maskLayout`, `schematic`.
?mode *t_mode*	Access mode of the cell. Valid Values: `r` (read), `a` (append)

Value Returned

w_windowId	Returns the ID of the window in which the design is created or is opened successfully. As a special case, `geOpen` returns `t` instead of the window ID. For example, when the window in which you want to open a design is not a DE window.
`nil`	Returns `nil` if a window could not be created or a design could not be opened.

© 2020 All Rights Reserved.

Example

```
geOpen( ?window hiGetCurrentWindow( ) ?lib "mylib"
?cell "mycell" ?view "layout" ?viewType "maskLayout" ?mode "a" )
```

=> `window:3`

Opens the `layout` view of `mycell` in `window:3` in edit mode.

© 2020 All Rights Reserved.

geQuerySelSet

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

Description

Displays the properties of each object in the selected set.

Arguments

w_windowId	Database ID of the window containing the selected set. If no *w_windowId* is specified, the current window is used.

Value Returned

t	Selected objects are found.
nil	Selected objects are not found.

Example

```
geQuerySelSet( window3 )
```

Displays a properties form for each object in the selected set in `window3`.

© 2020 All Rights Reserved.

geRegSaveAllTrigger

```
geRegSaveAllTrigger(
     trigName
     )
     => t / nil
```

Description

Registers a trigger name for the *Save All* form.

Arguments

trigName	Name of the trigger.

Value Returned

t	The trigger with the specified name is saved successfully.
nil	The trigger with the specified name could not be saved as an error is encountered.

Example

```
geRegSaveAllTrigger( "_geiSaveAllCVTrigger" )
```


© 2020 All Rights Reserved.

geResetCellView

```
geResetCellView(
     d_cellview
     )
     => t / nil
```

Description

Re-displays the last saved version of a cellview.

Arguments

d_cellview	Database ID of the cellview to reset.

Value Returned

`t`	The cellview is reset.
`nil`	The cellview is not reset.

Example

```
geResetCellView( cell2 )
```

Reverts to the last saved cellview of `cell2`.

© 2020 All Rights Reserved.

geTransformUserBBox

```
geTransformUserBBox(
     l_bbox
     l_transform
     )
     => l_bBox_new
```

Description

Transforms the specified bounding box by a list describing the offset, orientation, and magnification.

Arguments

l_bBox	Bounding box to be transformed.
l_transform	List the offset, orientation, and magnification of the bBox.

Value Returned

l_bBox_new	The bounding box resulting from transformation.

Example

```
geTransformUserBBox(list(lowerLeft upperRight) list(Offset Orient Mag))

geTransformUserBBox(list(2:3 4:8) list(1:0  "R0" 1.1))
```


© 2020 All Rights Reserved.

geTransformUserPoint

```
geTransformUserPoint(
     l_point
     l_list
     )
     => l_point / nil
```

Description

Transforms a coordinate pair by the displacement and rotation passed in.

For each coordinate pair defined by the first argument, the corresponding coordinate pair in the result is derived by

- Rotating or reflecting the coordinate pair about the point (0,0)
- Translating the result of the reorientation by the displacement vector

`geTransformUserPoint` performs calculations and is not related to any database object. It can be used in calculating, for example, where a certain shape (such as a pin on the master) within an instance of a cell ends up, given the location and orientation of the cell.

Arguments

l_point	Point to be transformed.
l_list	List of the following: displacement vector for the transformation (offset), rotation or reflection for the objects (orientation), and optional magnification. Valid values for rotation and reflection: `R90`, `R180`, `R270`, `MX`, `MY`

Value Returned

l_point	A list of points is returned if the transformation occurs.
`nil`	No transformation occurs.

Example

```
geTransformUserPoint((1.5:2.5) list(0:10 "R90"))
=> (-2.5 11.5)
```


© 2020 All Rights Reserved.

geValidateWindow

```
geValidateWindow(
     [ w_windowId ]
     )
     => w_windowId / nil
```

Description

Verifies the database ID of a window.

Verifies that a graphics cellview is installed in the window. Returns an error if *w_windowId* is not a graphics window.

Arguments

w_windowId	Database ID of the window.

Value Returned

w_windowId	The window exists.
nil	The window does not exist.

Example

```
geValidateWindow( window10 )
```

Returns `window:123456`, the database ID of `window10`.

© 2020 All Rights Reserved.

geWriteCellView

```
geWriteCellView(
     d_cellview
     t_libName
     t_cellName
     t_viewName
     [ t_version ]
     [ d_contextId ]
     )
     => t / nil
```

Description

Creates a copy of a cellview.

Arguments

d_cellview	Database ID of the window containing the design to copy.
t_libName	Library name in which the cellview is created.
t_cellName	New cell name for the design.
t_viewName	New view name for the design.
t_version	This argument is ignored in 4.4 and 5.0, but is left in for compatibility.
d_contextId	Database ID of the context for the cell.

Value Returned

`t`	The cellview is written.
`nil`	The cellview is not written.

Example

```
geWriteCellView( cell1 "mylib" "mycell" "layout" )
```

Creates a copy of the cellview `cell1` in `mylib` with the name `mycell` and the view `layout`.

© 2020 All Rights Reserved.

Microwave Object Functions

geCreateBend

```
geCreateBend(
     d_cellview
     txl_layerPurpose
     l_points
     f_width1
     f_width2
     t_bendStyle
     f_bendFac
     f_radFac
     f_chamFac
     x_nPtsToPI
     )
     => d_bendId / nil
```

Description

Creates a transmission line bend in a cellview.

Arguments

d_cellview	Database ID of the cellview in which the transmission line bend is created.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair. For more information, see Type txl_layerPurpose.
l_points	List of points used to create the transmission line bend. The point list must contain three points.
f_width1	Specifies the width of the first end of the bend. Valid Values: any number greater than zero in user units
f_width2	Specifies the width of the second end of the bend. Valid Values: any number greater than zero in user units

© 2020 All Rights Reserved.

t_bendStyle	Specifies the type of corner to be use for the bend. (Refer to the *Virtuoso Layout Suite L User Guide* for descriptions and illustrations of the bend styles available.) Default Value: `bend` Valid Values: `bend`, `chamfer`, or `radial`
f_bendFac	The maximum width allowed in a standard bend before the bend is automatically chamfered. (Refer to the *Virtuoso Layout Suite L User Guide* for further details.) Default Value: `1` Valid Value: any number
f_radFac	The ratio of the centerline bend width to the transmission line width for a radial bend. Valid Value: any number
f_chamFac	The fraction of the corner that is removed in a chamfer bend. Valid Value: a number between 0 and 1
x_nPtsToPI	The number of segments in a 180-degree radial bend. Default Value: 20 Valid Value: any integer greater than 1

Value Returned

d_bendId	The database ID of the dbLine that represents the transmission line bend if the transmission line bend is created.
`nil`	The transmission line is not created.

Example

```
geCreateBend( "cell2" "metal1R" list(4:4 8:12 4:20) 4.0 6.0 "radial"
1 3.0 0.6 20.0)
```

Creates a transmission line bend in cellview `cell2` on layer `metal1R` along the coordinates `4:4`, `8:12`, and `4:20`. The width of the first end of the bend is `4` and the width of the second end of the bend is `6`. The bend style is `radial`. Returns the database ID of the dbLine that represents the transmission line.

© 2020 All Rights Reserved.

geCreateTaper

```
geCreateTaper(
     d_cellview
     txl_layerPurpose
     l_points
     f_width1
     f_width2
     t_taperStyle
     x_nPtsToTaper
     )
     => d_taperId / nil
```

Description

Creates a tapered transmission line in a cellview.

Arguments

d_cellview	Database ID of the cellview containing the transmission line taper.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair. For more information, see Type txl_layerPurpose.
l_points	Points used to create the transmission line taper. The point list must contain two points.
f_width1	Specifies the width of the first end of the taper. Valid Values: any number greater than zero in user units
f_width2	Specifies the width of the second end of the taper. Valid Values: any number greater than zero in user units
t_taperStyle	Specifies the type of taper to be created. (Refer to the *Virtuoso Layout Suite L User Guide* for descriptions and illustrations of the taper styles available.) Default Value: `linear`. Valid Values: `linear` and `exponential`.
x_nPtsToTaper	The number of segments to use in approximating the shape of the taper. Valid Value: any integer greater than 1

© 2020 All Rights Reserved.

Value Returned

d_taperId	The database ID of the dbLine that represents the tapered transmission line if the transmission line taper is created.
nil	The transmission line taper is not created.

Example

```
geCreateTaper( cell2 "metal1R" list(0:0 12:12) 4 6 linear 20)
```

Creates a tapered transmission line in the cellview `cell2` on layer `metal1R` from 0:0 to 12:12. The width of the first end of the taper is 4, and the width of the second end of the taper is 6. The bend style is *linear* and 20 segments are used to approximate the shape of the taper. Returns the database ID of the dbLine the represents the tapered transmission line.

© 2020 All Rights Reserved.

geCreateTrl

```
geCreateTrl(
     d_cellview
     txl_layerPurpose
     l_points
     f_width
     t_bendStyle
     f_bendFac
     f_radFac
     f_chamFac
     x_nPtsToPI
     )
     => d_trlId / nil
```

Description

Creates a transmission line in a cellview.

Arguments

d_cellview	Database ID of the cellview in which the transmission line is created.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair. For more information, see Type txl_layerPurpose.
l_points	List of points used to create the transmission line.
f_width	Width of the transmission line. Valid Values: any number greater than zero in user units
t_bendStyle	Bend style for the transmission line. (Refer to the *Virtuoso Layout Suite L User Guide* for descriptions and illustrations of the bend styles available.) Default Value: `bend` Valid Values: `bend`, `chamfer`, and `radial`.
f_bendFac	The maximum width allowed in a standard bend before the bend is automatically chamfered. (Refer to the *Virtuoso Layout Suite L User Guide* for further details.) Default Value: `1` Valid Value: any number

© 2020 All Rights Reserved.

f_radFac	The ratio of the centerline bend width to the transmission line width for a radial bend. Valid Value: any number
f_chamFac	The fraction of the corner that is removed in a chamfer bend. Valid Value: a number between `0` and `1`
x_nPtsToPI	The number of segments in a 180-degree radial bend. Default Value: `20` Valid Value: any integer greater than `1`

Value Returned

d_trlId	The database ID for the dbLine that represents the transmission line if the transmission line is created.
`nil`	The transmission line is not created.

Example

```
geCreateTrl( cell2 "metal1R" list(0:0 10:0 10:10) 4.0 "radial" 1.0 3.0 0.6 20)
```

Creates a transmission line in cellview `cell2` on layer `metal1R` along the coordinates `0:0`, `10:0`, and `10:10`. The width of the transmission line is `4` and the bend style is `radial`. Returns the database ID of the dbLine that represents the transmission line.1pf

Mouse Functions

Mouse functions change the mode of the point selection in the current window. These functions are set up to be used as mouse bindkeys.

© 2020 All Rights Reserved.

mouseAddPt

```
mouseAddPt( )
```

Description

Adds the point at the location where you clicked the mouse button to the enter function in the current window.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

mouseAddSelectPt

```
mouseAddSelectPt( )
```

Description

Performs an add mode point selection at the point you click in the current window.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

mouseFinishPt

```
mouseFinishPt( )
```

Description

Adds the point at the location where you clicked the mouse button to the enter function in the current window and completes the enter function, if possible.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

mousePopUp

```
mousePopUp( )
```

Description

Displays the popup menu associated with the given windowId, which must be a valid data structure returned by a call to `hiOpenWindow` or `hiCreateWindow`.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

mouseSingleSelectPt

```
mouseSingleSelectPt( )
```

Description

Performs a single mode point selection at the point you click in the current window.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

mouseSubSelectPt

```
mouseSubSelectPt( )
```

Description

Performs a sub-mode point selection at the point you click in the specified window.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

mouseUndoPt

```
mouseUndoPt( )
```

Description

Deletes the point at the location where you clicked the mouse button to the enter function and cancels the enter function in the current window.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

2

Database Access Functions

Before reading this chapter, you need to be familiar with the basics of the SKILL language and the concepts used in the design framework II unified database. This chapter describes the SKILL functions used to create, modify, and delete Cadence® database objects.

- Data Stored as Objects
  - Database Types
  - Object Identifier (dbObject)
  - Object Types and Classes
  - Relationships Among Objects
  - Access Function Argument Types
- Inherited Connections Functions
- Undo and Redo Functions
- Pcell Freeze Functions
- Fluid Shape Functions
- Pcell Abutment Support Functions
- Shape Creation and Retrieval Functions
- Shape Conversion Functions
- Mosaic Creation and Retrieval Functions
- Property Creation and Modification Functions
- Object Deletion Functions
- Group Creation and Modification Functions
- Shape Transformation Functions
- Bounding Box Functions

© 2020 All Rights Reserved.

- Property Bag Functions
- Layer Boolean Functions
- Point Array Functions
- Soft Connect Terminal Functions
- Information Retrieval Functions
- Blockage Functions
- Boundary Functions
- Bus Definition and Bit Order Functions
- Cluster Functions
- Counter Functions
- Interconnect Functions
- Patch Functions
- Marker Functions
- Ruler Functions
- Query Functions
- Row Functions
- Row Header Functions
- Steiner Functions
- Supply and Ground Sensitivity Functions
- Terminal Ordering Functions
- Terminal Antenna Functions
- Track Pattern Functions
- GCell Pattern Function
- Via Functions
- Optical Data Functions

© 2020 All Rights Reserved.

Data Stored as Objects

All Cadence tools use the Design Framework II unified database; a binary database that stores data as “objects.” There are many types of objects, each representing a distinct concept in the world of electronic design automation. Examples of object types include rectangles, terminals, instances, and cellviews.

The Design Framework II database can store both physical and logical information about a design. Physical information is stored in objects such as geometrical shapes and an IC layout. Logical information is stored in objects such as nets or schematics, which can exist without any corresponding physical realizations.

The database allows modification in read-only mode to support upgrade, bookkeeping, and analysis operations on read-only cellviews. However, these cellviews can be saved only when they are re-opened in an editable mode.

Database Types

Design Framework II supports two database types, CDB and OpenAccess. There is a function to determine which database type is supported by the current executable.

dbGetDatabaseType

```
dbGetDatabaseType(
     )
     => CDBA / OpenAccess
```

Description

Returns `CDBA` when called from a regular CDB executable, and `OpenAccess` when called from an OpenAccess executable.

Arguments

None

© 2020 All Rights Reserved.

Value Returned

`CDBA`	When called from a regular CDB executable.
`OpenAccess`	When called from OpenAccess.

Object Identifier (dbObject)

To access an object in the database, you must use a label known as the *object identifier* or *ID*. Each ID uniquely identifies a database object and is represented by the special type dbObject. Only database routines can create IDs, and you cannot alter IDs directly. When you use a function that operates on a database object, you must also give the ID as an argument to the function.

In programming terms, a variable of dbObject type represents the ID of an object in the database. If you delete the database object identified by a dbObject variable, that dbObject variable is invalid and cannot be used to do any database operation.

The *delete* and inter-cellview *move* commands in the Graphics Editor can destroy existing database objects and invalidate a dbObject ID. In addition, closing the last reference to a cellview (for example, closing the last Graphics Editor window the cellview is displayed in or directly calling `dbClose` or `dbPurge`) purges the cellview from virtual memory. This action makes obsolete all IDs of database objects stored in that cellview.

Object Types and Classes

Each object is associated with a type, and each object type has a set of attributes that describe the object.

An object class is a data-type abstraction that groups related object types into a class. Object classes are created when a number of distinct object types share enough attributes that they can be discussed as a single class, without being concerned about the exact object type.

The different types and classes of objects form a class hierarchy. At the top of the hierarchy is a class containing all types. At the bottom of the hierarchy, each leaf node represents an object type that can be created, deleted, and saved on disk. Each node in the hierarchy has attributes that are common to all of its children. Geometrical shapes, for example, have the common attributes of layer, purpose, and bounding box.

© 2020 All Rights Reserved.

Object Attributes

Each object type has a predefined set of attributes that you can retrieve using type-specific access functions. There are three types of attributes:

- Mandatory

  Mandatory attributes must be specified at the time you create an object. Most attributes fall into this category.

- Optional

  Optional attributes may or may not exist for a particular object. These attributes do not need to be specified at the time you create an object. You can add them to the object at any time.

- Derived

  These attributes are derived from the object's other attributes. You have read-only access to derived attributes. Derived attributes cannot be stored in the database, but they play an important role in speeding up repetitive computation.

A complete list of attributes is found starting with Attribute Types.

Object Properties

An object property has a name and a value, such as a Boolean value, an integer, a floating-point number, a string, or a representation of time. You can attach an arbitrary number of properties to an object. Use properties to store different kinds of information and as mechanisms for extending the basic schema of the database.

Differences Between Attributes and Properties

The most important distinction between attributes and properties is that attributes are predefined and managed by the database, while properties are defined and managed by applications built on top of the database. This distinction is often blurred because some information currently stored as properties is also essential to the system's operation.

Properties are not mandatory as far as the database is concerned; however, some properties are essential to the operation of certain applications (for example, netlisting, Place and Route, and so forth). Within these applications, such properties can be considered mandatory.

The database access operator (~>) works on both attributes and properties associated with a given database object. Because the name of an attribute or a property can be specified on the right-hand side of the operator, attributes and properties share a single name space. For

© 2020 All Rights Reserved.

this reason, in earlier documentation attributes and properties were lumped together and referred to as “properties.” For the “~>” operator, attributes always have precedence over properties, which means that you cannot access a property that has the same name as an attribute using the “~>” operator.

Relationships Among Objects

The Cadence database stores many predefined relationships among different types of objects. Some relationships are automatically kept by the database, while other inter-object relationships must be explicitly created and maintained by applications.

Some relationships in the database are mandatory, for example, each terminal must be attached to a net. Application programs cannot, therefore, create a terminal without specifying the net it connects to. Mandatory relationships are often unidirectional; for example, it is acceptable to have a net that does not connect to a terminal.

Some relationships in the database are optional. Objects can be created independently now with a relationship created later by an application.

When a predefined relationship is one-to-one or many-to-one, there is an access function (in the form of a pseudo-attribute) to go from the source object to the object it relates to. When a predefined relationship is a one-to-many relationship, there is an access function (in the form of a pseudo-attribute) that returns a list of the objects contained in the relationship.

Access Function Argument Types

A function argument is of a particular type, which are obvious from the name of the argument. The following list describes those types that are not obvious from the Description.

Type l_bBox

A list of two *l_point* arguments describing the bounding box of an object. The first point in the list is the xy coordinates of the lower-left corner of the bounding-box; the second point is the xy coordinates of the upper-right corner.

Examples:

```
list(list(0 0) list(10 100))
list(0:0 10:100)
```


© 2020 All Rights Reserved.

Type l_group

A list of strings describing a group's type. The recognized strings and their definitions are

`collection`	The group can contain a *dbObject* more than once.
`set`	The group can contain a *dbObject* only once.
`ordered`	The order of the group members is significant. You establish and maintain the ordering through calls to add *dbObjects* to a group and calls to move group members within an ordered group. The initial order is determined by the sequence in which *dbObjects* are added to the ordered group.
`unordered`	The order of the group members is insignificant.
`nonUniqueName`	Many groups within the same cellview have the same name.
`uniqueName`	Each group within the same cellview has a unique name.
`deleteNever`	Do not delete the group automatically.
`deleteFirst`	Automatically delete the group when the first group member is deleted.t
`deleteLast`	Automatically delete the group when the last group member is deleted.

Type l_LP

A list containing the layer and purpose.

Example:

```
list( tx_layer tx_purpose)
```

Type l_point

A list of two coordinates describing an xy location. Each coordinate is defined as an integer.

Examples:

```
list(0 100)
0:100
```

A list of *l_point* arguments.

Note: Virtuoso supports a maximum of 4000 points or vertices.

© 2020 All Rights Reserved.

Examples:

```
list(list(0 0) list(10 10) list(100 100))
list(0:0 10:10 100:100)
```

Type l_string

A list of strings. There is no restriction on the string values.

Example:

```
list("dog" "cat" "horse" "")
```

Type l_transform

A list describing the transformation. The list contains the following elements in the specified order:

`offset`	The offset from the object's origin described as an *`l_point`*.
`orient`	The orientation of the object described as a *`t_orient`*.
`mag`	The magnification of the object described as a floating-point number. This argument is optional. If it is not specified. Defaults to 1.0.

Examples:

```
list(10.0:2.0 "R90" 2.0)
list(list(10.0 2.0) "R90")
```


© 2020 All Rights Reserved.

Type tx_layer

A string or integer describing a layer.

Example:

```
"hilite" or 254
```

Type txl_layerPurpose

A string, integer or list. If it is a string or integer then the string or integer is used to describe the layer and the purpose is assumed as `drawing`. If it is a list, then the list must be of a name or number describing a layer and a name or number describing a purpose. If a purpose is not defined, it defaults to `drawing`.

Type t_direction

A string describing a terminal's direction. The recognized strings are `input`, `inputOutput`, `jumper`, `output`, `switch`, `unused`, `powerInput`, `powerOutput`, `bidir`, and `tristate`.

Type t_font

A string describing a label's font. The recognized strings are *euroStyle* (or *Europe*), *gothic*, *math*, *roman*, *script*, *stick*, *swedish*, *fixed*, and *raster*.

Type t_just

A string describing a label's justification. The recognized strings are *lowerLeft*, *lowerCenter*, *lowerRight*, *centerLeft*, *centerCenter*, *centerRight*, *upperLeft*, *upperCenter*, *upperRight, leftTop, leftCenter, leftBottom, centerTop*, *centerBottom, rightTop, rightCenter*, and *rightBottom*.

Type t_orient

A string describing an object's orientation. The recognized strings are *0*, *90*, *180*, *270*, *sideways*, *sideways&90*, *upsideDown*, *sideways&270*, *R0*, *R90*, *R180*, *R270*, *MY*, *MYR90*, *MX*, and *MXR90*.

© 2020 All Rights Reserved.

Type t_pathStyle

A string describing a path's style. The recognized strings are *extendExtend*, *roundRound*, *truncateExtend*, *varExtendExtend*, *squareFlush*, *squareOffset*, and *octagonEnded*.

Type t_prop

A string describing a property's type. The recognized strings are: *boolean*, *int*, *fileName*, *float*, *hierProp*, *ILExpr*, *ILList*, *NLPExpr*, *string*, *time*, *fixnum*, *integer*, *flonum*, *Boolean*, *ilExpr*, *nlpExpr*, *property*, and *ilList*.

Type tx_purpose

A string or number describing a purpose.

Example

```
"drawing" or 252
```


© 2020 All Rights Reserved.

dbIsId

```
dbIsId(
     d_id
     )
     => t / nil
```

Description

Returns `t` if the argument is a valid database object ID, otherwise, it returns `nil`.

Arguments

d_id	Database object ID.

Value Returned

`t`	The argument is a valid database object ID in the current session.
`nil`	The argument is an invalid database object ID in the current session.

Example

```
dbIsId( instID )
```


© 2020 All Rights Reserved.

dbValidP

```
dbValidP(
     d_id
     )
     => t / nil
```

Description

Determines if the argument is a valid database object ID.

The two functions `dbValidP` and `dbIsId` perform the same action. Both are kept active for compatibility.

Arguments

d_id	Database ID of the object.

Value Returned

`t`	The argument is a valid database object ID.
`nil`	The argument is not a valid database object ID.

© 2020 All Rights Reserved.

dbobjectp

```
dbobjectp(
     d_id
     )
     => t / nil
```

Description

Determines if its argument is a database object.

Arguments

d_id	The object can be an instance, a net, or a terminal, and is specified either by the database ID of the object or by a list of strings describing a path to the object. If the database ID of the object is used, the path is derived from the hierarchical path from the top level of *w_windowId* to the object.

Value Returned

t	The argument is a valid database object
nil	The argument is not a valid database object.

Example

```
dbobjectp( netId ) => t
```


© 2020 All Rights Reserved.

Inherited Connections Functions

Inherited connections is an extension to the connectivity model that allows you to create global signals and override their names for selected branches of the design hierarchy. This flexibility allows you to use

- Multiple power supplies in a design
- Substrate connections that can be overridden
- Parameterized power and ground symbols

To learn about using inherited connections and net expressions with various Cadence® tools in the design flow, refer to the *Inherited Connections Flow Guide*.

To learn about connectivity and naming conventions for inherited connections and how to add and edit net expressions in a schematic or symbol cellview, refer to the *Composer: Design Entry User Guide*.

The following section deals with the Design Framework II SKILL Access Functions for inherited singals and inherited terminals.

© 2020 All Rights Reserved.

dbCreateSigNetExpr

```
dbCreateSigNetExpr(
     d_sigId
     t_netExpression
     )
     => t / nil
```

Description

Given an inherited net expression in *t_netExpression* and a signal id in *d_sigId,* this function attaches the inherited net expression to the given signal and adds the signal to the set of signals having inherited net expressions.

Arguments

d_sigId	Specifies a signal that has the inherited net expression attached to it.
t_netExpression	Specifies the inherited net expression that gets attached to the signal.

Value Returned

`t`	The inherited net expression is successfully attached to the signal.
`nil`	The given signal does not exist, the given net expression is of illegal syntax, or if the signal already has an inherited net expression associated with it.

© 2020 All Rights Reserved.

dbCreateTermNetExpr

```
dbCreateTermNetExpr(
     d_termId
     t_netExpression
     )
     => t / nil
```

Description

Given an inherited net expression in *t_netExpression* and a term ID, in *d_termId*, this function attaches the inherited net expression to the given terminal and adds the terminal to the set of terminals having inherited net expressions.

Arguments

d_termId	Specifies a terminal that has the inherited net expression attached to it.
t_netExpression	Specifies the inherited net expression that attaches itself to the terminal.

Value Returned

`t`	The operation is successful.
`nil`	The given terminal does not exist, the given net expression is of illegal syntax, or if the terminal already has an inherited net expression associated with it.

© 2020 All Rights Reserved.

dbCVHasInheritedSig

```
dbCVHasInheritedSig(
     d_cellViewid
     )
     => t / nil
```

Description

Checks if there is any inherited signal in the given cellview.

Argument

d_cellViewId	Specifies the cellview ID.

Value Returned

`t`	There is an inherited signal in the given cellview.
`nil`	There is no inherited signal or the given cellview cannot be accessed.

© 2020 All Rights Reserved.

dbCVHasInheritedTerm

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

Description

Checks whether there is any inherited terminal in the given cellview.

Arguments

d_cellviewId	Specifies the cellview ID.

Value Returned

t	There is an inherited terminal in the given cellview.
nil	There is no inherited terminal, or the given cellview cannot be accessed.

© 2020 All Rights Reserved.

dbCVHasNetSetProp

```
dbCVHasNetSetProp(
     d_cellviewId
     )
     => t / nil
```

Description

Determines if the netSet property is set on a cellview.

Arguments

d_cellviewId	ID of the source cellview.

Value Returned

t	The property is set on the cellview.
nil	The property is not set.

Example

```
cv = dbOpenCellViewByType( "netSetLib" "test1" "schematic" "schematic" "w")
dbCVHasNetSetProp(cv) ==> t
```


© 2020 All Rights Reserved.

dbDeleteSigNetExpr

```
dbDeleteSigNetExpr(
     d_sigId
     )
     => t / nil
```

Description

Given a signal IDin *d_sigId*, this function deletes any existing inherited net expression associated with this signal and removes it from the set of inherited signals.

Arguments

d_sigId	Specifies a signal that has the inherited net expression attached to it.

Values Return

t	The inherited net expression exists on the object and is successfully deleted.
nil	No inherited net expression belongs to the object, or if the given signal cannot be found.

© 2020 All Rights Reserved.

dbDeleteTermNetExpr

```
dbDeleteTermNetExpr(
     d_termId
     )
     => t / nil
```

Description

Given *d_termId*, this function deletes any existing inherited net expression associated with this terminal and removes it from the set of inherited terminals.

Arguments

d_termId	Specifies a terminal that has the inherited net expression attached to it.

Value Returned

t	The inherited net expression exists on the object and is successfully deleted.
nil	No inherited net expression belongs to the object, or if the given terminal cannot be found.

© 2020 All Rights Reserved.

dbFix

```
dbFix(
     d_cellViewId
     [ ?reBind g_reBind ]
     )
     => t / nil
```

Description

Fixes data corruption in inherited connectivity, rectangle, and instance header. This is a no-op function.

Arguments

d_cellViewId	Specifies the cellview ID.
g_reBind	Specifies whether rebinding is needed.

Value Returned

`t`	The data corruption was fixed.
`nil`	The data corruption was not fixed.

© 2020 All Rights Reserved.

dbGetCellViewInheritedSig

```
dbGetCellViewInheritedSig(
     d_cellViewId
     )
     => l_sigId / nil
```

Description

Retrieves from the given cellview all signals to which inherited net expressions are attached.

Arguments

d_cellViewId	Specifies the cellview ID.

Value Returned

l_sigId	A list of signals that have inherited expressions.
nil	There are no inherited signals in the given cellview.

© 2020 All Rights Reserved.

dbGetCellViewInheritedTerm

```
dbGetCellViewInheritedTerm(
     d_cellViewId
     )
     => l_termId / nil
```

Description

Retrieves from the given cellview all terminals to which inherited net expressions are attached.

Arguments

d_cellViewId	Specifies the cellview ID.

Value Returned

l_termId	A SKILL list of terminals that have inherited expressions.
nil	There are no inherited terminals in the given cellview.

© 2020 All Rights Reserved.

dbGetSigNetExpr

```
dbGetSigNetExpr(
     d_sigId
     )
     => t_netExpression / nil
```

Description

Retrieves any inherited net expression found attached to the signal.

Arguments

d_sigId	A signal.

Value Returned

t_netExpression	The inherited net expression.
nil	No inherited net expression is found, or the given signal cannot be found.

© 2020 All Rights Reserved.

dbGetTermNetExpr

```
dbGetTermNetExpr(
     d_termId
     )
     => t_netExpression / nil
```

Description

Retrieves any inherited net expression found attached to the terminal.

Arguments

d_termId	A terminal.

Value Returned

t_netExpression	The inherited net expression.
nil	No inherited net expression is found, or the given terminal cannot be found.

© 2020 All Rights Reserved.

dbIsValidSigNetExpr

```
dbIsValidSigNetExpr(
     t_netExpression
     )
     => t / nil
```

Description

Given an inherited net expression in the *t_netExpression* associated with a signal, this function checks whether its syntax is valid. This function does as much local checking as possible, but it does not detect whether the given default net name is global, nor whether the evaluation of the property results in a valid net name.

Arguments

t_netExpression	The inherited net expression that gets attached to a signal.

Value Returned

`t`	The syntax is valid.
`nil`	The given inherited net expression is of illegal syntax.

Example

```
[@vdd:%:vdd!] is valid syntax.
[@vdd:%] is illegal because the required default net name is missing.
[&vdd:%:vdd!] is illegal because the & character is not a valid search scope
designator.
```


© 2020 All Rights Reserved.

dbIsValidTermNetExpr

```
dbIsValidTermNetExpr(
     t_netExpression
     )
     => t / nil
```

Description

Given an inherited net expression in *t_netExpression* associated with a terminal, this function checks whether its syntax is valid. This function does as much local checking as possible, but it does not detect whether the given default net name is global nor whether the evaluation of the property results in a valid net name.

Arguments

t_netExpression	The inherited net expression that gets attached to a terminal.

Value Returned

t	The syntax is valid.
nil	The given inherited net expression is of illegal syntax.

Example

```
[@bulk:%:gnd!] is legal syntax.
[@bulk:%] is illegal because the required default net name is missing.
[&bulk:%:bulk!] is illegal because the & character is not a valid search scope
designator.
```


© 2020 All Rights Reserved.

dbIsSigInherited

```
dbIsSigInherited(
     d_sigId
     )
     => t / nil
```

Description

Given a signal id in *sigId*, this function checks if there is an inherited net expression attached to it.

Arguments

d_sigId	A signal.

Value Returned

`t`	An inherited net expression is found to be attached to the given signal.
`nil`	There is no inherited net expression attached to the signal, or the given signal cannot be found.

© 2020 All Rights Reserved.

dbIsTermInherited

```
dbIsTermInherited(
     d_termId
     )
     => t / nil
```

Description

Checks whether there is an inherited net expression attached to the given terminal.

Arguments

d_termId	A terminal.

Value Returned

`t`	An inherited net expression is found to be attached to the given terminal.
`nil`	There is no inherited net expression attached to the , or the given terminal cannot be found.

© 2020 All Rights Reserved.

dbParseSigNetExpr

```
dbParseSigNetExpr(
     t_netExpression
     )
     => t_searchScope
        t_propName
        t_formatString
        t_defaultNetName / nil
```

Description

Parses the expression and returns the individual components of the expression that include the search scope, the property name to be used in searching, the format string, and the default net name.

Arguments

t_netExpression	The inherited net expression that gets attached to a signal.

Value Returned

t_searchScope	Search scope.
t_propName	The property name to be used in searching.
t_formatString	The format string.
t_defaultNetName	The default net name.
`nil`	The given inherited net expression is of illegal syntax.

Example

Given the net expression found in the *schematic* view of the cell *inv*, `[@vdd:%:vdd!]`, this function returns

```
list("@" "vdd" "%" "vdd!")
```


© 2020 All Rights Reserved.

dbParseTermNetExpr

```
dbParseTermNetExpr(
     t_netExpression
     )
     => t_searchScope
          t_propName
          t_formatString
          t_defaultNetName / nil
```

Description

Parses the expression and returns the individual components of the expression that include the search scope, the property name to be used in searching, the format string, and the default net name.

Argument

t_netExpression	The inherited net expression that gets attached to a terminal.

Value Returned

t_searchScope	Search scope.
t_propName	Property name to be used in searching.
t_formatString	Format string.
t_defaultNetName	Default net name.
nil	The given inherited net expression is of illegal syntax.

Example

```
list("@" "vdd" "%" "vdd!")
```

Given the net expression found in the schematic view of the cell *inv*, `[@vdd:%:vdd!]`, this function returns

© 2020 All Rights Reserved.

dbReplaceSigNetExpr

```
dbReplaceSigNetExpr(
     d_sigId
     t_netExpression
     )
     => t / nil
```

Description

Replaces any existing inherited net expression already associated with the signal. If none exists, *t_netExpression* is attached to the given signal and adds the signal to the set of signals having inherited net expressions.

Arguments

d_sigId	A signal.
t_netExpression	The inherited net expression that gets attached to a signal.

Value Returned

`t`	The inherited net expression exists on the object and is successfully replaced or created.
`nil`	The given signal cannot be found, or the given inherited net expression is of illegal syntax.

© 2020 All Rights Reserved.

dbReplaceTermNetExpr

```
dbReplaceTermNetExpr(
     d_termId
     t_netExpression
     )
     => t / nil
```

Description

Replaces any existing inherited net expression already associated with the terminal. If none exists, *t_netExpression* is attached to the given terminal and adds the terminal to the set of terminals having inherited net expressions.

Arguments

d_termId	A terminal.
t_netExpression	The inherited net expression that gets attached to a terminal.

Value Returned

`t`	The inherited net expressions exists on the object and is successfully replaced or created.
`nil`	The given terminal cannot be found, or if the given inherited net expression is of illegal syntax.

© 2020 All Rights Reserved.

Net and Terminal Name Syntax

The following section describes name syntax for both net and terminal names. The name syntax furnished below is in Backus Nour Format.

`[ ]`	Items enclosed in brackets are optional where zero or, at most, one item is present.
`{ }`	Items enclosed in curly braces are also optional, but zero or more items can be present.
`|`	The vertical bar means *or.*
`N`	Defined as a nonnegative integer less than 64K (65535).
`nameExpr`	A string containing a name term optionally prefixed with the <*N> construct. This construct is also referred to as a *prefix repeat operator.* This means the name term is repeated (enumerated) N times. Name expressions can be concatenated to form a comma-separated list.
`nameTerm`	A base name optionally followed by a vector expression contained in angle brackets. A name term can also be a name expression enclosed in parenthesis.
`baseName`	A string of printable ASCII characters which does not include reserved special characters such as angle brackets (< >), parentheses (()), comma (,), slash (/), or white space. Parentheses (()) are allowed in a base name only if they enclose a number.
`vectExpr`	Either a single or repeated vector term. The optional *N suffix on a vector term is called a *suffix repeat operator.* It indicates N repetitions of each bit of the vector term. The value of N must be greater than zero. Vector expressions can also be concatenated to form a comma-separated list.
`vectTerm`	A single number or a range of numbers. A range of numbers can be expressed with a starting, ending, and increment number separated by colons. A vector term can also be a vector expression enclosed in parentheses. In this case, the innermost term is expanded first and the repeat operator is then applied to repeat *the entire expanded term* enclosed in parentheses.

Examples:

```
nameExpr ::= [ < *N > ] nameTerm { , nameExpr }
nameTerm ::= baseName [ < vectExpr > ] | < nameExpr >
vectExpr ::= vectTerm [ *N ] { , vectExpr }
vectTerm ::= N [ : N [ : N ] ] | < vectExpr >
```


© 2020 All Rights Reserved.

Figure 2-1 illustrates how name and vector expressions are expanded.

Figure 2-1 Expanded Net/Terminal Names

Name	Members	Expanded format
clk	1	clk
data<2>	1	data<2>
<*1>base	1	base
<*2>term	2	term,term
<*2>(a,b),c	5	a,b,a,b,c
<*2>(a,<*2>b)	6	a,b,b,a,b,b
b<0:2>	3	b<0,1,2> or b<0>,b<1>,b<2>
b<0:2:1>	3	b<0,1,2>
b<3:0:2>	2	b<3,1>
b<0:2*2>	6	b<0,0,1,1,2,2,>
b<(0:2)*2>	6	b<0,1,2,0,1,2>
b<0,2*2>	3	b<0,2,2>
b<(0,2)*2>	4	b<0,2,0,2>
b<0,1:3:4*1,2:2>	3	b<0,1,2>
b<0:1,2:2>	3	b<0,1,2>

As shown, the expanded form of a name is semantically equivalent to the original specification. This form is created by expanding the range specifications, applying repeat operators, and removing parentheses.

To determine the *N*th member name, use a zero-based counting scheme to find the *N*th element in the expanded form. For example, member zero of *<*2>(a,<*2>b)* is *a,* member one is *b,* member two is *b,* and so on. If the name contains a vector expression, such as *b<0:1,2:2>*, then member zero in this case is called *b<0>,* member one is *b<1>,* and member two is *b<2>.*

© 2020 All Rights Reserved.

Instance Name Syntax

The instance name syntax below is in BNF (Backus-Naur Format) notation. Items enclosed in brackets ([ ]) specify the item is optional where zero items or one item might be present. Items enclosed in curly braces ({}) are also optional, except zero or more items can be present. The word `number` indicates a nonnegative integer less than 65536.

```
instName ::= baseName [ < number [ : number ] > ]
```

An instance name (`instName`) is a base name that is optionally followed by a simple vector term. The restrictions on the base name and number are the same as the ones for the net and terminal name syntax with some minor restrictions.

The simple vector term can consist of a single number or imply a range of numbers. The range can be expressed by a starting and ending number separated by colons. The vector term has the same expanded form as the net and terminal names except that for instance names, increment, repeat operators and parenthetic grouping are not allowed.

The instance name must be unique at both the full instance and member name levels. A single instance with a name that implies sharing a member between full instances is not allowed.

© 2020 All Rights Reserved.

Undo and Redo Functions

This section describes various undo and redo handle functions.

dbSetUndoHandle

```
dbSetUndoHandle(
     t_handleName
     [ g_moveHandle ]
     )
     => t / nil
```

Description

Sets an undo handle with the specified name on the current undo checkpoint. If there already exists a handle with the same name, you can optionally specify that the existing handle is moved from its current position in the undo stack and associated with the current undo checkpoint.

Arguments

t_handleName	Name of the handle name that needs to be set as the undo handle.
g_moveHandle	Controls what happens if there is an existing undo checkpoint with the same name. ■ When set to `t`, the existing handle is moved to the current undo checkpoint and the undo checkpoint to which it was previously assigned becomes unnamed. ■ When set to `nil`, if there is an existing undo checkpoint with the same name, an error is returned because the handle names must be unique.

© 2020 All Rights Reserved.

Value Returned

`t`	The handle was set.
`nil`	The handle was not set.

Example

```
dbSetUndoHandle("checkpoint5")
```

Sets the undo handle, `checkpoint5` on the current undo checkpoint.

```
dbSetUndoHandle("checkpoint5" t)
```

Sets the undo handle, `checkpoint5` on the current undo checkpoint. The existing handle, `checkpoint5`, is moved to the current undo checkpoint and the undo checkpoint to which it was previously assigned becomes unnamed.

© 2020 All Rights Reserved.

dbUndoToHandle

```
dbUndoToHandle(
     [ t_handleName ]
     )
     => t / nil
```

Description

Invokes undo back to the given handle. If no handle name is given, rolls back to the last labeled checkpoint.

Arguments

t_handleName	Handle name that was set using dbSetUndoHandle.

Value Returned

`t`	The undo was invoked.
`nil`	The undo was not invoked.

Example

```
dbUndoToHandle("checkpoint5")
dbUndoToHandle()
```

Rolls back to the undo handle `checkpoint5`.

© 2020 All Rights Reserved.

dbGetUndoHandles

```
dbGetUndoHandles(
     )
     => l_handlelist
```

Description

Returns a list of handles in the undo stack.

Arguments

None

Value Returned

l_handlelist	The list of handles in the undo stack.

Example

```
dbGetUndoHandles()
```


© 2020 All Rights Reserved.

dbGetRedoHandles

```
dbGetRedoHandles(
     )
     => l_handlelist
```

Description

Returns a list of handles in the redo stack.

Arguments

None

Value Returned

`l_handlelist`	The list of handles in the redo stack.

Example

```
dbGetRedoHandles()
```


© 2020 All Rights Reserved.

dbRedoToHandle

```
dbRedoToHandle(
     [ t_handleName ]
     )
     => t / nil
```

Description

Invokes redo up to the given handle. If no handle name is given, it rolls forward to the next labeled checkpoint.

Arguments

t_handleName	The handle name.

Value Returned

`t`	The redo was invoked.
`nil`	The redo was not invoked.

Example

```
dbRedoToHandle("my_checkpoint")
```


© 2020 All Rights Reserved.

dbEnableUndo

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

Description

Enables undo and redo operations in a Virtuoso session.

Arguments

None

Value Returned

`t`	The undo and redo operations were enabled.
`nil`	The undo and redo operations were not enabled.

Example

```
dbEnableUndo()
```


© 2020 All Rights Reserved.

Pcell Freeze Functions

dbFreeze

```
dbFreeze(
     d_topCellViewId
     t_regLibName
     t_pcellLibName
     [ g_cleanUp ]
     )
     => d_figGroupId / nil
```

Description

Freezes a pcell design hierarchy into a non-pcell design hierarchy by converting a pcell instance into a static instance and converting a pcell customVia into a static customVia. If `t_regLibName` and `t_pcellLibName` are different, regular and pcell frozen instances will be put into two different libraries.

Arguments

d_topCellViewId	The top design cellview ID.
t_regLibName	The name of a new library that will be used to store the regular frozen design hierarchy. The specified library should not exist.
t_pcellLibName	The name of a new library, which is used to store the pcell frozen design hierarchy
g_cleanup	Optional argument. When set to `t`, removes unused instances. Default is `nil`.

Value Returned

d_figGroupId	Returns `t` if freezes a pcell design hierarchy successfully.
`nil`	If the specified library exists, an error is issued and `nil` is returned.

Example

```
cvId = dbOpenCellView("testlib" "Atop" "layout")
when( !cvId
```


© 2020 All Rights Reserved.

```
   info("Failed to open design(testlib Atop layout) in 'r' mode.\n")
   return(nil)
)
when( !dbFreeze(cvId "fzLib" "pcellLib")
info("Failed to freeze design(testlib Atop layout)
   hierarchically by
      dbFreeze().\n")
   return(nil)
)
```


© 2020 All Rights Reserved.

Fluid Shape Functions

This section covers the following fluid shape functions:

- dbGetFluidShapeByName
- dbGetFluidShapeName
- dbGetFluidShapes
- dbIsFluidShape
- dbSetFluidShape

© 2020 All Rights Reserved.

dbGetFluidShapeByName

```
dbGetFluidShapeByName(
     d_cellViewId
     t_shapeName
     )
     => d_fluidShapeId
```

Description

Returns the database ID for a specified fluid shape name in a given cellview.

Arguments

d_cellViewId	Database ID of the cellview.
t_shapeName	Name of the fluid shape.

Value Returned

d_fluidShapeId	Database ID of the fluid shape.

Example

```
rect1 = dbGetFluidShapeByName(inst~>master "fluidShape2")
```


© 2020 All Rights Reserved.

dbGetFluidShapeName

```
dbGetFluidShapeName(
     d_shapeId
     )
     => t_FluidName / nil
```

Description

Returns the fluid name of the specified shape if it is a fluid shape.

Arguments

d_shapeId	Database ID of a shape.

Return Value

t_FluidName	Fluid name of the specified fluid shape.
nil	The given shape is not a fluid shape.

Example

```
name = dbGetFluidShapeName(rect1)
```

Returns the fluid name of the shape `rect1`.

© 2020 All Rights Reserved.

dbGetFluidShapes

```
dbGetFluidShapes(
     d_cvId
     )
     => l_fluidShapes
```

Description

Returns all fluid shapes on the specified cellview`dbGetFluidShapes`.

Arguments

d_cvId	Database ID of a cellview.

Return Value

l_fluidShapes	List of all fluid shapes on the specified cellview.

Example

```
fluidShapes = dbGetFluidShapes(inst~>master)
```

Returns list of fluid shapes on master cellview.

© 2020 All Rights Reserved.

dbIsFluidPcell

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

Description

Checks if submaster belongs to a Fluid Pcell supermaster.

Arguments

d_cellViewId	Database Id of a cellview.

Value Returned

`t`	Specified submaster belongs to a Fluid Pcell supermaster.
`nil`	Specified submaster does not belong to a Fluid Pcell supermaster.

Example

```
dbIsFluidPcell(cvId) => t
```

Cellview `cvId` belongs to a Fluid Pcell supermaster.

© 2020 All Rights Reserved.

dbIsFluidShape

```
dbIsFluidShape(
     d_shapeId
     )
     => t / nil
```

Description

Checks if the specified shape is marked as a Fluid shape.

Arguments

d_shapeId	Database ID of a shape.

Return Value

t	Specified shape is a fluid shape.
nil	Specified shape is not a fluid shape.

Example

```
when(dbIsFluidShape(rect2) printf("Shape: %L is a fluid shape.\n" rect2~>??)
```

If the shape `rect2` is a fluid shape, prints the following:

```
Shape: rest2 is a fluid shape.
```


© 2020 All Rights Reserved.

dbSetFluidShape

```
dbSetFluidShape(
     d_shapeId
     t_fluidName
     )
     => t / nil
```

Description

Sets the given name for the specified shape to mark it as a fluid shape. This function can be used only in a sub-master layout view, for example, during Pcell generation.

Arguments

d_shapeId	Database ID of a shape to be marked as a fluid shape
t_fluidName	The name to mark the specified shape as a fluid shape.

Return Value

`t`	Specified name is set for the specified shape to mark it as a fluid shape.
`nil`	The operation failed.

Example

```
dbSetFluidShape(rect2 "fluidShape2")cv
```

Sets the name `fluidShape2` on the shape `rect2` to mark it as a fluid shape.

© 2020 All Rights Reserved.

Pcell Abutment Support Functions

The following functions describe how to set, unset, and retrieve the values of two abutment properties, *abutment name* and *abutment reference name*, supported in figures (pcells). The only kinds of figures that these functions work with are shapes, vias, and instances.

The figures owned by a pin can be identified with two names, a pinFigName and an abutment name. An *abutment name* is a unique value that you assign to a figure in a subMaster design. You can search for a specific figure based on the assigned abutment name.

In addition, you can specify a name for the reference shape of a figure. This name is called an *abutment reference name*. The abutment reference name need not be unique because multiple shapes can refer to the same reference shape.

Note: The abutment reference name on a shape must not be identical to its abutment name, as a shape cannot be its own reference shape.

© 2020 All Rights Reserved.

dbAbutIsInteractive

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

Description

This function is used by user-defined abutment functions. It indicates if the call to the abutment function was triggered by an interactive action.

Arguments

None

Value Returned

t	Returns `t` if the abutment being called is interactive.
`nil`	Returns `nil` if the abutment being called is not interactive.

Example

Following abutment function uses the `dbAbutIsInteractive` SKILL function:

```
userAbutFunction(
 ...
 case 1: ...
 case 2: ;; this is where abutment happens
         ...
         dbAbutIsInteractive()
 ...
)
```

The SKILL function returns `t` if the abutment is triggered by an interactive action such as a `move` command. It returns `nil` if the abutment is triggered by a non-interactive action such as chaining.

© 2020 All Rights Reserved.

dbAbutIsReAbutGroup

```
dbAbutIsReAbutGroup(
     d_groupID
     )
     => t / nil
```

Description

When called during abutment event 3 (un-abutment), the SKILL function indicates if re-abutment of Pcell instances will be triggered immediately after un-abutment. If the SKILL function returns `true`, the PDK SKILL function can store relevant information, if needed, from the current abutment group.

Arguments

d_groupID

Database ID of the group going through un-abutment (event 3).

Value Returned

`t`	If the SKILL function is called during event 3 (un-abutment) and re-abutment will be triggered just after un-abutment.
`nil`	If the SKILL function is called during event 3 (un-abutment) but re-abutment will not be triggered just after un-abutment. Or, the SKILL function is not called during event 3.

Example

```
groupId = Database Id of the group going through un-abutment (event 3)
dbAbutIsReAbutGroup(groupId)
```


© 2020 All Rights Reserved.

dbSetFigAbutName

```
dbSetFigAbutName(
     d_figId
     t_abutName
     )
     => t / nil
```

Description

Assigns a unique abutment name to the figure associated with the given ID (*d_figId*). If you set the name to an empty string, Virtuoso clears any existing abutment names specified for the figure ID.

The following conditions must be met to successfully assign a unique abutment name to the figure:

- The specified ID is that of a figure.
- The figure is in a pcell.
- The abutment name being assigned is unique within a pcell submaster.
- The abutment name and abutment reference name assigned to a figure must be different.

Arguments

d_figId	Specifies the ID associated with the figure to which you want to assign an abutment name
t_abutName	Specifies the abutment name to be assigned to the figure ID

Value Returned

t	Returns `t` if the abutment name is successfully assigned
`nil`	Returns `nil` if the abutment name is not assigned

Example

```
dbSetFigAbutName(fig "dpolyRight")
```


© 2020 All Rights Reserved.

dbSetFigAbutRefName

```
dbSetFigAbutRefName(
     d_figId
     t_abutRefName
     )
     => t / nil
```

Description

Assigns an abutment reference name to the specified figure associated with the given ID (*d_figId*). If you set the name to an empty string, Virtuoso clears any existing abutment reference names specified for the figure ID.

The following conditions must be met to successfully assign an abutment reference name to the figure:

- The specified ID is that of a figure.
- The figure exists in a pcell.
- The abutment reference name and name assigned to the figure must be different.

Arguments

d_figId	Specifies the ID of the figure to which you want to assign an abutment reference name
t_abutRefName	Specifies the abutment reference name to be assigned to the figure ID

Value Returned

`t`	Returns `t` if the abutment reference name is successfully assigned
`nil`	Returns `nil` if the abutment reference name is not assigned

Example

```
dbSetFigAbutRefName(fig "diffpolyRight")
```


© 2020 All Rights Reserved.

dbGetFigAbutName

```
dbGetFigAbutName(
     d_figId

     => t_abutName / nil
```

Description

Retrieves the abutment name assigned to the figure associated with the given ID (*d_figId*).

The following conditions must be met to successfully retrieve the abutment name assigned to a figure:

- The specified ID is that of a figure.
- An abutment name must have been assigned to the figure associated with the given ID.
- The figure exists in a Pcell.

Arguments

d_figId	Specifies the figure ID for which you want to retrieve the abutment name

Value Returned

t_abutName	Returns the abutment name assigned to the figure if it is successfully retrieved
`nil`	Returns `nil` if an abutment name is not assigned to the given figure ID or if the above conditions are not met

Example

```
dbGetFigAbutName(fig)=> "dpolyRight”
```


© 2020 All Rights Reserved.

dbGetFigAbutRefName

```
dbGetFigAbutRefName(
     d_figId
     )
     => t_abutname / nil
```

Description

Retrieves the abutment reference name assigned to the figure associated with the given ID (*d_figId*).

The following conditions must be met to successfully retrieve the abutment reference name assigned to a figure:

- The specified ID is that of a figure.
- An abutment reference name must have been assigned to the figure associated with the given ID.
- The figure exists in a Pcell.

Arguments

d_figId	Specifies the ID of the figure for which you want to retrieve the abutment reference name

Value Returned

t_abutName	Returns the abutment reference name assigned to the figure if it is successfully retrieved
`nil`	Returns `nil` if an abutment reference name is not assigned to the given figure ID or if the conditions specified above are not met

Example

```
dbGetFigAbutRefName(fig)=> "diffpolyRight"
```


© 2020 All Rights Reserved.

dbHasFigAbutName

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

Description

Returns a Boolean value indicating whether an abutment name was assigned to the figure associated with the given ID (*d_figId*)

The following conditions must be met for this function to return `t`:

- The specified ID is that of a figure.
- The figure exists in a Pcell.

Arguments

d_figId	Specifies the ID of the figure for which you want to confirm if an abutment name was set

Value Returned

t	Returns `t` if an abutment name was set for the figure with the specified ID
`nil`	Returns `nil` if no abutment name was assigned to the specified figure ID or if the conditions specified above are not met

Example

```
dbHasFigAbutName(fig) => t
```


© 2020 All Rights Reserved.

dbHasFigAbutRefName

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

Description

Returns a Boolean value indicating whether an abutment reference name was assigned to the figure associated with the given ID (*d_figId*)

The following conditions must be met for this function to return `t`:

- The specified ID is that of a figure.
- The figure exists in a Pcell.

Arguments

d_figId	Specifies the ID of the figure for which you want to confirm if an abutment reference name was set

Value Returned

`t`	Returns `t` if an abutment reference name was set for the figure with the specified ID
`nil`	Returns `nil` if no abutment reference name was assigned to the specified figure or if the figure did not exist in the Pcell

Example

```
dbHasFigAbutRefName(fig) => t
```


© 2020 All Rights Reserved.

dbFindAbutFig

```
dbFindAbutFig(
     d_cvId
     t_abutName
     )
     => d_figId / nil
```

Description

Searches for the figure with the specified abutment name in the given subMaster cellview and returns the ID of the searched figure.

The following conditions must be met for this function to successfully return the ID of the searched figure:

- The specified abutment name is that of a figure.
- The figure with the specified abutment name exists in the subMaster cellview.

Arguments

d_cvId	Specifies the ID of the submaster cell in which you want to search for the figure with the specified abutment name
t_abutName	Specifies the abutment name of the figure that you want to search

Value Returned

d_figId	Returns the ID of the figure you are searching for
`nil`	Returns `nil` if no figure with the specified abutment name exists in the subMaster cellview or if the specified abutment name is not that of a figure

Example

```
dbFindAbutFig(cv "dpolyRight") => fig
```


© 2020 All Rights Reserved.

dbUnabutGroup

```
dbUnabutGroup(
     d_groupID
     )
     => t / nil
```

Description

Unabuts members of an abutment group.

Arguments

d_groupID	Database ID of the group going through unabutment.

Value Returned

`t`	Members of the specified abutment group are unabutted.
`nil`	The operation fails.

Example

For an abutted layout instance `inst`, this function can be used in the following way to unabut all of its associated abutment groups:

```
foreach(_abutGroup _inst~>groupMembers~>group
       dbUnabutGroup(_abutGroup)
)
```


© 2020 All Rights Reserved.

Shape Creation and Retrieval Functions

The following database functions create new shapes in the database.

dbCopyShape

```
dbCopyShape(
     d_shapeId
     d_cellviewId
     [ l_transform ]
     )
     => d_newShapeId / nil
```

Description

Copies a shape to the specified cellview with the given transformation.

Note: (ICADVM20.1 Only) By default, coloring information is not copied. To copy coloring information, set the `cdba copyMPAttributes` environment variable to `t` before using `dbCopyShape`.

Arguments

d_shapeId	The database ID of the shape to be copied.
d_cellviewId	The database ID of the cellview.
l_transform	Optional list value to transform the copied shape. List includes: origin offset, rotation, and magnitude. Default is `'(0:0 "R0" 1)`

Value Returned

d_newShapeId	Returns the database ID of the new shape.
`nil`	An error occurred during execution.

Example

```
dbCopyShape(shapeId cvId list(10:10 "R0" 1)
```


© 2020 All Rights Reserved.

dbCreateLine

```
dbCreateLine(
     d_cellView
     txl_layerPurpose
     l_points
     )
     => d_line / nil
```

Description

Creates a line.

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
l_points	Points list, must contain at least two points.
	For more information, see Type l_point.

Value Returned

d_line	The *dbObject* of the line.
nil	The line is not created.

Examples

```
dbCreateLine(cv list( 68 "drawing") list(0:0 1:5) )
dbCreateLine(cv 1 list(1:1 2:5))
dbCreateLine(cv list("wire" "drawing") list(-4:1 2:2))
```


© 2020 All Rights Reserved.

dbCreatePath

```
dbCreatePath(
     d_cellView
     txl_layerPurpose
     l_points
     n_width
     [ t_pathStyle ]
     )
     => d_path / nil
```

Description

Creates a path.

Note: `dbCreatePath` is a core database SKILL function that checks only for even multiples of database units to create a path. As a result, the created path can be off grid. To create a path with required constraints and additional level of automation use an end-application SKILL function, such as `leCreatePath`.

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
t_purpose	Purpose, if not specified, defaults to *drawing*.
l_points	Point list; must contain at least two points.
	For more information, see Type l_point.
n_width	Path width specified in user units.
t_pathStyle	Path style, defaults to `truncateExtend`.
	For more information, see Type t_pathStyle.

© 2020 All Rights Reserved.

Value Returned

d_path	The *dbObject* of the path.
nil	The path is not created.

Examples

```
dbCreatePath(cv 2 list(0:0 0:1 1:1 1:0) 4 "extendExtend")
dbCreatePath(cv list("background" "drawing") list(1:6 1:1 2:1) 2 )
dbCreatePath(cv list ("background") list(0:0 1:0 1:2 2:3) 2 "squareFlush")
```

Note: if you specify a list of points where more than two points have the same X coordinate or the same Y coordinate, a message is generated

Note: If you specify a list of points where more than two points have the same X coordinate or the same Y coordinate, a message is generated in the CIW. For example, the following code generates a warning message in the CIW:

```
dbCreatePath(cv list("metal3" "drawing") '((2 2) (2 4) (2 6)) 2)
```


© 2020 All Rights Reserved.

dbCreatePolygon

```
dbCreatePolygon(
     d_cellView
     txl_layerPurpose
     l_points
     )
     => d_polygon / nil
```

Description

Creates a polygon.

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair. For more information, see Type txl_layerPurpose.
l_points	The point list must contain at least three points and should not be self-intersecting. The point list does not have to explicitly close because the database assumes that there is always an implicit boundary segment between the first and last points. For more information, see Type l_point.

Value Returned

d_polygon	The *dbObject* of the polygon.
nil	The polygon is not created.

Examples

```
dbCreatePolygon(cv "background" list(0:0 0:1 1:1 1:0))
dbCreatePolygon(cv list(11 "drawing") list(5:10 0:10 0:0))
```


© 2020 All Rights Reserved.

dbCreateRect

```
dbCreateRect(
     d_cellView
     txl_layerPurpose
     l_bBox
     )
     => d_rect / nil
```

Description

Creates a rectangle.

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair. For more information, see Type txl_layerPurpose.
l_bBox	Defines the lower left and upper right corners of the bounding box.

Value Returned

d_rect	The *dbObject* of the rectangle.
nil	The rectangle is not created.

Example

```
dbCreateRect(cv list(87 "pin") list(0:0 6.3:8.7))
```


© 2020 All Rights Reserved.

dbCreateEllipse

```
dbCreateEllipse(
     d_cellView
     txl_layerPurpose
     l_bBox
     )
     => d_ellipse / nil
```

Description

Creates an ellipse.

Arguments

d_cellView	Name of the cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
l_bBox	Defines the bounding box of the ellipse.

Value Returned

d_ellipse	The *dbObject* of the ellipse.
nil	The ellipse is not created.

Example

```
dbCreateEllipse(cv 1 list(0:0 2:3))
```


© 2020 All Rights Reserved.

dbCreateArc

```
dbCreateArc(
     d_cellView
     txl_layerPurpose
     l_ellipse_bBox
     l_arc_bBox
     )
     => d_arc / nil
```

Description

Creates an arc.

The arc is determined by calculating the overlap or the interception between the ellipse and arc bBox given by the caller. In order to correctly determine the intersection of the given bBoxes, overlapping edges between the bBoxes are required.

Two overlapping edges

One overlapping edge

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
l_ellipse_bBox	Bounding box of the ellipse containing the arc.
l_arc_bBox	Bounding box of the arc.

© 2020 All Rights Reserved.

Value Returned

d_arc	The *dbObject* of the arc.
nil	The arc is not created.

Example

```
dbCreateArc(cv list("background" "drawing") list(0:0 2:2) list(1:1 2:2))
```


© 2020 All Rights Reserved.

dbCreateDonut

```
dbCreateDonut(
     d_cellView
     txl_layerPurpose
     l_point
     x_outR
     x_holeR
     )
     => d_donut / nil
```

Description

Creates a donut (two concentric circles).

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
l_point	Defines the center of the donut.
	For more information, see Type l_point.
x_outR	Outer circle radius, must be larger than *x_holeR*.
x_holeR	Inner circle radius.

Value Returned

d_donut	The *dbObject* of the donut.
nil	The donut is not created.

Example

```
dbCreateDonut(cv 1 0:0 100 50)C
```


© 2020 All Rights Reserved.

dbCreateDot

```
dbCreateDot(
     d_cellView
     txl_layerPurpose
     l_point
     )
     => d_dot / nil
```

Description

Creates a dot.

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
l_point	Defines the location of the dot.
	For more information, see Type l_point.

Value Returned

d_dot	The *dbObject* of the dot.
nil	The dot is not created.

Example

```
dbCreateDot(cv "metal1" 1:1)
```


© 2020 All Rights Reserved.

dbCreateLabel

```
dbCreateLabel(
     d_cellView
     txl_layerPurpose
     l_point
     t_label
     t_just
     t_orient
     t_font
     x_height
     )
     => d_label / nil
```

Description

Creates a graphic text-string label.

Arguments

d_cellView	The cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
l_point	Defines the origin of the label.
	For more information, see Type l_point.
t_label	Text string for the label.
t_just	Justification string for the label.
t_orient	Orientation string for the label.
t_font	Font style string for the label.
x_height	Height of the label, must be given as a positive number.

© 2020 All Rights Reserved.

Value Returned

d_label	The *dbObject* of the label.
nil	The label is not created.

Example

```
dbCreateLabel(cv 4 1:1 "myLabel" "centerLeft" "R0" "roman" 2)
```

Creates a label in a particular cellview `cv` with specified layer `4`, purpose `drawing`, origin `1:1`, text `myLabel`, justification `centerLeft`, orientation `R0`, font `roman`, and height `2`.

© 2020 All Rights Reserved.

dbCreateTextDisplay

```
dbCreateTextDisplay(
     d_associateId
     d_ownerId
     txl_layerPurpose
     l_displayFlags
     l_point
     t_just
     t_orient
     t_font
     x_height
     [ g_isDrafted [ g_isOverbar [ g_isVisible [ g_isNameVisible [
     g_isValueVisible [ t_attrOrParamName [ g_isParamAssoc ] ] ] ] ] ] ]
     )
     => d_textDisplay / nil
```

Description

Creates a text display object.

TextDisplays are associated with design objects so that graphical applications can display text strings representing attributes and properties of the associated object. The owner and associate of a text display are usually the same but they can be different. For example, a text display may be “owned” by an instance, but may be “associated” with a label in the master of that instance.

TextDisplays cannot be “associated” with objects in other cellviews unless the TextDisplay is “owned” by an instance or Mosaic of that cellview.

© 2020 All Rights Reserved.

Arguments

d_associateId

Associate of the text display. This can be a label in a master cellview, a property, or a *dbObject* whose attribute *t_attrName* is to be displayed. However, not all attributes of objects can be displayed with a textDisplay. Like labels, properties and attributes of master cellviews and terminals can be the associates of text displays that you create in the cellview where that master is instantiated. Such text display objects are called cross-cellview text displays. Some of the attributes that can be displayed using text display include:

- Terminals: `"name"`, `"hasPins"`, `"numBits"`
- Nets: `"name"`, `"sigType"`,`"isGlobal"`, `"isEmpty"`, `"numBits"`
- InstTerm: `"name"`
- Inst/Mosaic: `"name"`,`"libName"`, `"cellName"`, `"viewName"`, `"numBits"`, `"width"`, `"isBound"`
- Cellview: `"libName"`, `"cellName"`, `"viewName"`, `"cellType"`, `"lastSavedTime"`
- Labels: The values of labels can be displayed with a text display.

d_ownerId

Owner of the text display. This can be any database object except properties or groups.

txl_layerPurpose

Name, number, or a list of layer or layer-purpose pair.

For more information, see Type txl_layerPurpose.

l_displayFlags

Display characteristics of the label associate that are to be overridden by the text display. For properties and for attribute associates, every display characteristic must be specified; therefore, `t` must be specified here, followed by a specification for each of the display characteristics listed below.

l_point

Origin of the text display. This must be specified for properties and for object attribute associates.

For more information, see Type l_point.

© 2020 All Rights Reserved.

t_just	Justification string for the text display. This must be specified for properties and for object attribute associates. For more information, see Type t_just.
t_orient	Orientation string for the text display. This must be specified for properties and for object attribute associates. For more information, see Type t_orient.
t_font	Font style string for the text display. This must be specified for properties and for object attribute associates. For more information, see Type t_font.
x_height	Height of the text display that must be given as a positive number. This must be specified for properties and for object attribute associates.
g_isDrafted	Specifies whether or not drafting is set for the text display. The default is `nil`.
g_isOverbar	Specifies whether or not over-bar is set for the text display. The default is `nil`.
g_isVisible	Specifies whether or not the text display is visible. The default is `t`. When *g_isVisible* is set to `t`, `g_isNameVisible` and `g_isValueVisible` cannot be set to `nil`. For example, if a user tries to set `(g_isVisible g_isNameVisible g_isValueVisible)` to `(t nil nil)`, a warning is issued, and `g_isValueVisible` is set to `t`.
g_isNameVisible	Specifies whether or not the property or attribute name is visible. The default is `t`. This argument applies only to a text display with a property or attribute as its associate.
g_isValueVisible	Specifies whether or not the property or attribute value is visible. The default is `t`. This argument applies only to a text display with a property or attribute as its associate.
t_attrOrParamName	Name of the attribute or the CDF parameter of the associate object specified in *d_associateId*.
g_isParamAssoc	Specifies whether or not *t_attrOrParamName* is a CDF parameter name. The default is `nil`.

© 2020 All Rights Reserved.

Value Returned

d_textDisplay	The *dbObject* of the text display.
nil	The text display is not created.

Examples

```
inst = dbFindAnyInstByName(cv "I0")
tdMaster = dbOpenCellViewByType("tranTst" "iopin" "symbolr")
assoc = car(setof(i tdMaster~>shapes i~>objType == "label" && i~>theLabel
=="[@terminalName]"))
obj = dbCreateTextDisplay(assoc inst 229:237 t -10:0 "lowerLeft" "R0" "stick" 1 t
nil t t t "mylabel" nil)
obj~>assocType
"labelAssocType"
obj~>text
"[@terminalName]"
obj~>xy
(-11.4375 0.375)
```


© 2020 All Rights Reserved.

In the following example, the owner is an instance and the associate is a label:

```
inst = dbFindAnyInstByName(cv "I0")
tdMaster = dbOpenCellViewByType("tranTst" "iopin" "symbolr")
assoc = car(setof(i tdMaster~>shapes i~>objType == "label" && i~>theLabel
=="[@terminalName]"))
obj = dbCreateTextDisplay(assoc inst 229:237 t -10:0 "lowerLeft" "R0" "stick" 1 t
nil t t t "mylabel" nil)
obj~>assocType
"labelAssocType"
obj~>text
"[@terminalName]"
obj~>xy
(-11.4375 0.375)
```


© 2020 All Rights Reserved.

dbHasShape

```
dbHasShape(
     d_cellviewId
     )
     => t / nil
```

Description

Checks whether the specified cellview contains any shapes.

Arguments

d_cellviewId	Database ID of the cellview.

Value Returned

t	The cellview contains shapes.
nil	The cellview does not contain shapes.

© 2020 All Rights Reserved.

dbMoveShape

```
dbMoveShape(
     shapeId
     cellviewId
     [l_transform]
     )
     => d_shapeId / nil
```

Description

Moves a shape to given cellview with given transformation.

Arguments

d_shapeId	The database ID of the shape to be moved.
d_cellviewId	The database ID of the cellview.
l_transform	Optional list value to transform the copied shape. List includes: origin offset, rotation and magnitude. Default is `'(0:0 "R0" 1)`

Value Returned

d_shapeId	The database ID of the moved shape.
`nil`	An error occurred during execution.

Example

```
dbMoveShape(shapeId cvId list(10:10 "R0" 1)
```


© 2020 All Rights Reserved.

dbSetTextDisplayNameValueVisible

```
dbSetTextDisplayNameValueVisible(
     d_tdId
     g_isNameVisible
     g_isValueVisible
     )
     => t / nil
```

Description

Sets the `isNameVisible` and the `isValueVisible` flags. The `isNameVisible` flag sets the text display name visibility. When this flag is set to `true`, the name of the text display will be visible. The `isValueVisible` flag sets the text display value visibility. When this flag is set to `true`, the value of the text display will be visible.

Arguments

d_tdId	The `textDisplay` ID.
g_isNameVisible	The `isNameVisible` flag, sets name visibility.
g_isValueVisible	The `isValueVisible` flag, sets value visibility.

Value Returned

`t`	The flags are set successfully.
`nil`	An error occurred during execution.

© 2020 All Rights Reserved.

Shape Conversion Functions

The following functions convert shapes from one form to another.

© 2020 All Rights Reserved.

dbConvertArcToLine

```
dbConvertArcToLine(
     d_arc
     x_nSides
     )
     => d_line / nil
```

Description

Converts an arc to a line. A line is defined as a number of connecting segments.

Arguments

d_arc	The arc to be converted.
x_nSides	Number of segments to be created.

Value Returned

d_line	The *dbObject* of the line if the conversion is successful.
nil	The arc is not converted.

Example

```
dbConvertArcToLine(arcId 5)
```


© 2020 All Rights Reserved.

dbConvertDonutToPolygon

```
dbConvertDonutToPolygon(
     d_donut
     x_nSides
     )
     => d_polygon / nil
```

Description

Converts a donut to a polygon.

Both the inner and outer circles describing the donut are converted into a single polygon. A single line segment connects to two sub-polygons corresponding to the concentric circles, forming one polygon.

Arguments

d_donut	The donut to be converted.
x_nSides	Number of sides created for each circle.

Value Returned

d_polygon	The *dbObject* of the polygon if the conversion is successful.
nil	The donut is not converted.

Example

```
dbConvertDonutToPolygon(donutId 7)
```


© 2020 All Rights Reserved.

dbConvertEllipseToPolygon

```
dbConvertEllipseToPolygon(
     d_ellipse
     x_nSides
     )
     => d_polygon / nil
```

Description

Converts an ellipse to a polygon.

Arguments

d_ellipse	The ellipse to be converted.
x_nSides	Number of sides defining the polygon.

Value Returned

d_polygon	The *dbObject* of the polygon if the conversion is successful.
nil	The ellipse is not converted.

Example

```
dbConvertEllipseToPolygon(ellipseId 7)
```


© 2020 All Rights Reserved.

dbConvertPathToPolygon

```
dbConvertPathToPolygon(
     d_path
     )
     => d_polygon / nil
```

Description

Converts a path to a polygon with the same number of points as the path.

Arguments

d_path	The path to be converted.

Value Returned

d_polygon	The *dbObject* of the polygon if the conversion is successful.
nil	The path is not converted.

Example

```
dbConvertPathToPolygon(pathId)
```


© 2020 All Rights Reserved.

dbConvertPathSegToPolygon

```
dbConvertPathSegToPolygon(
     d_pathSeg
     )
     => d_polygon / nil
```

Description

Converts a pathseg to a polygon based on the boundary list.

Arguments

d_pathSeg	The pathseg to be converted.

Value Returned

d_polygon	The *dbObject* of the polygon if the conversion is successful.
nil	The pathseg is not converted.

Example

```
dbConvertPathSegToPolygon(pathSegId)
```


© 2020 All Rights Reserved.

dbConvertLineToPath

```
dbConvertLineToPath(
     d_line
     x_width
     [ t_pathStyle ]
     )
     => d_path / nil
```

Description

Converts a line to a path.

Arguments

d_line	The line to be converted.
x_width	Path width of the path specified in user units.
t_pathStyle	Path style of the path, defaults to `"truncateExtend"`.

Value Returned

d_path	The *dbObject* of the path if the conversion is successful.
`nil`	The line is not converted.

Example

```
dbConvertLineToPath(lineId 4)
```


© 2020 All Rights Reserved.

dbConvertRectToPolygon

```
dbConvertRectToPolygon(
     d_rect
     )
     => d_polygon / nil
```

Description

Converts a rectangle into a polygon.

Arguments

d_rect	The rectangle object that needs to be converted to a polygon object.

Value Returned

d_polygon	The converted polygon object.
nil	The rectangle to polygon object conversion failed.

Example

```
dbConvertRectToPolygon(d_rect)
```


© 2020 All Rights Reserved.

dbCompressPointArray

```
dbCompressPointArray(
     l_pointArray
     x_isClosed
     [ f_DBUPerUU ]
     )
     => l_newPointArray
```

Description

Compresses collinear points in a list of points.

Arguments

l_pointArray	Point array to be compressed.
x_isClosed	Either `t` or `nil` specifying that the figure is closed or not.
f_DBUPerUU	Number of database units in one user unit. The default is `1.0`.

Value Returned

l_newPointArray	Compressed point array.

Example

```
l1 = list((0:0) (0:1) (0:2) (0:3))
l2 = dbCompressPointArray(l1 nil)
=> ((0:0) (0:3))
```


© 2020 All Rights Reserved.

Mosaic Creation and Retrieval Functions

The following functions creates a simple mosaic in the database.

© 2020 All Rights Reserved.

dbCreateParamSimpleMosaic

```
dbCreateParamSimpleMosaic(
     d_cellView
     d_masterCellView
     t_name
     l_origin
     l_orient
     x_rows
     x_cols
     n_rowSpacing
     n_colSpacing
     l_params

     )
     => d_mosaic / nil
```

Description

Creates a parameterized simple mosaic (from the master *d_masterCellView)* in *d_cellView,* with an origin and orientation specified by *l_origin* and *l_orient*.

Arguments

d_cellView	The *dbObject* ID of the cellview.
d_masterCellView	The *dbObject* ID of the master cellview.
t_name	If specified, the mosaic is named *t_name*; if nil, the program assigns a unique name to the mosaic.
l_origin	Origin of the lower left instance in the array.
l_orient	Orientation of the instance.
x_rows	Number of rows defining the size of the array.
x_cols	Number of columns defining the size of the array.
n_rowSpacing	Spacing between the origins of the mosaic elements in y-direction.
n_colSpacing	Spacing between the origins of the mosaic elements in x-direction.
l_params	List of *l_param*, the names, types, and values of parameters of the mosaic.

© 2020 All Rights Reserved.

Value Returned

d_mosaic	The *dbObject* of the mosaic.
nil	The mosaic is not created.

Example

```
dbCreateParamSimpleMosaic(cv masterTile "myMos" 0:0 "R0" 5 7 3 2 list(list("l"
"float" 5.0)))
```


© 2020 All Rights Reserved.

dbCreateParamSimpleMosaicByMasterName

```
dbCreateParamSimpleMosaicByMasterName(
     d_cellView
     t_libName
     t_cellName
     t_viewName
     t_name
     l_origin
     l_orient
     x_rows
     x_cols
     n_rowSpacing
     n_colSpacing
     l_params

     )
     => d_mosaic / nil
```

Description

Creates a parameterized simple mosaic (from the master *t_libName/t_cellName/t_viewName)* in *d_cellView,* with an origin and orientation specified by *l_origin* and *l_orient*.

Arguments

d_cellView	The *dbObject* ID of the cellview.
t_libName	Library name of the master cellview.
t_cellName	Cell name of the master cellview.
t_viewName	View name of the master cellview.
t_Name	Name of the mosaic. If *t_Name* is `nil`, the program assigns a unique name to the mosaic.
l_origin	Origin of the lower left instance in the array.
l_orient	Orientation of the instance.
x_rows	Number of rows defining the size of the array.
x_cols	Number of columns defining the size of the array.
n_rowSpacing	Spacing between the origins of the mosaic elements in y-direction.

© 2020 All Rights Reserved.

n_colSpacing	Spacing between the origins of the mosaic elements in x-direction.
l_params	List of *l_param*, the names, types, and values of parameters of the mosaic.

Value Returned

d_mosaic	The *dbObject* of the mosaic.
nil	The mosaic is not created.

Example

```
dbCreateParamSimpleMosaicByMasterName(cv "masterLib" "masterCell" "masterView"
"myMos" 0:0 "R0" 5 7 3 2 list(list("l" "float" 5.0)))
```


© 2020 All Rights Reserved.

dbCreateSimpleMosaic

```
dbCreateSimpleMosaic(
     d_cellView
     d_masterCellView
     t_name
     l_point
     t_orient
     x_rows
     x_cols
     n_rowSpacing
     n_colSpacing
     )
     => d_mosaic / nil
```

Description

Creates a simple mosaic in the cellview from the master *d_masterCellView.*

Arguments

d_cellView	Name of the cellview.
d_masterCellView	Name of the master cellview.
t_name	If specified, the instance is named *t_name*; if `nil`, the program assigns a unique name to the instance.
l_point	Origin of the lower left instance in the array. For more information, see Type l_point.
t_orient	Orientation of the instance.
x_rows	Number of rows defining the size of the array.
x_cols	Number of columns defining the size of the array.
n_rowSpacing	Spacing between the origins of the mosaic elements in y-direction.
n_colSpacing	Spacing between the origins of the mosaic elements in x-direction.

© 2020 All Rights Reserved.

Value Returned

d_mosaic	The *dbObject* of the mosaic.
nil	The mosaic is not created.

Example

```
dbCreateSimpleMosaic(cv masterTile "myMos" 0:0 "R0" 5 7 3 2)
```


© 2020 All Rights Reserved.

dbFindMosaicByName

```
dbFindMosaicByName(
     d_cellViewId
     t_name
     )
     => d_mosaicId / nil
```

Description

Returns a mosaic ID if a mosaic with the given name is found in the cellview specified.

Arguments

d_cellViewId	The database cellview ID.
t_name	The name of the mosaic.

Value Returned

d_mosaicId	A mosaic ID if a mosaic with the given name is found in the cellview specified.
`nil`	Otherwise, `nil` is returned.

© 2020 All Rights Reserved.

Property Creation and Modification Functions

The following functions create and replace properties in the database.

dbCreateProp

```
dbCreateProp(
     g_object
     t_name
     t_propType
     g_value
     )
     => d_prop / nil
```

Description

Creates a scalar property for an object. If a property named *t_name* already exists on the specified object, it returns `nil`.

Arguments

g_object	Specifies an object the property is to be associated with. The object can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.
t_propType	Type of the property.
g_value	Value of the property (must be consistent with *t_propType*).

Value Returned

d_prop	A scalar property belonging to the database object;.
`nil`	A property named *t_name* already exists on the specified object or an error occurs.

Examples

```
dbCreateProp(cv "prop1" "boolean" t)
dbCreateProp(cv "DBUperUU" "float" 1.0)
```


© 2020 All Rights Reserved.

dbCreateRangeProp

```
dbCreateRangeProp(
     g_object
     t_name
     t_propType
     g_value
     g_lBound
     g_uBound
     )
     => d_prop / nil
```

Description

Creates a range property for an object.

Arguments

g_object	The database object which can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.
t_propType	Type of the property.
g_value	Value of the property.
g_lBound	Lower bound of the range of values.
g_uBound	Upper bound of the range of values.

Value Returned

d_prop	A range property belonging to the database object.
`nil`	A property named *t_name* already exists on the specified object, or if the value passed in is not within the specified range. Also returns `nil` if an error occurs, such as an invalid argument, or if *d_object* is not a hierarchical property.

Examples

```
dbCreateRangeProp(cv "prop1" "int" 5 3 8)
dbCreateRangeProp(cv "prop1" "float" 100.0 0 "infinity")
```


© 2020 All Rights Reserved.

dbCreateEnumProp

```
dbCreateEnumProp(
     g_object
     t_name
     t_value
     l_string
     )
     => d_prop / nil
```

Description

Creates an enumerated property for an object.

An enumerated property can have only string values.

Arguments

g_object	The database object which can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.
t_value	Value of the property.
l_string	Array of strings enumerating the possible values of the property, one of which must equal *t_value*.

Value Returned

d_prop	An enumerated property belonging to the database object.
nil	A property named *t_name* already exists on the specified object, or if the initial value is not in the enumerated property range passed in. Also returns nil if an error occurs.

Examples

```
dbCreateEnumProp(cv "prop1" "apple" list("apple" "pear" "peach"))
dbCreateEnumProp(cv "prop1" "float" list("sink" "float" "beach" "pool"))
```


© 2020 All Rights Reserved.

dbCreateHierProp

```
dbCreateHierProp(
     g_object
     t_name
     )
     => d_prop / nil
```

Description

Creates a hierarchical property for an object.

A hierarchical property is the only kind of property that can have other properties associated with it.

Arguments

g_object	The database object which can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.

Value Returned

d_prop	A hierarchical property belonging to the database object.
nil	A property named *t_name* already exists on the specified object or if *d_object* is invalid or is not a hierarchical property.

Example

```
dbCreateHierProp(dbId "HProp")
```


© 2020 All Rights Reserved.

dbFindProp

```
dbFindProp(
     d_object
     t_name
     )
     => d_propId / nil
```

Description

If a property of the given name is found on the object specified, *d_propId* of the property is returned; otherwise, `nil` is returned. This function has the same functionality as `dbGetPropByName`.

Argument

d_object	Can be a *dbId*, *propBagId*, or *ddId*.
t_name	Name of the property to be searched.

Value Returned

d_propId	*PropId* if property of the given name is found on the object
`nil`	Returns `nil` otherwise.

Example

```
propId = dbFindProp(instId "myProp")
```


© 2020 All Rights Reserved.

dbGetPropByName

```
dbGetPropByName(
     d_object
     t_name
     )
     => d_propId / nil
```

Description

If a property of the given name is found on the object specified, *d_propId* of the property is returned; otherwise, `nil` is returned.

Argument

d_object	Can be a *dbId*, *propBagId*, or *ddId*.
t_name	Name of the property to be searched.

Value Returned

d_propId	*PropId* if the property of the given name is found on the object.
`nil`	Returns `nil` otherwise.

Example

```
propId = dbGetPropByName(instId "myProp")
```


© 2020 All Rights Reserved.

dbReplaceProp

```
dbReplaceProp(
     g_object
     t_name
     t_propType
     g_value
     )
     => d_prop / nil
```

Description

Replaces a scalar property for an object. Calls to `dbReplaceProp` for abutment groups must be made before calls to `dbReplaceProp` for pcell properties.

This function is similar to `dbCreateProp`, except that if a property named *t_name* already exists on the specified object, its type and value are replaced by the ones specified in *t_propType* and *g_value*.

Arguments

g_object	The database object which can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property. If *t_name* exists and is a range or enumerated property, it is replaced by a scalar property.
t_propType	Type of the property.
g_value	Value of the property; must be consistent with *t_propType*.

Value Returned

d_prop	A scalar property belonging to the database object.
nil	Any argument is in error, or *d_object* is not a hierarchical property.

Examples

```
dbReplaceProp(dbId "netNumber" "int" 12)
dbReplaceProp(dbId "drawGrid?" "boolean" t)
```


© 2020 All Rights Reserved.

dbReplaceRangeProp

```
dbReplaceRangeProp(
     g_object
     t_name
     t_propType
     g_value
     g_lBound
     g_uBound
     )
     => d_prop / nil
```

Description

Replaces a range property for an object.

This function is similar to `dbCreateRangeProp` except that if a property named *t_name* already exists on the specified object, it is replaced by the one specified as *t_propType*, *g_value*, *g_lBound*, and *g_uBound*.

© 2020 All Rights Reserved.

Arguments

g_object	The database object which can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.
t_propType	Type of the property.
g_value	Value of the property. *g_value*, *g_lBound*, and *g_uBound* must be consistent with *t_propType*.
g_lBound	Lower bound of the range of values.
g_uBound	Upper bound of the range of values.

Value Returned

d_prop	A range property belonging to the database object.
nil	The value is not in the given range or if *d_object* is not a hierarchical property.

Examples

```
dbReplaceRangeProp(dbId "prop" "float" 120.0 0 nil)
dbReplaceRangeProp(dbId "prop" "int" 5 3 12)dbReplaceRangeProp
```


© 2020 All Rights Reserved.

dbReplaceEnumProp

```
dbReplaceEnumProp(
     g_object
     t_name
     t_value
     l_string
     )
     => d_prop / nil
```

Description

Replaces an enumerated property for an object.

This function is similar to `dbCreateEnumProp` except that if a property named *t_name* already exists on the specified object, it is replaced by the one specified as *t_value* and *l_string*. An enumerated property can only have values that are strings.

Arguments

g_object	The database object which can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.
t_value	Value of the property.
l_string	Array of strings enumerating the possible values of the property, one of which must equal *t_value*.

Value Returned

d_prop	An enumerated property belonging to the database object.
`nil`	The value is not in the enumerated property range or if *d_object* is not a hierarchical property.

Example

```
dbReplaceEnumProp(dbId "prop" "grape" list("orange" "grape" "apricot"))
```


© 2020 All Rights Reserved.

dbReplaceHierProp

```
dbReplaceHierProp(
     g_object
     t_name
     )
     => d_prop / nil
```

Description

Replaces a hierarchical property for an object.

This function is similar to `dbCreateHierProp` except that if a property named *t_name* already exists on the specified object, its value is deleted as are the properties associated with it if it is a hierarchical property. A hierarchical property can be defined to have other properties associated with it.

Arguments

g_object	The database object which can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.

Value Returned

d_prop	A hierarchical property belonging to the database object.
`nil`	*d_object* is not valid or *t_name* is not a string.

Example

```
dbReplaceHierProp(inst "prop1")
```


© 2020 All Rights Reserved.

dbCopyProp

```
dbCopyProp(
     g_object1
     g_object2
     )
     => t / nil
```

Description

Copies all the properties of one database object to another database object.

Note: For Pcell super master cellviews, this function will skip the Pcell parameters. Pcell parameters are represented as a hierarchical property called `parameters`. This property is skipped by `dbCopyProp`. To copy the Pcell parameters, use dbCopySingleProp.

Arguments

g_object1	Database object the properties are copied from. The object can be a *dbId*,*ddId*, or *dbBagId*.
g_object2	Database object the properties of the first database object are copied to. The object can be a *dbId*, *ddId*, or *dbBagId*.

Value Returned

`t`	The copy is successful.
`nil`	The properties are not copied or if they are only partially copied.

Example

```
dbCopyProp(dbId1 dbId2)
```


© 2020 All Rights Reserved.

dbCopyPropList

```
dbCopyPropList(
     d_dbId1
     d_dbId2
     )
     => t / nil
```

Description

Copies all the properties from the database object given in the first argument to the database object given in the second argument in a single, atomic, copy operation. Properties with the same name in the destination object are overwritten. If the given database objects are pcell instances, then pcell evaluation happens once after all the properties are copied.

Arguments

d_dbId1	Database object the properties are copied from. The object can be a *dbId*,*ddId*, or *dbBagId*.
d_dbId2	Database object the properties of the first database object are copied to. The object can be a *dbId*, *ddId*, or *dbBagId*.

Value Returned

`t`	The copy is successful.
`nil`	The properties are not copied or if they are only partially copied.

Example

```
dbCopyPropList(i1 i2)
```


© 2020 All Rights Reserved.

dbResetPropList

```
dbResetPropList(
     d_object
     l_props
     )
     => t / nil
```

Description

Resets a list of properties on a given object with the specified list of properties.

The function works in the following ways in different scenarios:

- The values of already existing properties specified in the property list are updated with the new values.
- New properties in the specified property list are created on the object.
- The properties that already exist on the object, but are not specified in the property list, are deleted.
- If the specified property list is empty, all the existing properties on the object are deleted.

Arguments

d_object Database ID of the object whose properties are being replaced.

l_props `List(propName propType propValue)`

List of properties.

where,

- `propName`
  Name of the property
- `propType`
  Type of property
- `propValue`
  Value of the property.

© 2020 All Rights Reserved.

Value Returned

`t`	List of properties on a given object are reset with the specified list of properties.
`nil`	Returned if an error occurs.

Example

```
lib = ddCreateLib("lib")
; Create Pcell Master
pcellM = dbOpenCellViewByType("lib" "pcellM" "layout" "maskLayout" "w")
; Create top cellview
top = dbOpenCellViewByType("lib" "top" "layout" "maskLayout" "w")
; Define pcell function
pcDefinePCell(
    (list (ddGetObj "lib") "pcellM" "layout")
        ((w float 1)
        (l float 1)
        (h float 1)
        (b float 1)
        (k float 1))
        warn("INFO 1\n")
)
; Create instance I1 with 3 parameters
i1 = dbCreateParamInst(top pcellM "I1" list(0 0) "R0" 1 '(("l" "float" 2) ("h"
"float" 3) ("b" "float" 4)))
; Create properties on I1 that are not Pcell parameters
dbCreateProp(i1 "prop1" "int" 1)
dbCreateProp(i1 "prop2" "string" "prop2")
; Reset the properties
dbResetPropList(i1 '(("prop2" "string" "newProp2") ("prop3" "int" 10) ("l" "float"
4)))
;Examine the results
mapcar(lambda((p) printf("%s %s %L\n" p~>name p~>valueType p~>value)) i1~>prop)
; The result will be:
; prop3 int 10
; prop2 string "newProp2"
; l float 4.0
; (t t t)
```


© 2020 All Rights Reserved.

dbDeletePropByName

```
dbDeletePropByName(
     g_object
     t_name
     )
     => t / nil
```

Description

Deletes a property from an object.

Arguments

g_object	The database object can be a *dbId*, *ddId*, or *dbBagId*.
t_name	Name of the property.

Value Returned

t	The property is not found or if the property is found and deleted.
nil	The property is not deleted.

Note: This function might return t even if an error occurs.

Example

```
dbDeletePropByName(dbId1 "myProp")
```


© 2020 All Rights Reserved.

dbCopySingleProp

```
dbCopySingleProp(
     d_srcPropId
     g_dstObjId
     )
     => d_dstPropId
```

Description

Copies a single property to a destination object.

If the property already exists on the destination object, it is overwritten with the copied property. This function differs from the `dbCopyProp` function, which copies *all* properties from the source object.

Arguments

d_srcPropId	ID of the source property to be copied.
g_dstObjId	Destination object where the property is to be copied. This destination object can be a property, *dbId*, *ddId*, or *dbBagId*.

Value Returned

d_dstPropId	The property ID of the destination property.

Example

This example copies the property `prop` from cellview `cv1` to cellview `cv2`.

```
cv1 = dbOpenCellViewByType("test" "abc" "layout" "maskLayout" "a")
cv2 = dbOpenCellViewByType("test" "xyz" "layout" "" "a")
prop = dbCreateProp(cv1 "width" "int" 10)
dstProp = dbCopySingleProp(prop cv2)
```


© 2020 All Rights Reserved.

dbMergeSingleProp

```
dbMergeSingleProp(
     d_srcPropId
     g_dstObjId
     )
     => d_dstPropId
```

Description

Merges a property with a destination object without overwriting the property if it already exists.

Arguments

d_srcPropId	Source property to be merged.
g_dstObjId	Destination object where the property is to be merged. The destination object can be a property, *dbId*, *ddId*, or *dbBagId*.

Value Returned

d_dstPropId	The property ID of the merged property.

Example

A property on cellview `cv1` is created and copied to cellview `cv2`.

```
cv1 = dbOpenCellViewByType("test" "abc" "layout" "maskLayout" "a")
cv2 = dbOpenCellViewByType("test" "xyz" "layout" "" "a")
prop = dbCreateProp(cv1 "width" "int" 10)
dstProp = dbCopySingleProp(prop cv2)
dstProp1 = dbMergeSingleProp(prop cv2)
```

The same property is merged from `cv1` to `cv2`. The return values, `dstProp` and `dstProp1`, have the same property ID because the merge process does not overwrite a property that already exists.

© 2020 All Rights Reserved.

dbReplacePropList

```
dbReplacePropList(
     g_object
     l_params
     )
     => t / nil
```

Description

Replaces the specified list of properties (*l_params*) in the given object. `dbReplacePropList` will trigger Pcell evaluation once as an atomic operation.

Arguments

g_object	Database object or property bag for which the given list of properties will be replaced.
l_params	List of properties. This list is a *list of sub-lists*. Each sub-list should contain three elements: a property name, a property type (acceptable to `dbCreateProp`) and a value corresponding to the property type. The first two of these elements must be strings.

Value Returned

`t`	The properties are replaced successfully.
`nil`	The function fails.

Example

```
dbReplacePropList(cv list( list("prop1" "int" 100)
list("prop2" "string" "hello") ) )
dbReplacePropList(cv '(("enable" "boolean" t) ("a" "int" 3) ("b" float" 4.3u)))
dbReplacePropList(cv '(("enable" "boolean" t)))
dbReplacePropList(cv '(("enable" "boolean" "TRUE")))
dbReplacePropList(cv '(("count" "int" 3)))
dbReplacePropList(cv '(("pdkinfo" "ILList" ((1 2 3) (4 5 6) (("hello" "world"))))))
```


© 2020 All Rights Reserved.

dbSearchPropByName

```
dbSearchPropByName(
     d_object
     t_name
     )
     => d_propId / nil
```

Description

Searches for the specified property.

The function uses the following search rules:

- If the given ID is an instance, the function looks for the specified property on the instance first, then on the instance's parent cellview bag, then on the cell bag and/or the library bag.
- If the given ID is a cellview, the function looks for the specified property on the cellview first, then on the cellview bag, then on the cell bag and/or the library bag.
- If the given ID is a *ddId*, the function searches its bag first, then the bag at the next-higher level, up to and including the library-level bag.
- If the given ID is a *bagId*, the function searches the given bag first, then the bag at the next-higher level, up to and including the library bag.

Note: For property bag-related functions, see the "Property Bag Functions" sections of this chapter.

Arguments

d_object	Database object for which the given list of properties will be replaced.
t_name	List of properties.

Value Returned

d_propId	The properties are replaced successfully.
nil	The properties are not replaced successfully.

© 2020 All Rights Reserved.

Object Deletion Functions

The following is the database function for deleting different types of database objects.

dbDeleteObject

```
dbDeleteObject(
     d_object
     )
     => t / nil
```

Description

Deletes one of the following objects: shapes, nets, terminals, instance terminals, pins, instance pins, mosaics, instances, mosaic instances, properties, groups, and group members. Any other object type produces an error.

If the object is a mosaic, the associated mosaic instance list and personality array are also deleted. If the object is a net, the net's terminal is automatically deleted and the instance terminals are disconnected from the net. If the object is a terminal, all of the associated pins are automatically deleted and corresponding instance terminals and instance pins are unbound. If the object is a pin, the instance pins are unbound and the figure is detached from the pin but not deleted.

Other functions that delete database objects, documented elsewhere in this chapter, are `dbDeletePropByName`, `dbDeleteObjectFromGroup` and `dbDeleteAllNet`.

Arguments

d_object	The object to be deleted; can be shapes, nets, terminals, instance terminals, pins, instance pins, mosaics, instances, mosaic instances, properties, groups, and group members. Any other object type produces an error.

Value Returned

`t`	The object is deleted.
`nil`	The object is not deleted.

© 2020 All Rights Reserved.

Example

```
dbDeleteObject(net)
```


© 2020 All Rights Reserved.

Group Creation and Modification Functions

The following functions create and replace groups in the database. A group collects any number of different objects in a cellview, thus allowing user-defined relationships to exist between database objects. A group contains *groupMember* objects, which in turn refer to the database objects kept as part of the group.

dbAddObjectToGroup

```
dbAddObjectToGroup(
     d_group
     d_object
     )
     => d_groupMem / nil
```

Description

Adds a database object to a group.

For compatibility, the parameters are also accepted in the opposite order, unless `d_object` is itself a group.

Arguments

`d_group`	Database object of the group.
`d_object`	Database object to be added to the group.

Value Returned

`d_groupMem`	The `dbObject` of the group member if the database is successfully added to the group.
`nil`	The object is not added.

Example

```
dbAddObjectToGroup(groupId objId)
```


© 2020 All Rights Reserved.

dbChangeGroupType

```
dbChangeGroupType(
     d_groupId
     l_groupType
     )
     => t / nil
```

Description

Changes the type of a group.

Arguments

d_groupId	ID of the group to be changed.
l_groupType	List of group attributes. Valid values: `collection` or `set`—to define the attribute `groupType`. `ordered` or `unordered`—to define the attribute `ordered`. `nonUniqueName` or `uniqueName`—to define the attribute `uniqueName`. `deleteNever`, `deleteFirst`, or `deleteLast`–to define the attribute `deleteWhen`. For more information on valid values of `l_groupType` see Type l_group.

Value Returned

`t`	The group type is successfully changed
`nil`	The group type is not changed.

© 2020 All Rights Reserved.

dbCreateGroup

```
dbCreateGroup(
     d_cellViewID
     t_groupName
     l_groupType
     [ t_groupDefName ]
     )
     => d_group / nil
```

Description

Creates a group of the specified group definition in the specified cellview. If a group definition is not specified, a traditional, unrestricted group is created, which allows all types of objects and all types of groups as members and can exist in any database.

© 2020 All Rights Reserved.

Arguments

d_cellViewID	The database object.
t_groupName	Name of the group to be created.
l_groupType	List of group attributes. Valid values: `collection` or `set`—to define the attribute `groupType`. `ordered` or `unordered`—to define the attribute `ordered`. `nonUniqueName` or `uniqueName`—to define the attribute `uniqueName`. `deleteNever`, `deleteFirst`, or `deleteLast`–to define the attribute `deleteWhen`. For more information on valid values of `l_groupType` see Type l group.
t_groupDefName	Name of a group definition. The specified group definition must already exist. If the specified groupDef is an invalid string or does not exist, the function fails to create the group and returns `nil`. Built-in groupDefs include: `unrestricted`, `diffNetPair`, `net`, `matchedNet`, `nestedNet`, and `symmtericNet`

Value Returned

d_group	The *dbObject* of the group if the creation is successful.
`nil`	The group is not created.

Examples

```
dbCreateGroup(cv "myGroup1" list("set" "ordered"))
```

Creates the DM4 group `myGroup1` of the groupDef `unrestricted` in the `cv` cellview with the specified attributes.

```
dbCreateGroup(cv "myGroup2" list("collection" "ordered") "net")
```

Creates the DM4 group `myGroup2` of the groupDef `net` in the `cv` cellview with the specified attributes.

© 2020 All Rights Reserved.

dbDeleteGroupByName

```
dbDeleteGroupByName(
     d_cellView
     t_name
     )
     => t / nil
```

Description

Deletes a group and its members from a cellview.

Arguments

d_cellView	Cellview from which the group is to be deleted.
t_name	Name of the group.

Value Returned

`t`	The group does not exist or it is found and successfully deleted.
`nil`	The group is not deleted.

Example

```
dbDeleteGroupByName(cv "GROUP")
```


© 2020 All Rights Reserved.

dbDeleteObjectFromGroup

```
dbDeleteObjectFromGroup(
     d_group
     d_object
     )
     => t / nil
```

Description

Deletes an object from a group.

Note: For the sake of compatibility, the parameters are also accepted in the opposite order, unless `d_object` is itself a group. If the group has the type *deleteNever*, the group will not be deleted even if the last member is deleted. If it is of type *deleteFirst*, the entire group will be deleted if `d_object` is a member of the group. If the group is of type *deleteLast*, it will be deleted if no other members remain after deleting `d_object`.

Arguments

`d_group`	Database object of the group.
`d_object`	Database object to be deleted from the group.

Value Returned

`t`	The object is deleted.
`nil`	The object is not deleted.

Example

```
dbDeleteObjectFromGroup(groupId objId)
```


© 2020 All Rights Reserved.

dbGetGroupByName

```
dbGetGroupByName(
     d_cellviewId
     t_groupName
     )
     => d_groupId / nil
```

Description

Returns the database ID of the group specifed by name.

Arguments

d_cellviewId	The database ID of the cellview.
t_groupName	The name of the group.

Value Returned

d_groupId	Returns the database ID of the group.
nil	The named group does not exist. An error occurred during execution.

© 2020 All Rights Reserved.

dbGetGroupsWithDef

```
dbGetGroupsWithDef(
     d_cellViewID
     t_groupDefName
     )
     => d_group / nil
```

Description

Returns a list of groups in the specified cellview and of the specified group definition name.

Arguments

d_cellViewID	ID of the database object.
t_groupDefName	Name of the group definition.

Value Returned

d_group	List of groups is returned if the cellview with the specified ID exists.
`nil`	`nil` is returned if the cellview with the specified ID does not exist.

Example

```
dbGetGroupsWithDef( )
```


© 2020 All Rights Reserved.

dbMoveGroupMember

```
dbMoveGroupMember(
     d_groupMem1
     [ d_groupMem2 ]
     )
     => t / nil
```

Description

Moves group member *d_groupMem1* immediately after group member *d_groupMem2.*

Moving group members within a group is how members of ordered groups are reorganized. Group must be ordered and both members must belong to the same group.

Arguments

d_groupMem1	*dbObject* of the group member to be moved to a new position.
d_groupMem2	*dbObject* of the group member whose position is right before *d_groupMem1* when the move is completed. If not passed or nil, *d_groupMem1* is moved to the first position of the list.

Value Returned

t	The move was successful or if *d_groupMem1* equals *d_groupMem2*.
nil	The group member is not moved.

Examples

```
g = dbCreateGroup(cv "GROUP" list("set" "ordered"))
a = dbAddObjectToGroup(g obj1)
b = dbAddObjectToGroup(g obj2)
c = dbAddObjectToGroup(g obj3)
d = dbAddObjectToGroup(g obj4)
dbMoveGroupMember(b c)
dbMoveGroupMember(d)
```


© 2020 All Rights Reserved.

dbDeleteGroupMemberFromGroup

```
dbDeleteGroupMemberFromGroup(
     d_groupMemId
     )
     => t / nil
```

Description

Deletes a particular group member from the group to which it belongs.

Arguments

d_groupMemId	The group member ID.

Value Returned

`t`	The group member is deleted.
`nil`	The group member is not deleted.

Example

```
dbDeleteGroupMemberFromGroup(groupMemId) => t
```


© 2020 All Rights Reserved.

Shape Transformation Functions

The following functions let you transform a shape or object by specifying modifications to the coordinates of the shape or object.

dbTransformCellView

```
dbTransformCellView(
     d_cellViewId
     f_mag
     f_angle
     )
     => t / nil
```

Description

Scales (magnifies) and/or rotates all shapes in a cellview.

Arguments

d_cellViewId	The database cellview ID.
f_mag	The magnification factor - 1.0 is a no-op.
f_angle	The rotation angle in degrees.

Value Returned

`t`	The function executed successfully.
`nil`	An error occurred.

© 2020 All Rights Reserved.

dbTransformPoint

```
dbTransformPoint(
     l_point
     l_list
     )
     => l_point / nil
```

Description

Transforms a coordinate pair.

Using the location and orientation you specify, this function calculates how a certain shape (such as a pin on the master) in an instance of a cell is repositioned. This function performs calculations and is not related to any database object.

Arguments

l_point	Coordinate pair to be transformed.
l_list	List of the following: displacement vector offset, rotation or reflection of the object, and magnification (optional). Valid values for rotation are `R0`, `R90`, `R180`, and `R270`. Valid values for reflection are `MX`, `MXR90`, `MY`, and `MYR90`.

Value Returned

l_point	The coordinate pair of the location of the object after the transformation.
`nil`	The transformation is not successful.

Example

```
dbTransformPoint( (1.5:2.5) list(0:10 "R90" 2.5) )
=> (-6.250000 13.750000)
```


© 2020 All Rights Reserved.

dbTransformPointList

```
dbTransformPointList(
     l_points
     l_transform
     )
     => l_points / nil
```

Description

Transforms all points in an array in the same way as dbTransformPoint does with a single point. This function does not modify the original array, and returns the transformed array.

Arguments

l_points	The array of coordinate pairs to be transformed.
l_transform	`list(l_offset t_transform [f_scale])`

- A transformation object.
  `l_offset`: `list(f_offsetX f_offsetY)`
- An offset vector.
  `t_transform`: A transformation name.
  Valid values are `"R0"`, `"R90"`, `"R180"`, `"R270"`, `"MX"`, `"MXR90"`, `"MY"`, and `"MYR90"`; or, their corresponding symbols such as `'R0`, `'R90`, `'R180`, `'R270`, `'MX`, `'MXR90`, `'MY`, and `'MYR90`.
- A scale ratio (optional).
  `f_scale`
  Default value is `1.0`.

Value Returned

l_points	The list of transformed points.
`nil`	The transformation was not successful.

Example

```
dbTransformPointList(
list((0:0) (0:1) (0:2) (0:3))
list(0:0 "R90" 1.0)
```


© 2020 All Rights Reserved.

```
)
=>
((0.0 0.0)
(-1.0 0.0)
(-2.0 0.0)
(-3.0 0.0))
```


© 2020 All Rights Reserved.

dbTransformBBox

```
dbTransformBBox(
     l_bBox
     l_list
     )
     => l_bBox_new
```

Description

Transforms a bounding box.

Arguments

l_bBox	A list representing the coordinates of a bounding box.
l_list	List of the following: displacement vector for the transformation, rotation or reflection of the objects, and magnification (optional). Valid values for rotation are `R0`, `R90`, `R180`, and `R270`. Valid values for reflection are `MX`, `MXR90`, `MY`, and `MYR90`.

Value Returned

l_bBox_new	A list of the coordinates of the new bounding box.

Example

```
dbTransformBBox(list(1.5:2.5 0:10) list(10:20 "R90"))
=> ((0.0 21.5) (7.5 20.0))
```


© 2020 All Rights Reserved.

dbConcatTransform

```
dbConcatTransform(
     l_transform1
     l_transform2
     )
     => l_transform_new
```

Description

Merges two individual transformations into a single new transformation.

For example, the call `(dbMoveFig F cv (dbConcatTransform T1 T2))` results in figure `F` being moved to the same position and orientation as two successive calls to `dbMoveFig` in the following order:

```
(dbMoveFig F cv T1)
(dbMoveFig F cv T2)
```

Arguments

l_transform1	Defines the first transformation in terms of the displacement vector for the transformation, the rotation or reflection of the objects, and the magnification (optional).
l_transform2	Defines the second transformation in terms of the displacement vector for the transformation, the rotation or reflection of the objects, and the magnification (optional). Valid values for rotation are `R0`, `R90`, `R180`, and `R270`. Valid values for reflection are `MX`, `MXR90`, `MY`, and `MYR90`.

Value Returned

l_transform_new	The new transformation resulting from the merged transformations.

Example

```
dbConcatTransform(list(0:0 "R90") list(1.5:2.5 "R90"))
=> ((1.500000 2.500000) "R180" 1.000000)
```


© 2020 All Rights Reserved.

dbCanonicalizeAnyAngleTransform

```
dbCanonicalizeAnyAngleTransform(
     g_anyAngleTransform...
     )
     => list(t_orient f_acuteAngleDeg)
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Canonicalizes composition of the given transformations into a pair of orientation angles and a non-negative acute angle. Transformations are either numbers representing counter-clockwise rotation in degrees or strings representing orientation. The effect of transformation by the composition of the transformations in the return value is the same as effect of transformation by the composition of all input transformations.

Important

Composition of transforms is not always commutative, therefore the order of arguments is important. Both in the argument list and in the pair, returned transformations are expected to be applied from right to left. For example, `("MX" 45)` means that the object is first rotated by 45 degrees and then is mirrored across the X-axis.

Arguments

g_anyAngleTransform

The function accepts any number of this argument in the following format:

n_angleDeg | *t_orient*

where,

- *n_angleDeg*
  A number representing counter-clockwise rotation in degrees.
- *t_orient*
  A string representing orientation. See Type t_orient.

© 2020 All Rights Reserved.

Value Returned

`list(t_orient f_acuteAngleDeg)`

A list with two elements:
`(t_orient f_acuteAngleDeg)`

where,

- *t_orient*
  A string representing orientation. See Type t_orient.
- *f_acuteAngleDeg*
  A number representing counter-clockwise rotation by acute angle in degrees.

Examples

Any-angle transform with angle greater than or equal to 90 degrees is decomposed into acute angle rotation and rotation by multiple of 90.

```
dbCanonicalizeAnyAngleTransform(90.0)
=> ("R90" 0.0)
dbCanonicalizeAnyAngleTransform(120.0)
=> ("R90" 30.0)
dbCanonicalizeAnyAngleTransform(90.0 "MY")
=> ("MYR90" 0.0)
dbCanonicalizeAnyAngleTransform("MY" 90.0)
=> ("MXR90" 0.0)
dbCanonicalizeAnyAngleTransform(12.0 "MY")
=> ("MYR90" 78.0)
dbCanonicalizeAnyAngleTransform("MY" 12.0)
=> ("MY" 12.0)
dbCanonicalizeAnyAngleTransform("MY" 120.0 "MY")
=> ("R180" 60.0)
dbCanonicalizeAnyAngleTransform("MX" 120.0 "MY")
=> ("R0" 60.0)
dbCanonicalizeAnyAngleTransform(-5) =>
("R270" 85.0)
```


© 2020 All Rights Reserved.

dbConvertAnyAngleTransformFromDbToMY

```
dbConvertAnyAngleTransformFromDbToMY(
     t_orient
     f_angleDeg
     )
     => list(f_angleDeg g_MY)
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Converts orientation-based representation of an orthogonal transformation into an MY-based representation. Orientation-based representation is a pair of DB orientation and any-angle rotation with the rotation applied first. The MY-based representation is a pair of any-angle rotation and an optional reflection over a Y-axis with the reflection applied first.

Arguments

t_orient	A string representing orientation. See Type t_orient.
f_angleDeg	Floating point number representing any-angle rotation in degrees.

Value Returned

`list(f_angleDeg g_boolMY)`

A list with two elements:
`(f_angleDeg g_boolMY)`

where,

- *f_angleDeg*
  A floating point number representing any-angle rotation in degrees. Valid value: Any number between `0` and `360`.
- *g_MY*
  A Boolean (`t` | `nil`) specifying whether "MY" mirroring should be applied.

Example

The following example checks the orientation is flipped:

© 2020 All Rights Reserved.

```
(defun flipsOrientation (t_orient)
    (cadr dbConvertAnyAngleTransformFromDbToMY(t_orient 0.0)))
```


© 2020 All Rights Reserved.

Bounding Box Functions

Figure 2-2 Bounding Box Points for a Polygon

You can abbreviate bounding box point names as follows:

`upperLeft`	or	`uL`	`lowerLeft`	or	`lL`
`upperCenter`	or	`uC`	`lowerCenter`	or	`lC`
`upperRight`	or	`uR`	`lowerRight`	or	`lR`
`centerLeft`	or	`cL`	`centerCenter`	or	`cC`
`centerRight`	or	`cR`			

© 2020 All Rights Reserved.

lowerLeft

```
lowerLeft(
     l_bBox
     )
     => l_list
```

Description

Returns the lower left point of the given bounding box. The bounding box is not validated and assumed to be correct.

Arguments

l_bBox	A list representing a bounding box.

Value Returned

l_list	A list representing the lower left point of the bounding box.

Example

```
lowerLeft( list( list(0 0) list(5 5)) => list(0 0)
```


© 2020 All Rights Reserved.

upperRight

```
upperRight(
     l_bBox
     )
     => l_list
```

Description

Returns the upper right point of the given bounding box. The bounding box is not validated and is assumed to be correct.

Arguments

l_bBox	A list representing a bounding box.

Value Returned

l_list	A list representing the upper right point of the bounding box.

Example

```
upperRight( list( list(0 0) list(5 5)) => list(5 5)
```


© 2020 All Rights Reserved.

leftEdge

```
leftEdge(
     d_object
     )
     => g_coordinate / nil
```

Description

Returns the lower left point's X coordinate of the figure's bounding box.

Arguments

d_object	A database figure

Value Returned

g_coordinate	The X coordinate of the lower left point of the bounding box.
nil	The object is not a database figure or it does not have a bounding box.

© 2020 All Rights Reserved.

rightEdge

```
rightEdge(
     d_object
     )
     => g_coordinate / nil
```

Description

Returns the upper right point's X coordinate of the figure's bounding box.

Arguments

d_object	A database figure

Value Returned

g_coordinate	The X coordinate of the upper right point of the bounding box.
nil	The object is not a database figure or it does not have a bounding box.

© 2020 All Rights Reserved.

bottomEdge

```
bottomEdge(
     d_object
     )
     => g_coordinate / nil
```

Description

Returns the lower left point's Y coordinate of the figure's bounding box.

Arguments

d_object	A database figure.

Value Returned

g_coordinate	The Y coordinate of the lower left point of the bounding box.
nil	The object is not a database figure or it does not have a bounding box.

© 2020 All Rights Reserved.

topEdge

```
topEdge(
     d_object
     )
     => g_coordinate / nil
```

Description

Returns the upper right point's Y coordinate of the figure's bounding box.

Arguments

d_object	A database figure.

Value Returned

g_coordinate	The Y coordinate of the upper right point of the bounding box.
nil	The object is not a database figure or it does not have a bounding box.

© 2020 All Rights Reserved.

centerBox

```
centerBox(
     l_bBox
     )
     => l_point / nil
```

Description

Returns the center point of the given bounding box. The bounding box is not validated and is assumed to be correct.

Arguments

l_bBox	A list representing a bounding box.

Value Returned

l_point	A list representing the center point of the bounding box.
nil	The bounding box is not valid.

Example

```
centerBox( list( 0:0 9:9 ) ) => ( 4 4 )

centerBox( list(0.0:0.0 9.0:9.0 ) ) => ( 4.5 4.5 )
```


© 2020 All Rights Reserved.

isLocation

```
isLocation(
     l_point
     )
     => t / nil
```

Description

Validates that the given point is properly defined.

Note: This function must not be used with pcells.

Arguments

l_point	A list representing a point. For more information, see Type l_point.

Value Returned

`t`	The point is properly defined.
`nil`	The point is not properly defined.

Example

```
isLocation( list( 0 0 )) => t
```


© 2020 All Rights Reserved.

isBBox

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

Description

Validates that the given bounding box is properly defined.

Note: This function must not be used with pcells.

Arguments

l_bBox	A list representing a bounding box.

Value Returned

`t`	The bounding box is properly defined.
`nil`	The bounding box is not properly defined.

Example

```
isBBox( list( list( 0 0 ) list(5 5)) ) => t
```


© 2020 All Rights Reserved.

Property Bag Functions

The property bag is a new database object that contains only properties and has a new identifier, *dbBagId*. Functions in the “Property Creation and Modification” section support the property bag. Thus, they can be used to create and/or modify properties in the property bag. The following section contains property bag functions.

dbOpenBag

```
dbOpenBag(
     g_ddObject
     t_mode
     )
     => g_bagId / nil
```

Description

Opens the library, cell, or view property bag in the given mode.

© 2020 All Rights Reserved.

Arguments

g_ddObject	*ddId* of the library, cell, or view.
t_mode	Mode in which the property bag will be opened. The mode can be one of the following values: ■ `r` Opens the property bag in read mode. The property bag must already exist. ■ `a` Opens the existing property bag in append mode. If the property bag does not exist, it is created. ■ `w` Opens the property bag in write mode. If the property bag does not exist, it is created. If the property bag already exists, its contents is truncated in virtual memory.

Value Returned

g_bagId	*dbBagId* object if the property bag opens successfully.
`nil`	It fails to open the property bag.

Example

```
dbOpenBag(libId "a")
```


© 2020 All Rights Reserved.

dbReOpenBag

```
dbReOpenBag(
     g_bagId
     t_mode
     )
     => t / nil
```

Description

Reopens the given *g_bagId* for the given mode.

Arguments

g_bagId	*g_bagId* to be reopened.
t_mode	Mode in which the property bag will be reopened.

Value Returned

`t`	The bag is reopened in the desired mode.
`nil`	It fails to reopen the bag.

Example

```
dbReOpenBag( bagId "a" )
```


© 2020 All Rights Reserved.

dbGetBag

```
dbGetBag(
     g_ddObject
     )
     => g_bagId / nil
```

Description

Opens the property bag in read mode for the given *ddId* object.

If the property bag is newly created but not yet saved to disk, this function returns `nil`.

Arguments

g_ddObject	*ddId* of library, cell, or view.

Value Returned

g_bagId	The bag is opened successfully.
`nil`	It fails to open the bag.

Example

```
dbGetBag( libId )
```


© 2020 All Rights Reserved.

dbGetBagOwner

```
dbGetBagOwner(
     g_bagId
     )
     => g_ddId / nil
```

Description

Returns the owner of the specified property bag. The owner could be a library, cell, or view.

Arguments

g_bagId	The ID of a property bag.

Value Returned

g_ddId	The ID of the property bag owner.
nil	The function failed.

Example

```
dbGetBagOwner( bagId )
```


© 2020 All Rights Reserved.

dbCloseBag

```
dbCloseBag(
     g_bagId
     [t_mode]
     )
     => t / nil
```

Description

Closes the property bag in the given mode.

If closing for append or write, it will decrement the write count.

Arguments

g_bagId	The ID of the property bag.
t_mode	Mode to close the bag. If omitted, it closes in read mode.

Value Returned

`t`	The bag is closed successfully.
`nil`	It fails to close the bag.

Example

```
dbCloseBag( bagId "a" )
```


© 2020 All Rights Reserved.

dbPurgeBag

```
dbPurgeBag(
     g_bagId
     )
     => t / nil
```

Description

Forces a property bag to close and removes it from virtual memory.

Arguments

g_bagId	Property bag to purge.

Value Returned

`t`	The property bag is purged.
`nil`	The property bag cannot be purged.

Example

```
dbPurgeBag( bagId )
```


© 2020 All Rights Reserved.

dbRefreshBag

```
dbRefreshBag(
     g_bagId
     )
     => t / nil
```

Description

Refreshes the virtual memory image of the property bag by the one from the disk. If the disk file does not exist and the bag is in edit mode, then the virtual memory is truncated.

Arguments

g_bagId	Property bag to be refreshed.

Value Returned

`t`	The bag is refreshed.
`nil`	The property bag cannot be refreshed.

Example

```
dbRefreshBag( bagId )
```


© 2020 All Rights Reserved.

dbMakeBagCurrent

```
dbMakeBagCurrent(
     g_bagId
     )
     => t / nil
```

Description

Refreshes the given property bag if it is out of date relative to the one on disk.

Arguments

g_bagId	Property bag to be refreshed.

Value Returned

`t`	The bag is refreshed or it is already up to date.
`nil`	The property bag cannot be refreshed.

Example

```
dbMakeBagCurrent( bagId )
```


© 2020 All Rights Reserved.

dbSaveBag

```
dbSaveBag(
     g_bagId
     )
     => t / nil
```

Description

Writes the virtual memory image of the property bag to disk.

Arguments

g_bagId	Property bag to be saved.

Value Returned

`t`	The bag is saved.
`nil`	The property bag cannot be written to disk.

Example

```
dbSaveBag( bagId )
```


© 2020 All Rights Reserved.

dbIsBagReadable

```
dbIsBagReadable(
     g_ddObject
     )
     => t / nil
```

Description

Checks if the property bag exists on disk for the given *ddId* object and if the user has read access for that property bag file.

Arguments

g_ddObject	*ddId* of either library, cell, or view.

Value Returned

t	The property bag exists on disk and user has read access for it.
nil	Returns nil otherwise.

Example

```
dbIsBagReadable( libId )
```


© 2020 All Rights Reserved.

dbIsBagWritable

```
dbIsBagWritable(
     g_ddObject
     )
     => t / nil
```

Description

Returns the UNIX file permission of the file that contains the property information. For example, `basic` is a system owned library, `dbIsBagWritable` will always return `nil` on its property. If you own a library, and its prop.xx has write permission, even if the bag is opened in `r` mode, `dbIsBagWritable` will still return `t`. A new property bag that exists only in VM is not considered writable.

Note: `dbIsBagWritable` reports only the UNIX file permission of the file containing the property information. However, if a parent directory of the property bag file is not writable, you will not be able to edit the property bag. To check for this condition, use ddIsObjWritable.

Arguments

g_ddObject	*ddId* of library, cell, or view.

Value Returned

`t`	The property bag already exists for the given *ddId* object and user has write access to the file that contains it. Or, the bag does not exist, but the user has write access to create it for the given *ddId* Object.
`nil`	Returns `nil` otherwise.

Example

```
chmod 555 libFolder
namei -m libFolder/data.dm
=> f: libFolder/data.dm
   dr-xr-xr-x libFolder
   -rwxr-xr-x data.dm

dbIsBagWritable(ddGetObj("libFolder"))
=> t
```


© 2020 All Rights Reserved.

```
ddIsObjWritable(ddGetObj("libFolder"))
=> nil
```


© 2020 All Rights Reserved.

dbGetBagTimeStamp

```
dbGetBagTimeStamp(
     g_bagId
     )
     => t_time / nil
```

Description

Returns the last time that the bag was modified.

Arguments

g_bagId	*dbBagId* object.

Value Returned

t_time	Last time the bag was modified.
nil	The *bagId* is invalid.

Example

```
dbGetBagTimeStamp
```


© 2020 All Rights Reserved.

dbFindOpenBag

```
dbFindOpenBag(
     d_ddId
     )
     => g_bagId / nil
```

Description

Finds an opened bag in virtual memory for the specified *d_ddId*.

Arguments

d_ddId	The object ID.

Value Returned

g_bagId	The ID of the object's property bag.
nil	The *g_bagId* is invalid.

Example

```
dbFindOpenBag( ddId )
```


© 2020 All Rights Reserved.

dbGetBagDdId

```
dbGetBagDdId(
     g_bagId
     )
     => d_ddId / nil
```

Description

Returns the *ddId* associated with the specified *bagId*.

Arguments

g_bagId	*dbBagId* object.

Value Returned

d_ddId	The property bag's dd ID.
nil	The given *g_bagId* is invalid.

Example

```
dbGetBagDdId( bagId )
```


© 2020 All Rights Reserved.

dbGetOpenBags

```
dbGetOpenBags(
     )
     => d_bags / nil
```

Description

Returns a list of the currently open property bags.

Arguments

None

Value Returned

d_bags	A list of currently open property bags.
nil	There are no currently open property bags.

© 2020 All Rights Reserved.

dbIsBagId

```
dbIsBagId(
     g_bagId
     )
     => t / nil
```

Description

Returns `t` if the specified ID is a valid *`bagId`*, otherwise, it returns `nil`.

Arguments

`g_bagId`	*dbBagId* object.

Value Returned

`t`	The specified ID is a valid *`bagId`*.
`nil`	The *`bagId`* is invalid.

Example

```
dbIsBagId( bagId )
```


© 2020 All Rights Reserved.

Layer Boolean Functions

The following are layer Boolean functions.

These functions provide the user with the SKILL APIs to create certain geometries through the use of Boolean (i.e. “AND”) and relational selection operations (i.e. “ENCLOSE”) on existing geometries.

These SKILL APIs take input from one or two lists of database objects of shapes/polygons while the output of the API is created as database objects on a specified output layer. The object ID’s of the created database objects are returned.

© 2020 All Rights Reserved.

dbLayerAnd

```
dbLayerAnd(
     d_cellViewId
     t_layerName
     l_dbIdList1
     l_dbIdList2
     [ x_numVertices ]
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of objects derived from the overlap of objects on the input lists. This function generates the areas common to both input lists. The resulting objects are created in the specified layer and cellview.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list 1.
l_dbIdList2	List of objects from input list 2.
x_numVertices	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects derived from the overlap of objects on the input lists.

Example

```
list1 = geGetSelSet(cvId1)
list2 = geGetSelSet(cvId2)
m3Out = dbLayerAnd(cvId "metal3" list1 list2)
```

Finds the overlap of objects in list1 and list2 and creates the results in layer "metal3" in cellview cvId. The list of output objects is also returned in m3Out.

© 2020 All Rights Reserved.

dbLayerAndNot

```
dbLayerAndNot(
     d_cellViewId
     t_layerName
     l_dbIdList1
     l_dbIdList2
     [ x_numVertices ]
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of new objects derived from areas of the first input list that do not overlap any area of the second input list and creates the resulting objects in the specified layer and cellview.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list 1.
l_dbIdList2	List of objects from input list 2.
x_numVertices	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerAndNot(cvId "metal3" m1Rect m2Rect)
```

Finds the areas of m1Rect that do not overlap the area of m2Rect and creates the results in layer "metal3" in cellview cvId. The list of output objects is also returned in m3Out.

© 2020 All Rights Reserved.

dbLayerOr

```
dbLayerOr(
     d_cellViewId
     t_layerName
     l_dbIdList1
     [ l_dbIdList2 ]
     [ x_numVertices ]
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of new objects that encompass all areas on input lists and creates the resulting objects in the specified layer and cellview.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list 1.
l_dbIdList2	List of objects from input list 2.
x_numVertices	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerOr(cvId "metal3" m1Rect m2Rect)
```

Merges all objects on m1Rect and m2Rect and creates the results in layer "metal3" in cellview cvId. The list of output objects is also returned in m3Out.

© 2020 All Rights Reserved.

dbLayerXor

```
dbLayerXor(
     d_cellViewId
     t_layerName
     l_dbIdList1
     l_dbIdList2
     [ x_numVertices ]
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of new objects derived from those portions of either input list that do not overlap the other list and creates the resulting objects in the specified layer and cellview.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list 1.
l_dbIdList2	List of objects from input list 2.
x_numVertices	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerXor(cvId "metal3" m1Rect m2Rect)
```

Finds the areas of m1Rect and m2Rect that are not in common and creates the results in layer "metal3" in cellview cvId. The list of output objects is also returned in m3Out.

© 2020 All Rights Reserved.

dbLayerSize

```
dbLayerSize(
     d_cellViewId
     t_layerName
     l_dbIdList1
     n_value
     [ x_numVertices ]
     [ g_mfgGrid ]
     [ x_conicSides ]
     )
     => l_dbIdList
```

Performs an oversizing or undersizing operation on the object specified in the input list.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list 1.
n_value	Sizing value.
x_numVertices	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
g_mfgGrid	The manufacture grid to snap the points of output shapes.
x_conicSides	Number of conic sides of output objects. The valid range is `24` through `1000`. If not specified or a value outside the valid range is specified, the default value `24` is used.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Examples

```
m3Out = dbLayerSize(cvId "metal3" m1Rect 0.5)
```

Performs an oversizing by `0.5` user unit on `m1Rect` and creates the results in layer `Metal3` in cellview `cvId`. Also returns a list of output objects in `m3Out`.

```
M2Out = dbLayerSize(cvId "Metal2" geGetSelSet() 0.25 2000 0.001 100)
```


© 2020 All Rights Reserved.

Performs an oversizing by `0.25` user units on the selected figure and creates the resultant output objects in layer `Metal2` in cellview `cvId` with `2000` as numVertices, `0.001` as mfgGrid, and `100` as conicSides. Also returns a list of output objects in `M2Out`.

© 2020 All Rights Reserved.

dbLayerTile

```
dbLayerTile(
     d_cellViewId
     t_layerName
     l_dbIdList1
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Fractures edges of the specified object on the input list into vertical trapezoids.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list 1.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerTile(cvId "metal3" m1Rect)
```

Fractures the edges on objects in m1Rect into trapezoids and creates the results in layer "metal3" in cellview cvId. The list of output objects is also returned in m3Out.

© 2020 All Rights Reserved.

dbLayerStraddle

```
dbLayerStraddle(
     d_cellViewId
     t_layerName
     l_dbIdList1
     l_dbIdList2
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of new objects from those shapes on the first layer that straddle shapes on the second layer. A shape is straddling, if part of its area is covered by part of the shape on the second layer and part is not. The return list also includes copies of shapes that are touching these straddling shapes.
Butting and coincident edges do not affect this function.

Arguments

d_cellViewId	Database ID of the cellview in which to create the output objects.
t_layerName	The layer on which output objects are to be created.
l_dbIdList1	List of objects from input layer 1.
l_dbIdList2	List of objects from input layer 2.
g_mfgGrid	The manufacturing grid to which the output objects must be snapped.

Value Returned

l_dbIdList	List of objects resulting from the layer operation.

Example

```
m3Out = dbLayerStraddle(cvId "metal3" m1Rect m2Rect)
```

Returns a list of objects representing shapes on `m1Rect` that straddle those on `m2Rect` and, including any shapes that touch those straddling shapes `metal3` in cell `cvId`. The list of output objects is also returned in `m3Out`.

© 2020 All Rights Reserved.

dbLayerNoHoles

```
dbLayerNoHoles(
     d_cellViewId
     t_layerName
     l_dbIdList1
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of new objects from those areas consisting of all the original shapes of the input list without holes. A hole is an area created when the perimeter of a polygon touches itself, enclosing an area that is not the polygon.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerNoHoles(cvId "metal3" m1Rect m2Rect)
```

Returns shapes on m1Rect that are not holes of m2Rect and creates the results in layer "metal3" in cellview cvId. The list of output objects is also returned in m3Out.

© 2020 All Rights Reserved.

dbLayerHoles

```
dbLayerHoles(
     d_cellViewId
     t_layerName
     l_dbIdList1
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of new objects from those areas consisting of all the holes in shapes of the input list. A hole is an area created when the perimeter of a polygon touches itself, enclosing an area that is not the polygon.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerHoles(cvId "metal3" m1Rect m2Rect)
```

Returns shapes on m1Rect that are holes of m2Rect and create the results in layer "metal3" in cellview cvId. The list of output objects is also returned in m3.

© 2020 All Rights Reserved.

dbLayerInside

```
dbLayerInside(
     d_cellViewId
     t_layerName
     l_dbIdList1
     l_dbIdList2
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Creates a new merged object for each overlapping object on `dbIdList1` that is completely enclosed by an object on `dbIdList2` and returns a list of the new object/objects created. A shape on `dbIdList1` is considered inside if its entire area is enclosed by the area of a shape on `dbIdList2`. Coincident shapes are considered inside in this check. If both shapes are coincident on all sides, then each shape can be defined as being inside the other.

Arguments

d_cellViewId	Database ID of the cellview in which to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input layer 1.
l_dbIdList2	List of objects from input layer 2.
g_mfgGrid	The manufacturing grid to which the output objects must be snapped.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerInside(cvId "metal3" m1Rect m2Rect)
```

Creates a new metal3 object for each object in list `m1Rect` that is fully enclosed by an object in list `m2Rect` and outputs a list of the newly created objects in `m3Out`.

© 2020 All Rights Reserved.

dbLayerOutside

```
dbLayerOutside(
     d_cellViewId
     t_layerName
     l_dbIdList1
     l_dbIdList2
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Creates a new object for each object on `dbIdList1` that is not enclosed by any object on `dbIdList2` and returns a list of the new object created. A shape on `dbIdList1` is considered outside if none of its area is enclosed by the area of a shape on `dbIdList2`. Coincident shapes are considered inside in this check. Butting edges are considered outside.

Arguments

d_cellViewId	Database ID of the cellview in which to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input layer 1.
l_dbIdList2	List of objects from input layer 2.
g_mfgGrid	The manufacturing grid to which the output objects must be snapped.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerOutside(cvId "metal3" m1Rect m2Rect)
```

Creates a new metal3 object for each object in list `m1Rect` that is not enclosed by any object in list `m2Rect` and outputs a list of the newly created objects in `m3Out`.

© 2020 All Rights Reserved.

dbLayerEnclose

```
dbLayerEnclose(
     d_cellViewId
     t_layerName
     l_dbIdList1
     l_dbIdList2
     [ g_mfgGrid ]
     )
     => l_dbIdList
```

Returns a list of shapes on the first list that enclose shapes on the second list. A shape is enclosing the other if the entire area of a second shape is covered by first shape.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
t_layerName	Layer name to be used for creating the output objects.
l_dbIdList1	List of objects from input list 1.
l_dbIdList2	List of objects from input list 2.
g_mfgGrid	The manufacture grid to snap the points of output shapes.

Value Returned

l_dbIdList	List of objects from the resulting layer operation.

Example

```
m3Out = dbLayerEnclose(cvId "metal3" m1Rect m2Rect)
```

Returns shapes on `m1Rect` that enclose those on `m2Rect` and creates the results in layer `metal3` in cellview `cvId`. The list of output objects is also returned in `m3Out`.

© 2020 All Rights Reserved.

Point Array Functions

The following are point array functions.

© 2020 All Rights Reserved.

dbPointArrayAnd

```
dbPointArrayAnd(
     d_cellViewId
     l_pointList1
     l_pointList2
     [ x_maxPoints ]
     )
     => l_pointList / nil
```

Description

Returns a list of point lists derived from the overlap of shapes in the two input point lists. This function generates the areas common to both the input lists. The resulting point list is created in the specified cellview.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
l_pointList1	Input list 1 with the list of point lists.
l_pointList2	Input list 2 with the list of point lists.
x_maxPoints	Maximum number of points each point array in the output point list can contain.

Value Returned

l_pointList	List of point lists derived from the overlap of shapes in the input lists.

Example

```
pointArray1 = '( ((0 0 ) (0 2) (2 2) (2 0)) ((5 5) (5 7) (7 7) (7 5)) )
pointArray2 = '( ((1 1) (1 6) (6 6) (6 1)) )
dbPointArrayAnd( geGetEditCellView() pointArray1 pointArray2)
```


© 2020 All Rights Reserved.

dbPointArrayAndNot

```
dbPointArrayAndNot(
     d_cellViewId
     l_pointList1
     l_pointList2
     [ x_maxPoints ]
     )
     => l_pointList / nil
```

Description

Returns a list of point lists derived from the area of the first input point list that does not overlap any area of the second input point list. The resulting point list is created in the specified cellview.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
l_pointList1	Input list 1 with the list of point lists.
l_pointList2	Input list 2 with the list of point lists.
x_maxPoints	Maximum number of points each point array in the output point list can contain.

Value Returned

l_pointList	List of points derived from non-overlapping areas from the input lists.

Example

```
pointArray1 = '( ((0 0 ) (0 2) (2 2) (2 0)) ((5 5) (5 7) (7 7) (7 5)) )
pointArray2 = '( ((1 1) (1 6) (6 6) (6 1)) )
dbPointArrayAndNot( geGetEditCellView() pointArray1 pointArray2)
```


© 2020 All Rights Reserved.

dbPointArrayOr

```
dbPointArrayOr(
     d_cellViewId
     l_pointList1
     [ l_pointList2 ]
     [ x_maxPoints ]
     )
     => l_pointList / nil
```

Description

Returns a list of point lists derived from the union of shapes in the two input point lists. The resulting point list is created in the specified cellview. If the second input point list is not specified, the first input point list is returned.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
l_pointList1	Input list 1 with the list of point lists.
l_pointList2	Input list 2 with the list of point lists.
x_maxPoints	Maximum number of points each point array in the output point list can contain.

Value Returned

l_pointList	List of points derived from the union of shapes from the input lists.

Example

```
pointArray1 = '( ((0 0 ) (0 2) (2 2) (2 0)) ((5 5) (5 7) (7 7) (7 5)) )
pointArray2 = '( ((1 1) (1 6) (6 6) (6 1)) )
dbPointArrayOr( geGetEditCellView() pointArray1 pointArray2 4)
```


© 2020 All Rights Reserved.

dbPointArraySize

```
dbPointArraySize(
     d_cellViewId
     l_pointList
     n_value
     [ x_maxPoints ]
     )
     => l_pointList / nil
```

Description

Returns a list of point lists derived after performing an over size or under size operation on the shapes specified in the input list. The resulting point list is created in the specified cellview.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
l_pointList	Input list with the list of point lists.
n_value	The sizing value.
x_maxPoints	Maximum number of points each point array in the output point list can contain.

Value Returned

l_pointList	List of points derived after an over size or under size operation on the input list.

Example

```
pointArray = '( ((1 1) (1 6) (6 6) (6 1)) )
dbPointArraySize( geGetEditCellView() pointArray 1.0)
```


© 2020 All Rights Reserved.

dbPointArrayXor

```
dbPointArrayXor(
     d_cellViewId
     l_pointList1
     l_pointList2
     [ x_maxPoints ]
     )
     => l_pointList / nil
```

Description

Returns a list of point lists derived from portions of both the inputs lists that do not overlap the points in the other input list.

Arguments

d_cellViewId	Database ID of the cellview to create the output objects.
l_pointList1	Input list 1 with the list of point lists.
l_pointList2	Input list 2 with the list of point lists.
x_maxPoints	Maximum number of points each point array in the output point list can contain.

Value Returned

l_pointList	List of points in both input lists that do not overlap points in the other input list.

Example

```
pointArray1 = '( ((0 0 ) (0 2) (2 2) (2 0)) ((5 5) (5 7) (7 7) (7 5)) )
pointArray2 = '( ((1 1) (1 6) (6 6) (6 1)) )
dbPointArrayXor( geGetEditCellView() pointArray1 pointArray2)
```


© 2020 All Rights Reserved.

Soft Connect Terminal Functions

A `softConnect` terminal is a terminal that does not draw or carry any current.

A typical four terminal device has terminals for gate, drain, source, and the bulk node. The bulk node terminal is often not shown in a schematic representation. If the bulk node is shown, it is shown as connected to either power or ground. Similarly in a layout representation, the bulk node terminal is connected to either power or ground, but electrically it does not draw any current. Hence, it can be designated as a `softConnect` terminal.

Terminals are either explicit or implicit. Explicit terminals are explicitly created and managed by the user, whereas, implicit terminals are created and managed automatically by the database. An example of an implicit terminal would be one of the bit terminals associated with an explicit, multi-bit terminal.

Note: Only explicit, single-bit terminals can be designated as `softConnect` terminals.

You can use the `dbSetTermSoftConnect` function to designate a terminal as a `softConnect` terminal.

After you designate a terminal as `softConnect`, you have the option to specify the following two attributes on the terminal:

- `pinlessTerminalLayer:` when the `softConnect` terminal is not implemented by a shape, this layer is used to identify the area of the bulk node for the `softConnect` terminal.
- `connectToLayer:` when the `softConnect` terminal is not implemented by a shape, this identifies the top level layer to which the corresponding `instTerm` will connect.

These attributes can be set independent of one another. When the `softConnect` terminal has an associated physical shape, the `pinlessTerminalLayer` and `connectToLayer` attributes are not needed.

Note: It is an error to try to set either of these two attributes on a terminal that is not designated as `softConnect`.

© 2020 All Rights Reserved.

dbSetTermSoftConnect

```
dbSetTermSoftConnect(
     d_termId
     b_value
     )
     => t / nil
```

Description

Sets the `softConnect` attribute of the specified terminal to the given value.

Note: If the `softConnect` attribute of the terminal is set to `nil`, the associated `pinlessTerminalLayer` and `connectToLayer` attributes are removed (if already set).

The terminal must be an explicit, single-bit terminal. If you set the `softConnect` attribute on a terminal that is not an explicit, single-bit terminal, an error is reported and `nil` is returned.

Arguments

d_termId	The database ID of the terminal.
b_value	Value of the `softConnect` attribute.

Value Returned

`t`	The `softConnect` attribute has been successfully set.
`nil`	The `softConnect` attribute has not been set.

Example

```
dbSetTermSoftConnect( term t)
```


© 2020 All Rights Reserved.

dbGetSoftConnectTermConnectToLayer

```
dbGetSoftConnectTermConnectToLayer(
     d_termId
     )
     => x_layer
```

Description

Returns the `connectToLayer` of the `softConnect` terminal as a layer number. If the `connectToLayer` is not set for this terminal, an error is reported and `nil` is returned.

Arguments

d_termId	The database ID of the terminal.

Value Returned

x_layer	The layer number of the `connectToLayer`.

Example

```
dbGetSoftConnectTermConnectToLayer( term )=> 3
```


© 2020 All Rights Reserved.

dbGetSoftConnectTermPinlessLayer

```
dbGetSoftConnectTermPinlessLayer(
     d_termId
     )
     => tx_layer
```

Description

Returns the `pinlessTerminalLayer` of the `softConnect` terminal as a layer name or number. If the `pinlessTerminalLayer` is not set for this terminal, an error is reported and `nil` is returned.

Arguments

d_termId	The database ID of the terminal.

Value Returned

tx_layer	The layer name or number of the pinlessTerminalLayer.

Example

```
dbGetSoftConnectTermPinlessLayer( term )=> 20
```


© 2020 All Rights Reserved.

dbIsTermSoftConnect

```
dbIsTermSoftConnect(
     d_termId
     )
     => t / nil
```

Description

Checks the `softConnect` attribute value of the specified terminal. Since the `softConnect` attribute can only be set on explicit, single-bit terminals, using this query on an implicit or multi-bit terminal always returns `nil`.

Arguments

d_termId	The database ID of the terminal.

Value Returned

`t`	The `softConnect` attribute of the specified terminal is set.
`nil`	The `softConnect` attribute of the specified terminal is not set.

Example

```
dbIsTermSoftConnect( term )=> t
```


© 2020 All Rights Reserved.

dbSetSoftConnectTermConnectToLayer

```
dbSetSoftConnectTermConnectToLayer(
     d_termId
     tx_layer
     )
     => t / nil
```

Description

Sets the `connectToLayer` attribute of the specified `softConnect` terminal.

If you set the `connectToLayer` attribute on a terminal for which the `softConnect` attribute is not set, an error is reported and `nil` is returned. If the string value of `tx_layer` is an empty string, the value is effectively unset.

Note: If you specify the `tx_layer` value as text, the corresponding named layer must exist in the current technology database.

Arguments

d_termId	The database ID of the terminal.
tx_layer	The layer name or layer number of the terminal.

Value Returned

`t`	The `connectToLayer` on the specified `softConnect` terminal has been successfully set.
`nil`	An error occurred and the `connectToLayer` on the specified `softConnect` terminal could not be set.

Example

```
dbSetSoftConnectTermConnectToLayer( term "metal2")=> t
```


© 2020 All Rights Reserved.

dbSetSoftConnectTermPinlessLayer

```
dbSetSoftConnectTermPinlessLayer(
     d_termId
     tx_layer
     )
     => t / nil
```

Description

Sets the `pinlessTerminalLayer` attribute of the specified `softConnect` terminal. You can specify both the layer name or layer number of the terminal.

If you set the `pinlessTerminalLayer` attribute on a terminal for which the `softConnect` attribute is not set, an error is reported and `nil` is returned. If the string value of `tx_layer` is an empty string, the value is effectively unset.

Note: If you specify the `tx_layer` value as text, the corresponding named layer must exist in the associated technology database.

Arguments

d_termId	The database ID of the terminal.
tx_layer	The layer name or layer number of the terminal.

Value Returned

`t`	The `pinlessTerminalLayer` on the specified `softConnect` terminal has been successfully set.
`nil`	An error occurred and the `pinlessTerminalLayer` on the specified `softConnect` terminal could not be set.

Example

```
dbSetSoftConnectTermPinlessLayer( term "metal1")=> t
```


© 2020 All Rights Reserved.

dbSoftConnectTermHasConnectToLayer

```
dbSoftConnectTermHasConnectToLayer(
     d_termId
     )
     => t / nil
```

Description

Checks if the given terminal has the `softConnect` `connectToLayer` attribute set.

If the given terminal is not a `softConnect` terminal, `nil` is returned.

Arguments

d_termId	The database ID of the terminal.

Value Returned

`t`	The terminal has the `softConnect` `connectToLayer` attribute set.
`nil`	The terminal is not a `softConnect` terminal or does not have the `connectToLayer` attribute set.

Example

```
dbSoftConnectTermHasConnectToLayer( term )=> t
```


© 2020 All Rights Reserved.

dbSoftConnectTermHasPinlessLayer

```
dbSoftConnectTermHasPinlessLayer(
     d_termId
     )
     => t / nil
```

Description

Checks if the given terminal has the `softConnect` `pinlessTerminalLayer` attribute set.

If the given terminal is not a `softConnect` terminal, `nil` is returned.

Arguments

d_termId	The database ID of the terminal.

Value Returned

`t`	The terminal has the `softConnect` `pinlessTerminalLayer` attribute set.
`nil`	The terminal is not a `softConnect` terminal or does not have the `pinlessTerminalLayer` attribute set.

Example

```
dbSoftConnectTermHasPinlessLayer( term )=> t
```


© 2020 All Rights Reserved.

Information Retrieval Functions

dbDebugStats2

```
dbDebugStats2(
     [ d_cellViewId ]
     [ x_segNum ]
     [ g_objName ]
     [ g_outputFileName ]
     )
     => t / nil
```

Description

Lists memory used by every cellview in virtual memory, memory used by a specified cellview, content of a specified segment of the cellview, and number of *objName* objects in the specified segment or cellview.

Arguments

d_cellViewId	The database cellview ID.
x_segNum	The segment number.
g_objName	The object name.
g_outputFileName	Output file name.

Value Returned

`t`	The function executed successfully.
`nil`	An error occurred.

© 2020 All Rights Reserved.

dbGetBuildInfo

```
dbGetBuildInfo(
     )
     => t_dbBuildInfo
```

Description

Returns the OpenAccess release number, the tarkit revision number, and the date and machine on which the database was built.

Arguments

None

Value Returned

t_dbBuildInfo	The release number, tarkit revision number, and the date and machine on which the database was built.

Example

```
dbGetBuildInfo()
"@(#)$OpenAccess: 02.00 s092 06/11/03 00:01 (cds12106) $"
```


© 2020 All Rights Reserved.

dbGetFileDatabaseType

```
dbGetFileDatabaseType(
     t_libName
     t_cellName
     t_viewName
     )
     => t_dbFileDatabaseType
```

Description

Returns the *t_libName*, *t_cellName*, and *t_viewName* database type.

Arguments

t_libName	The library name.
t_cellName	The cell name.
t_viewName	The view name.

Value Returned

t_dbFileDatabaseType	The library/cell/view database type.

Example

```
dbGetFileDatabaseType("newTech18" "M3xM2xG" "layout")
"OpenAccess"
```


© 2020 All Rights Reserved.

dbGetVersion

```
dbGetVersion(
     )
     => t_dbVersion
```

Description

Returns the OpenAccess database version number.

Arguments

None

Value Returned

t_dbVersion	The OpenAccess database version number.

Example

```
dbGetVersion()
59
```


© 2020 All Rights Reserved.

dbSetNIMsgMode

```
dbSetNIMsgMode(
     g_arg
     )
     => t / nil
```

Description

Turns on/off the `not implemented` message.

Arguments

g_arg	Turns the `not implemented` message on or off. Valid Values: `t` (on), `nil` (off)

Value Returned

`t`	The function is executed successfully.
`nil`	*g_arg* was not set to `t` or `nil`.

© 2020 All Rights Reserved.

Blockage Functions

This section describes blockage SKILL functions.

A blockage is an object that indicates a specific area in which objects are not allowed to be placed. There are three types of blockages, area blockage, layer blockage, and area halo.

Owners

Different types of blockages can have different owners. If a blockage has an owner, transforming the owner transforms the blockage object as well. Deleting the owner also deletes the blockage. An owner can have multiple blockages. The owner of the blockage can also vary depending on the sub-type of the blockages.

Type of Blockage	Type of Owner	
areaHalo	instance, prBoundary	not optional
areaBlockage	instance, cluster	optional
layerBlockage	instance, cluster	optional

Blockage Types

Area halo blockages can only be of type `placement`. Area and layer blockages support all of the following blockage sub-types except `placement`.

```
routing
placement
wiring
fill
slot
pin
feedthru
Screen
```


© 2020 All Rights Reserved.

dbCreateAreaBlockage

```
dbCreateAreaBlockage(
     d_cellviewId
     l_points
     [ d_owner ]
     )
     => d_areaBlockageId / nil
```

Description

Creates an area blockage object of the type placement in the given cellview. Area blockages are used to prevent instances from being placed within a specific area.

Arguments

d_cellviewId	The database ID of the cellview.
l_points	Specifies the list of points for the blockage. For more information, see Type l_point.
d_owner	Optional argument indicating the owner ID of the blockage. Default is `nil`.

Value Returned

d_areaBlockageId	When successful, the blockage database ID is returned.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbCreateAreaHalo

```
dbCreateAreaHalo(
     d_ownerId
     n_topOffset
     n_bottomOffset
     n_leftOffset
     n_rightOffset
     )
     => d_areaHaloId / nil
```

Description

Creates an areaHalo blockage object of type `placement`. An area halo is associated with an instance or a prBoundary to represent an extended area around the master's prBoundary.

An area halo supports four offsets which extend the area around a instance or a prBoundary in four orthogonal directions. The offsets in these directions can be different.

Arguments

d_ownerId	The owner ID of the areaHalo blockage.
n_topOffset	Integer or floating-point number indicating the top offset of the area halo.
n_bottomOffset	Integer or floating-point number indicating the bottom offset of the area halo.
n_leftOffset	Integer or floating-point number indicating the left offset of the area halo.
n_rightOffset	Integer or floating-point number indicating the right offset of the area halo.

Value Returned

d_areaHaloId	When successful, the area halo database ID is returned.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbCreateLayerBlockage

```
dbCreateLayerBlockage(
     d_cellviewId
     tx_layer
     t_type
     l_points
     [ d_owner ]
     )
     => d_layerBlockageId / nil
```

Description

Creates a layer blockage object of the specified type in the given cellview.

A layer blockage represents an area on a given layer where an instance cannot be placed. In DFII, a layer blockage cannot be of type `placement`.

Arguments

d_cellviewId	The database ID of the cellview.
tx_layer	Name or number of the layer to be assigned to the blockage.
t_type	String argument indicating the blockage type. Valid Values: `routing`, `viaRouting`, `wiring`, `fill`, `slot`, `pin`, `feedthru`, `Screen`.
l_points	The list of points for the blockage. For more information, see Type l_point.
d_owner	Optional argument indicating the owner of the blockage. Default is `nil`.

Value Returned

d_layerBlockageId	The blockage database ID if the creation is successful.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbCreateLayerHalo

```
dbCreateLayerHalo(
     d_ownerId
     tx_layer
     t_type
     n_top
     n_bottom
     n_left
     n_right
     )
     => d_layerHalo / nil
```

Description

Creates a layer halo blockage object of the specified type.

Arguments

d_ownerId	The database ID of the layer halo owner. The owner can only be an instance or a prBoundary.
tx_layer	Name or number of the layer which the layer halo is on.

t_type	Type of layer halo. Valid values:
	`routing`
	`placement`
	`wiring`
	`fill`
	`slot`
	`pin`
	`feedthru`
	`Screen`

n_top	Integer or floating-point number indicating the top offset of the layer halo.
n_bottom	Integer or floating-point number indicating the bottom offset of the layer halo.

© 2020 All Rights Reserved.

n_left	Integer or floating-point number indicating the left offset of the layer halo.
n_right	Integer or floating-point number indicating the right offset of the layer halo.

Value Returned

d_layerHalo	Returns the database ID of the layer halo.
nil	An error occurred during execution.

© 2020 All Rights Reserved.

Boundary Functions

This section describes boundary SKILL functions.

A boundary is an object that designates an area enclosed by a polygon outline. There are four types of boundary objects, area boundary, snap boundary, prBoundary, and cluster boundary.

Boundary creation functions create a new boundary in the specified cellview with the specified attributes. `numEdges` specifies the number of edges of the boundary and `pointArray` is an array of points for the boundary. `edgeNames` contains the name of each edge and is optional If it is specified, the number of names is assumed to be the same as `numEdges`.

© 2020 All Rights Reserved.

dbCreateAreaBoundary

```
dbCreateAreaBoundary(
     d_cellviewId
     l_points
     [ t_name = nil]
     [ lt_edgeNames = nil]
     )
     => d_areaBoundaryId / nil
```

Description

Creates an area boundary object in the specified cellview.

An area boundary specifies an area in a block for a specific purpose. Area boundaries are named. A block can have multiple area boundaries, each identified by its name.

Arguments

d_cellviewId	The database ID of the cellview.
l_points	Specifies the list of points for the boundary. For more information, see Type l_point.
t_name	Optional argument that specifies the name of the area boundary.
lt_edgeNames	Optional argument that list the names of boundary edges.

Value Returned

d_areaBoundaryId	Returns the database ID of the boundary if the creation is successful.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbCreateClusterBoundary

```
dbCreateClusterBoundary(
     d_clusterId
     t_name
     l_points
     [ lt_edgeNames ]
     )
     => d_boundaryId / nil
```

Description

Creates a cluster boundary object in the cellview of the given cluster. The cluster boundary is associated with a cluster (grouping of objects) and defines the area of the cluster objects.

Arguments

d_clusterId	The database ID of the cluster in which to create the boundary.
t_name	Name of the cluster boundary.
l_points	Specifies the list of points for the boundary. For more information, see Type l_point.
lt_edgeNames	Optional argument that list the names of boundary edges.

Value Returned

d_boundaryId	Returns the database ID of the boundary if the creation is successful.
`nil`	`nil` is returned if the cluster ID is invalid.

© 2020 All Rights Reserved.

dbCreateCoverObstruction

```
dbCreateCoverObstruction(
     d_prBoundaryId
     x_topLayerMaskNumber
     [ g_allowPGNet ]
     )
     => d_coverObstructionId / nil
```

Description

Creates a cover obstruction object associated with the specified prBoundary object.

Arguments

d_prBoundaryId	The database ID of the prBoundary object.
x_topLayerMaskNumber	Specifies the mask number of the top layer up to which the obstruction will be created.
g_allowPGNet	Optional argument, which indicates whether power and ground nets can be routed over the areas defined by the cover obstruction.

Value Returned

d_coverObstructionId	Returns the database ID of the cover obstruction object if the creation is successful.
`nil`	`nil` is returned if the prBoundary ID is invalid.

© 2020 All Rights Reserved.

dbCreatePRBoundary

```
dbCreatePRBoundary(
     d_cellViewId
     l_points
     [ lt_edgeNames ]
     )
     => d_boundaryId / nil
```

Description

Creates a prBoundary object in the specified cellview for place-and-route purpose. A cellview can only have one prBoundary.

Arguments

d_cellViewId	The database ID of the cellview.
l_points	Specifies the points of the prBoundary. For more information, see Type l_point.
lt_edgeNames	Optional argument that list the names of edges.

Value Returned

d_boundaryId	The database ID of the prBoundary if the creation is successful.
`nil`	If a prBoundary already exists for the cellview, a warning is generated and `nil` is returned.

© 2020 All Rights Reserved.

dbCreateSnapBoundary

```
dbCreateSnapBoundary(
     d_cellViewId
     l_points
     [ lt_edgeNames ]
     )
     => d_boundaryId / nil
```

Description

Creates a snap boundary in the specified cellview.

Placement tools use the snap boundary to place instances of the cellview in rows. Depending on rotation and symmetry, the appropriate corner is snapped to the site in the row. A snap boundary must be an rectangle, therefore the number of points in the point array should be four. A cellview can only have one snap boundary.

Arguments

d_cellViewId	The database ID of the cellview.
l_points	Specifies the points of the snap boundary. For more information, see Type l_point.
lt_edgeNames	Optional argument that list the names of edges.

Value Returned

d_boundaryId	The database ID of the snap boundary if the creation is successful.
`nil`	If a snap boundary already exists for the cellview, a warning is generated and `nil` is returned.

© 2020 All Rights Reserved.

dbFindAreaBoundaryByName

```
dbFindAreaBoundaryByName(
     d_cellviewId
     t_name
     )
     => d_areaBoundaryId / nil
```

Description

Finds the area boundary specified by name and cellview ID.

Arguments

d_cellviewId	The cellview ID in which to search for the specified area boundary.
t_name	Name of the area boundary to be found.

Value Returned

d_areaBoundaryId	The database ID of the area boundary when successful.
nil	nil is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbFindClusterBoundaryByName

```
dbFindClusterBoundaryByName(
     d_clusterId
     t_name
     )
     => d_boundaryId / nil
```

Description

Finds the cluster boundary specified by name.

A cluster boundary is attached to a specific cluster. A cluster boundary has a unique name for all cluster boundaries in the cellview.

Arguments

d_clusterId	The cluster ID in which to search for the specified cluster boundary.
t_name	Name of the cluster boundary to be found.

Value Returned

d_boundaryId	The database ID of the boundary when successful.
`nil`	`nil` is returned if the cluster ID is invalid.

© 2020 All Rights Reserved.

dbSetBoundaryEdge

```
dbSetBoundaryEdge(
     d_boundaryId
     l_points
     [ lt_edgeNames ]
     )
     => t / nil
```

Description

Sets the points (edges) and edge names for the specified boundary.

Arguments

d_boundaryId	The database ID of the boundary to be set.
l_points	Specifies a list of points for the boundary. For more information, see Type l_point.
lt_edgeNames	Optional argument that list the names of edges.

Value Returned

`t`	The set of the boundary edges successfully.
`nil`	`nil` is returned if the boundary ID is invalid.

© 2020 All Rights Reserved.

Bus Definition and Bit Order Functions

This section describes bus definition and bit order SKILL functions.

© 2020 All Rights Reserved.

dbCreateBusNetDef

```
dbCreateBusNetDef(
     d_cellviewId
     t_baseName
     t_bitOrder
     )
     => d_busNetDefId / nil
```

Description

Creates a bus net definition with specified bit order and base name in the given cellview.

Arguments

d_cellviewId	The database ID of the cellview.
t_baseName	Specifies the base name for the bus net definition.
t_bitOrder	The bit order for the bus net definition. Valid Values: `none`, `ascendingBitOrder`, and `descendingBitOrder`

Value Returned

d_busNetDefId	When successful, the ID of the bus net definition is returned.
`nil`	If an explicit oaBusNetDef with the specified name already exists, a warning is issued and `nil` is returned.

Example

```
dbCreateBusNetDef(cv "in" "ascendingBitOrder")
```

This creates a busNet definition with base name "in" with ascending bit order.

© 2020 All Rights Reserved.

dbCreateBusTermDef

```
dbCreateBusTermDef(
     d_cellviewId
     t_baseName
     t_bitOrder
     )
     => d_busTermDefId / nil
```

Description

Creates a bus terminal definition with specified bit order and base name in the given cellview.

Arguments

d_cellviewId	The database ID of the cellview.
t_baseName	Specifies the base name for the bus terminal definition.
t_bitOrder	The bit order for the bus terminal definition. Valid Values: `none`, `ascendingBitOrder`, and `descendingBitOrder`

Value Returned

d_busTermDefId	When successful, the ID of the busTermDef is returned.
`nil`	If an explicit bus terminal definition with the specified name already exists, a warning is issued and `nil` is returned.

Example

```
dbCreateBusTermDef(cv "B" "ascendingBitOrder")
```

This creates a bus term definition with base name "B" with ascending bit order.

© 2020 All Rights Reserved.

dbCreateVectorInstDef

```
dbCreateVectorInstDef(
     d_cellviewId
     t_baseName
     t_bitOrder
     )
     => d_vectorInstDefId / nil
```

Description

Creates a vector instance definition with specified bit order and base name in the given cellview.

Arguments

d_cellviewId	The database ID of the cellview.
t_baseName	Specifies the base name for the vector instance.
t_bitOrder	The bit order for the vector instance. Valid Values: `none`, `ascendingBitOrder`, and `descendingBitOrder`

Value Returned

d_vectorInstDefId	When successful, the ID of the vector instance is returned.
`nil`	If an explicit vector instance with the specified name already exists, a warning is issued and `nil` is returned.

Example

```
dbCreateVectorInstDef(cv "A" "ascendingBitOrder")
```

This creates a vectorInst definition with base name "A" and ascending bit order.

© 2020 All Rights Reserved.

dbFindBusNetDefByName

```
dbFindBusNetDefByName(
     d_cellviewId
     t_baseName
     )
     => d_busNetDefId / nil
```

Description

Searches for a bus net definition with specified base name in the given cellview.

Arguments

d_cellviewId	The database ID of the cellview.
t_baseName	Specifies the base name for the bus net definition.

Value Returned

d_busNetDefId	If successful, the ID of the bus net definition is returned.
`nil`	`nil` is returned if the bus net definition does not exist.

Example

```
dbFindBusNetDefByName(cv "in")
```


© 2020 All Rights Reserved.

dbFindBusTermDefByName

```
dbFindBusTermDefByName(
     d_cellviewId
     t_baseName
     )
     => d_busTermDefId / nil
```

Description

Searches for a bus term definition with specified base name in the given cellview.

Arguments

d_cellviewId	The database ID of the cellview.
t_baseName	Specifies the base name for the bus term definition.

Value Returned

d_busNetDefId	If successful, the ID of the bus term definition is returned.
`nil`	`nil` is returned if the bus term definition does not exist.

Example

```
dbFindBusTermDefByName(cv "B")
```


© 2020 All Rights Reserved.

dbFindVectorInstDefByName

```
dbFindVectorInstDefByName(
     d_cellviewId
     t_baseName
     )
     => d_vectorInstDefId / nil
```

Description

Searches for a vector instance definition with specified base name in the given cellview.

Arguments

d_cellviewId	The database ID of the cellview.
t_baseName	Specifies the base name for the vector instance definition.

Value Returned

d_busNetDefId	If successful, the ID of the vector instance definition is returned.
`nil`	`nil` is returned if the vector instance definition does not exist.

Example

```
dbFindVectorInstDefByName(cv "A")
```


© 2020 All Rights Reserved.

Cluster Functions

This section describes cluster SKILL functions.

Cluster objects are used to specify and track a logical association of instances and other clusters within a cellview. They are hierarchical in nature as one cluster can have one or more other clusters as members. Every cluster can also have a set of associated cluster boundaries. A cluster accepts an instance member provided it resides in the same cellview as the cluster and does not already belong to another cluster. A cluster accepts another cluster as a member provided it resides in the same cellview, does not already belong to another cluster, is not an ancestor of its parent, and is not the same cluster as itself.

© 2020 All Rights Reserved.

dbCreateCluster

```
dbCreateCluster(
     d_cellViewId
     t_clusterName
     t_clusterType
     )
     => d_clusterId / nil
```

Description

Creates a cluster in the specified cellview.

Arguments

d_cellViewId	The database ID of the cellview.
t_clusterName	Name of the cluster.
t_clusterType	Indicates the type of cluster. Valid values: `inclusive`: Instances other than the instances in this cluster may be placed in the cluster's region. `exclusive`: Only instances in this cluster may be placed in the cluster's region. `suggested`: Instances in this cluster may be placed outside the cluster's region.

Value Returned

d_clusterId	The database ID of the cluster if the creation is successful.
`nil`	The creation fails. Failure can occur if either another cluster having the name specified already exists in the cellview, the specified *clusterType* is invalid, or the specified cellview is invalid

© 2020 All Rights Reserved.

dbFindClusterByName

```
dbFindClusterByName(
     d_cellViewId
     t_clusterName
     )
     => d_clusterId / nil
```

Description

Searches for a cluster in the specified cellview with the specified name.

Arguments

d_cellViewId	The database ID of the cellview.
t_clusterName	Name of the cluster.

Value Returned

d_clusterId	The database ID of the cluster when successful.
`nil`	`nil` is returned if the cellview ID is invalid or the specified cluster is not found.

© 2020 All Rights Reserved.

Counter Functions

This section describes counter SKILL functions.

© 2020 All Rights Reserved.

dbDisableCellViewCounter

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

Description

Disables updates to the `modifiedCounter` attribute of the cellview.

If the dbSave command is called after you disable updates to the `modifiedCounter` attribute, it will again enable updates to the attribute. Therefore, the `dbSave` command must be called after performing all the required modifications to the attribute. In case you have called `dbSave` to make further edits, ensure to use the `dbDisableCellViewCounter(cv)` command to disable the `modifiedCounter` attribute.

Important

Use this approach if you are making only cosmetic changes to the symbol, and does not require any connectivity-related updates to the schematic.

Do not use this approach when,

- adding or deleting a pin in the symbol
- modifying a pin name, direction, and so on
- changing the location of the existing pins

because it generates an inconsistent schematic design that requires connectivity-related updates but is not flagged by the tool.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

`t`	The updates were disabled.
`nil`	The cellview ID is invalid.

© 2020 All Rights Reserved.

Example

```
dbDisableCellViewCounter(cv)
```


© 2020 All Rights Reserved.

dbEnableCellViewCounter

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

Description

Enables updating of the cellview's timestamp counter.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

`t`	The updating was enabled.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbGetCellViewCounter

```
dbGetCellViewCounter(
     d_cellViewId
     )
     => x_counter / nil
```

Description

Gets the value of the cellview's timestamp counter.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

x_counter	The counter of the cellview.
nil	nil is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbGetFileTimeStamp

```
dbGetFileTimeStamp(
     d_cellViewId
     )
     => t_time / nil
```

Description

Gets the last saved time for the cellview's file.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

t_time	The time the file was last saved.
nil	nil is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbIsConnCurrent

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

Description

Tests whether the value of the connectivity current property is equal to the cellview's timestamp counter value.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

t	The value of the connectivity current property is equal to the cellview's timestamp counter value.
nil	The value of the connectivity current property is not equal to the cellview's timestamp counter value, or there are errors while executing the function.

© 2020 All Rights Reserved.

dbSetConnCurrent

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

Description

Sets the value of the connectivity current property to the cellview's timestamp counter value.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

t	The connectivity current property is successfully set.
nil	The cellview ID is invalid or there is an error.

© 2020 All Rights Reserved.

dbSetInstHeadersCounter

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

Description

Sets the value of the instance header change count property on each instance header to the instance master cellview's timestamp counter value.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

`t`	The change count property is successfully set.
`nil`	The cellview ID is invalid or there is an error.

© 2020 All Rights Reserved.

Interconnect Functions

This section describes interconnect SKILL functions.

dbCreateGuide

dbCreatePathSeg

dbCreateRoute

dbCreateVia

dbCreateCustomViaVariant

dbCreateStdViaVariant

dbFindViaVariantByName

dbGetViaLayer

dbIsCellViewStdViaMaster

dbSetPathSegPoints

dbSetViaParamDefault

dbSetViaShapePurpose

dbGetRouteAuthor

dbSetRouteAuthor

© 2020 All Rights Reserved.

dbCreateGuide

```
dbCreateGuide(
     d_cellviewId
     l_startPoint
     l_endPoint
     [ t_startLayerName ]
     [t_endLayerName ]
     )
     => d_guideId / nil
```

Description

Creates a guide based on the specified parameters.

A guide is a design object that corresponds to the OpenAccess oaGuide object. It specifies a connection between two objects that are intended to be wired together. You can use guides when you know the branching of a wire into segments for each end point and/or the layer constraints, but before you know the detailed location of the wiring.

Guides qualify as route objects, so you can add guides to routes. Routing nets usually replaces guides with wires, or routes.

Arguments

d_cellViewId	The database ID of the cellview.
l_startPoint	List which specifies the starting point of the guide.
l_endPoint	List which specifies the ending point of the guide.
t_startLayerName	Optional argument. The names of the layers in which the guide starts.
t_endLayerName	Optional argument. The names of the layers in which the guide ends.

Value Returned

d_guideId	When successful, returns the database ID of the created guide.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbCreatePathSeg

```
dbCreatePathSeg(
     d_cellviewId
     txl_layerPurpose
     l_beginPt
     l_endPt
     n_width
     t_beginStyle
     t_endStyle
     [list(
          n_beginExt
          n_endExt
          [ l_extValue ]
          )]
     )
     => d_pathSegId / nil
```

Description

Creates a path segment based on the specified parameters. The argument *l_extValue* is an optional list that contains six entries.

A pathSeg object represents a 2-point routing segment. Each point has the following

associated characteristics.

- a layer-purpose pair
- specific width
- specific style

The oaPathSeg class derives from the oaShape class. For more information, see Wire

Elements in Virtuoso Space-based Router User Guide.

Arguments

d_cellViewId	The database ID of the cellview.
txl_layerPurpose	Name, number, or a list of layer or layer-purpose pair.
	For more information, see Type txl_layerPurpose.
l_beginPt	The start point of the path segment.
l_endPt	The end point of the path segment.

© 2020 All Rights Reserved.

n_width	The width of the path segment. For a diagonal pathSeg, you should specify this value in diagonal unit (that is, *n_width** $\sqrt{2}$/2) to create a pathSeg with the desired width *n_width*. The width attribute reflects the oaPathSeg pathStyle width. The width is given in orthogonal units if the path is orthogonal and in diagonal units if the path is diagonal.
t_beginStyle *t_endStyle*	The beginning and ending style of the path segment. Valid values: `truncate`: The pathSeg terminates at the end point with no overlap. `extend`: The pathSeg terminates at one-half of the segment's width past the end point. `variable`: The pathSeg terminates at a specified (*endExt*) distance past the end point. You need to specify the value of *n_beginExt* and/or *n_endExt* in the argument *l_extValue*. `chamfer`: The pathSeg is terminated with an octagonal (chamfered) end style of fixed uniform dimensions. `custom`: A customized style. You need to specify the value set of *n_beginExt*, *n_beginLeftDiagExt*, *n_beginRightDiagExt*, *n_beginRightHalfWidth*, and/or *n_endExt*, *n_endLeftDiagExt*, *n_endRightDiagExt*, *n_endRightHalfWidth* in the argument *l_extValue*.

© 2020 All Rights Reserved.

When either of the pathSeg ends is “custom” style, the width of the pathSeg is determined by:

- When the *beginStyle* is not custom but the *endStyle* is custom: the width of this pathSeg defaults to twice the *n_endRightHalfWidth* value specified.
- When the *endStyle* is not custom but the *beginStyle* is custom: the width of this pathSeg defaults to twice the *n_beginRightHalfWidth* value specified.
- When both the *beginStyle* and the *endStyle* are custom: the width of this pathSeg defaults to the sum of *n_beginRightHalfWidth* and *n_endRightHalfWidth*.
- When neither of the begin or end styles are custom styles: the specified *n_width* value is used.

n_beginExt

(Optional) Extension of the beginning. For a diagonal pathSeg, you should specify this value in diagonal unit (that is, *n_beginExt* * $\sqrt{2}$) to create a pathSeg with the desired extension *n_beginExt*.

n_endExt

(Optional) Extension of the end. For a diagonal pathSeg, user should specify this value in diagonal unit (that is, *n_endExt* * $\sqrt{2}$) to create a pathSeg with the desired extension *n_endExt*.

l_extValue

(Optional) Specifies a list containing six entries that are defined as follows:

```
list( n_beginLeftDiagExt
n_beginRightDiagExt n_beginRightHalfWidth
n_endLeftDiagExt n_endRightDiagExt
n_endRightHalfWidth)
```

These entries are explained as follows:

n_beginLeftDiagExt: left diagonal extension for the beginning of a pathSeg with custom chamfering. For an orthogonal path segment, you should specify this value in diagonal unit (that is, *n_beginLeftDiagExt* * $\sqrt{2}$) to create a pathSeg with the desired beginning left diagonal extension *n_beginLeftDiagExt*.

© 2020 All Rights Reserved.

n_beginRightHalfWidth: the right half width to use at the beginning of a path segment with custom chamfering. For a diagonal pathSeg, you should specify this value in diagonal unit (that is, *n_beginRightHalfWidth*$*\sqrt{2}$) to create a pathSeg with the desired beginning right half width *n_beginRightHalfWidth*. If you do not specify a valid *beginRightHalfWidth* value, a warning message is generated in the CIW, and *beginRightHalfWidth* defaults to half of the segment width.

n_endLeftDiagExt: left diagonal extension for the ending of a pathSeg with custom chamfering. For an orthogonal pathSeg, you should specify this value in diagonal unit (that is, *n_endLeftDiagExt*$*\sqrt{2}$) to create a pathSeg with the desired ending left diagonal extension *n_endLeftDiagExt*.

n_endRightDiagExt: right diagonal extension for the ending of a pathSeg with custom chamfering. For an orthogonal pathSeg, you should specify this value in diagonal unit (that is, *n_endRightDiagExt*$*\sqrt{2}$) to create a pathSeg with the desired ending right diagonal extension *n_endRightDiagExt*.

n_endRightHalfWidth: the right half width to use at the ending of a pathSeg with custom chamfering. For a diagonal pathSeg, you should specify this value in diagonal unit (that is, *n_endRightHalfWidth*$*\sqrt{2}$) to create a pathSeg with the desired ending right half width *n_endRightHalfWidth*. If you do not specify a valid *endRightHalfWidth* value, a warning message is generated in the CIW, and *endRightHalfWidth* defaults to half of the segment width.

Value Returned

d_pathSegId	When successful, returns the database ID of the created path segment.
nil	nil is returned if the cellview ID is invalid.

Examples

```
; Two ways of creating the same pathSeg with zero extension on both ends
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "truncate" "truncate")
```


© 2020 All Rights Reserved.

```
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "variable" "variable" '(0.0 0.0))

; Two ways of creating the same pathSeg with half-width extension at the end
point
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "truncate" "extend")
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "truncate" "variable" '(0.0 0.05))

; Two ways of creating the same pathSeg with half-width extension at the begin
point
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "extend"   "truncate")
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "variable" "truncate" '(0.05 0.0))

; Two ways of creating the same pathSeg with chamfer extension at the begin point
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "chamfer" "truncate")
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "custom"  "truncate" list(0.05 0
'(0.075 0.075 0.05 0.0 0.0 0.0)))


; Two ways of creating the same pathSeg with chamfer extension at the end point
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "truncate" "chamfer")
dbCreatePathSeg(cv "Metal1" 0:0 1:0 0.1 "truncate" "custom"  list(0 0.05 '(0.0
0.0 0.0 0.075 0.075 0.05)))

; Example for creating diagonal pathSeg
width=0.12
cv=geGetWindowCellView()
mfgGrid=techGetMfgGridResolution(techGetTechFile(cv))
diagWidth=round(width*sqrt(2)/mfgGrid)*mfgGrid
dbCreatePathSeg(cv "Metal1" 0:0 2:2 diagWidth "variable" "variable")
dbCreatePathSeg(cv "Metal1" 0:0 2:2 diagWidth "extend" "extend")


; Example for creating a diagonal pathSeg with custom ends
halfDiag=diagWidth/2
halfWidth=width/2
dbCreatePathSeg(cv "Metal1" 5:0 7:2 diagWidth "custom" "custom" list(halfDiag
halfDiag list(halfWidth halfWidth halfDiag halfWidth halfWidth halfDiag)))


; Example for creating an orthogonal pathSeg with custom ends
```


© 2020 All Rights Reserved.

```
dbCreatePathSeg(cv "Metal1" 6:1 2:1 .65 "custom" "custom" list(.6 .6 list(.2 .2 .3
.1 .1 .35)))
```

Some examples of figures that define various terms for a Pathseg are given below.

An orthogonal PathSeg (in which the left and right diagonals of the chamfer are obtained by multiplying the extension by 3/2):

© 2020
All Rights Reserved.

A diagonal PathSeg (in which the left and right diagonals of the chamfer are obtained by multiplying the extension by 2/3):

© 2020
All Rights Reserved.

A PathSeg with two different end styles:

© 2020 All Rights Reserved.

dbCreateRoute

```
dbCreateRoute(
     d_cellviewId
     [ d_netId ]
     [ g_isGlobal ]
     [ t_routeStatus ]
     )
     => d_routeId / nil
```

Description

Creates a route based on the specified parameters.

A route represents a connection between two points. The connection can be logical, physical, or both. Each route can contain representations of the objects to which it connects.

A route can include multiple segments and vias, but a route cannot include a branch. A net with branching is represented by multiple routes. When a route encounters a branching point, a Steiner object represents the connection point at the branch. A Steiner object collects all the routes ending at the branch. It provides a way to traverse a set of routes through their connections.

The objects that support routes include pathSeg, customVia and stdVia, and Guide objects. For more information, see Routing Your Design in *Virtuoso Space-based Router User Guide.*

© 2020 All Rights Reserved.

Arguments

d_cellViewId	The database ID of the cellview.
d_netId	Optional argument. Specifies the net in which to assign the route. Default is `nil`.
g_isGlobal	Optional argument. Specifies whether the net is isGlobal. Default is `nil`.
t_routeStatus	Optional argument. Specifies the placement status of the route. Valid values: `normal`, `fixed`, and `locked`

Value Returned

d_routeId	When successful, returns the database ID of the created route.
`nil`	`nil` is returned if the cellview ID is invalid.

Example

```
dbCreateRoute(cv netId t "normal")
```


© 2020 All Rights Reserved.

dbCreateVia

```
dbCreateVia(
     d_cellviewId
     d_viaDefId
     l_point
     t_orient
     [ l_paramList ] ]
     )
     => d_viaId / nil
```

Description

Creates a custom or standard via according to the specified viaDef type and the optional parameter list

A via is a design object. It corresponds to the OpenAccess oaVia object.

A via represents a physical connection between two adjacent physical objects, such as path segments, that exist on two different layers. A via references a via definition, the oaViaDef, from the technology database that corresponds to the design. For more information, see Working with Vias in *Virtuoso Space-based Router User Guide*.

Note: This function cannot create a custom or standard via using parameters that will create inverted layer and implant enclosure shapes.

Arguments

d_cellViewId	The database ID of the cellview.
d_viaDefId	Specifies the database ID of the viaDef associated with the via.
l_point	Specifies a list of points for the via. For more information, see Type l_point.
t_orient	Specifies the orientation of the via.

© 2020 All Rights Reserved.

l_paramList

For custom vias, the values of the parameters can be optionally specified in the parameter list as follows:

l_param, where *l_param* is `list(t_propName g_value)`

Custom vias are usually Pcells defined in the PDK. These vias can be defined to have their own parameters and related requirements.

For standard vias, the parameters can be optionally specified in the parameter list.

Standard vias are connections between two adjacent signal layers on the chip being connected. The cut layer is where the chip processing cuts away the oxide between the two adjacent signal layers to provide the connection between them.

The paramList should be a list of lists, each of which is a name value pair.

t_cutLayerName n_cutWidth n_cutHeight x_cutRows

x_cutColumns l_cutSpacing l_layer1Enc l_layer2Enc

l_layer1Offset l_layer2Offset l_originOffset [l_imp1Enc

[l_imp2Enc [t_cutPattern]]]

Do not specify defaults in the parameter list. If the parameter list is not specified, then the default via parameter will be used to create the standard via.

© 2020 All Rights Reserved.

It is possible to partially specify the following 2D standard via parameters:

l_cutSpacing, l_layer1Enc, l_layer2Enc, l_layer1Offset, l_layer2Offset, l_originOffset, l_imp1Enc, l_imp2Enc.

A `nil` specification is allowed for either the *x* or *y* value of the 2D parameter, which implies that the value will be retrieved from the default value on the `viaDef`.

If implant layer enclosure values are given for a standard via that is associated with a viaDef which does not specify an implant layer, the enclosure values are ignored. No warning message is issued.

For more information on standard via parameters, see Standard Via Attributes

Value Returned

d_viaId	When successful, returns the database ID of the via.
`nil`	`nil` is returned if the cellview ID is invalid.

Example

Get the cellview id where you want to place the via

```
cvId = geGetEditCellView()
```

Get the technology file id of the library that contains the definition of the via

```
tech = techGetTechFile(cvId)
```

Get the id of the viaDef you want to place

```
viaDefId = techFindViaDefByName(tech "M2_M1v")
viaParams = list(list("cutRows" 3) list("cutColumns" 3))
```

Place the via at the coordinate, and with orientation you want

```
newVia = dbCreateVia(cvId viaDefId list(5 5) "R0" viaParams)
```


© 2020 All Rights Reserved.

dbCreateCustomViaVariant

```
dbCreateCustomViaVariant(
     d_cellViewId
     t_viaVariantName
     t_viaDefName
     l_viaParams
     )
     => d_viaVariantId / nil
```

Description

Creates a custom viaVariant in cellview.

Arguments

d_cellViewId	The database ID of the cellview.
t_viaVariantName	Specifies the name of the viaVariant.
t_viaDefName	Specifies the name of the viaDef associated with the viaVariant.
l_viaParams	The values and types of the parameters of the viaVariant, specified using a parameter list. *l_viaParams*, where *l_viaParams* is `list(t_propName t_propType g_value)`

Value Returned

d_viaVariantId	When successful, returns the database ID of the `viaVariant`.
`nil`	`nil` is returned if the cellview ID is invalid.

Example

```
dbCreateCustomViaVariant(cv "via1" list(list(p1 "float" 0.1)))
```


© 2020 All Rights Reserved.

dbCreateStdViaVariant

```
dbCreateStdViaVariant(
     d_cellViewId t_viaVariantName t_viaDefName
     (t_cutLayerName f_cutLayerWidth f_cutLayerHeight)
     (n_cutRows n_cutCol  (f_cutSpaceX f_cutSpaceY))
     (f_layer1EncX f_layer1EncY)
     (f_layer2EncX f_layer2EncY)
     (f_layer1OffsetX f_layer1OffsetY)
     (f_layer2OffsetX f_layer2OffsetY)
     (f_origOffsetX f_origOffsetY)
     (f_implant1EncX f_implant1EncY)
     (f_implant2EncX f_implant2EncY)
     (cut_pattern)
     )
     => nil / d_viaVariantId
```

Description

Creates a standard via variant with the specified attributes.

Note: This function cannot create standard via variant using parameters that will create inverted layer and implant enclosure shapes.

Arguments

d_cellViewId	Database ID of the cellview.
t_viaVariantName	Name of the standard viaVariant.
t_viaDefName	Name of the viaDef associated with the viaVariant.
t_cutLayerName	Name of the cut layer.
	Valid values: A string that is the name of a valid layer with the layer function cut defined in the technology database
t_cutLayerWidth	Width of a cut, in user units.
	Valid values: Any positive number
t_cutLayerHeight	Height of a cut, in user units.
	Valid values: Any positive number
n_cutRows	Number of rows in the cut array.
	Valid values: Any positive integer

© 2020 All Rights Reserved.

`n_cutCol`	Number of columns in the cut array. Valid values: Any positive integer
`f_cutSpaceX`	The horizontal edge-to-edge distance between cuts, in user units. Valid values: Any positive floating-point number
`f_cutSpaceY`	The vertical edge-to-edge distance between cuts, in user units. Valid values: Any positive floating-point number
`f_layer1EncX`	The horizontal distance from the cut edge to the edge of layer 1, in user units. Valid values: Any floating-point number
`f_layer1EncY`	The vertical distance, from the cut edge to the edge of layer 1, in user units. Valid values: Any floating-point number
`f_layer2EncX`	The horizontal distance from the cut edge to the edge of layer 2, in user units. Valid values: Any floating-point number
`f_layer2EncY`	The vertical distance from the cut edge to the edge of layer 2, in user units. Valid values: Any floating-point number
`f_layer1OffsetX`	The horizontal offset for layer 1 in user units. Valid values: Any floating-point number
`f_layer1OffsetY`	The vertical offset for layer 1 in user units. Valid values: Any floating-point number
`f_layer2OffsetX`	The horizontal offset for layer 2 in user units. Valid values: Any floating-point number
`f_layer2OffsetY`	The vertical offset for layer 2 in user units. Valid values: Any floating-point number
`f_origOffsetX`	The horizontal offset for the origin of the via in user units. Valid values: Any floating-point number

© 2020 All Rights Reserved.

f_origOffsetY	The vertical offset for the origin of the via in user units. Valid values: Any floating-point number
f_implant1EncX	The horizontal distance from the edge of layer 1 to the edge of implant layer 1 in user units. Valid values: Any floating-point number
f_implant1EncY	The vertical distance from the edge of layer 1 to the edge of implant layer 1, in user units. Valid values: Any floating-point number
f_implant2EncX	The horizontal distance from the edge of layer 2 to the edge of implant layer 2, in user units. Valid values: Any floating-point number
f_implant2EncY	The vertical distance from the edge of layer 2 to the edge of implant layer 2, in user units. Valid values: Any floating-point number
l_cutPattern	Cut pattern for the `viaVariant`. It is a 2D array with the value of 0 or 1 for each entry. 0 represents no cut for the correspondent cut in the `viaVariant`. Whereas, 1 represents cut for the correspondent cut in the `viaVariant`.

Value Returned

d_stdViaVariantId	Database ID of the `viaVariant` that was created.
`nil`	The `stdViaVariant` could not be created.

© 2020 All Rights Reserved.

dbFindViaVariantByName

```
dbFindViaVariantByName(
     d_cellViewId
     t_viaVariantName
     )
     => d_viaVariantId / nil
```

Description

Finds the viaVariant in a cellview.

Arguments

d_cellViewId	The database ID of the cellview.
t_viaVariantName	Specifies the name of the viaVariant.

Value Returned

d_viaVariantId	Database ID of the viaVariant that was created.
nil	The specified cellview is a regular cellview or the cellview ID is invalid.

Example

```
dbFindViaVariantByName(cv "via1")
```


© 2020 All Rights Reserved.

dbGetViaLayer

```
dbGetViaLayer(
     d_viaId
     t_viaLayerType
     )
     => t_layerName / nil
```

Description

Returns the layer name of the specified layer of the given via.

Arguments

d_viaId	Specifies the via ID.
t_viaLayerType	Specifies the via layer type. Valid values: `layer1`, `layer2`, `cutLayer`

Value Returned

t_layerName	Returns the layer name for the specified layer of the via.
`nil`	Returns `nil` if an error occurs (for example, invalid via ID or layer type).

Example

```
cv = dbOpenCellViewByType( "tech" "design" "layout" "maskLayout" "w" )
viaDef_M2_M1 = techFindViaDefByName( tf "M2_M1" )
via1t_m2_m1 = dbCreateVia( cv viaDef_M2_M1 10:10 "R0" )
```

```
dbGetViaLayer( via1t_m2_m1 "layer1" )
=> "metal2"
```

```
dbGetViaLayer( via1t_m2_m1 "layer2" )
=> "metal1"
```

```
dbGetViaLayer( via1t_m2_m1 "cutLayer" )
=> "via"
```


© 2020 All Rights Reserved.

dbIsCellViewStdViaMaster

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

Description

Returns a boolean value indicating whether the specified cellview is a standard via master.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

`t`	The specified cellview is a valid standard via master.
`nil`	The specified cellview is a regular cellview or the cellview ID is invalid.

© 2020 All Rights Reserved.

dbSetPathSegPoints

```
dbSetPathSegPoints(
     d_pathSegId
     l_beginPt
     l_endPt
     )
     => t / nil
```

Description

Specifies the beginning and ending points for the given path segment.

If the orientation of pathSeg is changed from orthogonal to diagonal, or conversely, the physical width will change, but the oaPathStyle width will remain the same. This is because the physical width = oaPathSeg width $\sqrt{2}/2$ when the pathSeg is diagonal.

Arguments

d_pathSegId	The database ID of the pathSeg object.
l_beginPt	The beginning point of the segment.
l_endPt	The ending point of the segment.

Value Returned

`t`	When successful, returns `t`.
`nil`	Returns `nil`, if the updates are not successful.

© 2020 All Rights Reserved.

dbSetViaParamDefault

```
dbSetViaParamDefault(
     d_viaId
     t_paramName
     )
     => t / nil
```

Description

Sets the value of a parameter of a specified via to use its default value by a parameter name.

Arguments

d_viaId	The database ID of the via.
t_paramName	The name of a via parameter for the specified via.

Value Returned

d_viaId	Database ID of the via.
`nil`	The cellview ID is invalid.

Example

```
dbSetViaParamDefault(stdViaId "cutWidth")
```

In the example, the via parameter "`cutWidth`" is set to use the default value for the specified standard via.

`stdViaId` represents a standard via ID. The default value of the "`cutWidth`" via parameter is defined in the specified standard via which is pointed by `stdViaId`.

© 2020 All Rights Reserved.

dbSetViaShapePurpose

```
dbSetViaShapePurpose(
     d_customViaId
     tx_purpose
     )
     => t / nil
```

Description

Sets the purposes of all shapes in the `cdsVia` to the specified purpose. The updated parameters include `primaryShapePurpose`, `cutShapePurpose`, and `otherShapePurpose`. To set all purposes to their original values (defined in the `cdsViaDevice` definition), use an empty string as the new `purpose` value.

Arguments

`d_customViaId`	The database ID of the custom `cdsVia` for which the layer purpose parameters are to be set.
`tx_purpose`	The new shape purpose. The purpose can be specified either by the purpose name or the purpose number.

Value Returned

`t`	The layer purpose parameter values for the `cdsVia` were successfully set.
`nil`	`nil` is returned if the layer purpose parameters were not set.

Example

```
dbSetViaShapePurpose(viaId "power")
dbSetViaShapePurpose(viaId 256)
```

To reset layer purposes back to their original values, as specified in the `cdsViaDevice` definition, use:

```
dbSetViaShapePurpose(viaId " ")
```


© 2020 All Rights Reserved.

dbGetRouteAuthor

```
dbGetRouteAuthor(
     d_routeId
     )
     => t_authorType
```

Description

Returns the author type of the specified route.

For more information, see the dbEnableRouteObservers environment variable.

Arguments

d_routeId	Database ID of the route.

Value Returned

t_authorType	Author type of the specified route.

Example

```
; Create a new route
rt1 = dbCreateRoute(MainCV mainNet t "normal")
db:0x1f7b9c1a

; Get the author type of the route rt1
dbGetRouteAuthor(rt1)
"Manual"
```

Additional Information

Enabling Observers

To enable observer, change the value of `dbEnableRouteObservers` environment variable to `t` in the .cdsenv file as shown below:

```
cdba    dbEnableRouteObservers boolean   t
```


© 2020 All Rights Reserved.

dbSetRouteAuthor

```
dbSetRouteAuthor (
     d_routeId,
     t_authorType
     )
      => t / nil
```

Description

Sets the author type for the specified route.

Arguments

d_routeId	Database ID of a route.
t_authorType	Author type to be set for the route.
	Valid values: `Auto`, `Manual`

Value Returned

`t`	Specified author type is set for the given route.
`nil`	Specified author type is not set for the given route.

Example

The following example shows that `t_authorType` can either be `Auto` or `Manual`. Any other value will give an error.

```
; Create a new route
rt1 = dbCreateRoute(MainCV mainNet t "normal")
=> db:0x1f7b9c1a

; Set author type to "Auto"
dbSetRouteAuthor(rt1 "Auto")
=> t

; Check the author type
dbGetRouteAuthor(rt1)
=> “Auto"
```


© 2020 All Rights Reserved.

xdvReplaceInstWithVias

```
xdvReplaceInstWithVias(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Detects and replaces the via-like instances with standard vias in the existing design across the hierarchy of the given library, cell, and view. The function overwrites the existing design and saves it. New via definitions can be created if the technology file is writable.

Arguments

t_libName	Specifies the library name.
t_cellName	Specifies the cell name.
t_viewName	Specifies the view name.

Value Returned

`t`	Via-like instances are detected and replaced with standard vias. This is possible only when correct library, cell, and view have been specified.
`nil`	Library, cell, and view specified do not exist, or there is an error.

Example

```
xdvReplaceInstWithVias("designLib" "Chip_Top" "layout")
```

Detects and replaces via-like instances with standard vias across the hierarchy of the cellview `designLib/Chip_Top/layout`.

© 2020 All Rights Reserved.

Trim Layer Functions (ICADVM20.1 Only)

This sections covers the following topics:

- Trim Functions
- Patch Functions

© 2020 All Rights Reserved.

Trim Functions

This section covers the following trim function.

- dbGetAllTrimFillShapes
- dbGetShapeTrimFillType

dbGetAllTrimFillShapes

```
dbGetAllTrimFillShapes(
     d_cellviewId
     )
     => l_shapeID / nil
```

Description

(ICADVM20.1 Only) Returns a list of database IDs of all shapes present in the specified cellview with a valid trim fill type.

Note: The `dbGetAllTrimFillShapes` function expects the constraint `trimShape` to be present on the trim layer.

Arguments

d_cellviewId	Database ID of a cellview.

Value Returned

l_shapeID	List of database IDs of the shapes with a valid trim fill type.
	Valid trim fill types: `bridgeMetal`, `SADPFillPatch`, `gapFill`
`nil`	No shapes with valid trim fill type were found.

Example

```
cv = dbOpenCellViewByType("testLib" "cellview" "layout" "maskLayout" "w")
dbGetAllTrimFillShapes(cv)
=> (db:0x1d15cf1b db:0x1d15cf1a)
```

Returns the database IDs of the shapes with the trim fill type set as `bridgeMetal` or `SADPFillPatch` in the cellview `cv`.

© 2020 All Rights Reserved.

dbGetShapeTrimFillType

```
dbGetShapeTrimFillType (
     d_shapeId
     )
     => t_trimFillType / nil
```

Description

(ICADVM20.1 Only) Returns the trim fill type for the specified trim shape.

Arguments

d_shapeId	Database ID of a shape.

Value Returned

t_trimFillType	Fill type for the trim shape. Valid values: `notTrim`, `bridgeMetal`, `SADPFillPatch`, `gapFill`
`nil`	Returned in case of failure.

Example

```
s1 = dbCreateRect(cv "M1" list(0:0 1:1))
dbGetShapeTrimFillType(s1)
=> "notTrim"
```

Returns that the fill type for the trim shape `s1` is `notTrim`.

© 2020 All Rights Reserved.

Patch Functions

This section describes the patch SKILL functions.

When routing on metal layers that support trim metal, the router cuts a continuous line of metal with trim shapes. The metal on one side of the trim shape belongs to one net and the metal on the other side belongs to a different net. Based on the manufacturing rules, the trim shapes are frequently not placed at the end of the routing line on both sides. To fix the gap between a shape and the trim shape, an extra metal is added. This extra metal is called as a patch.

For any given pathSeg, a patch is represented by two end styles (begin and end) both, none, or by any one of the ends. The patch type can be any one of the following: `notPatch`, `beginStyle`, `endStyle`, `bothStyle`, `future`, `beginFull`, `endFull`, `bothFull`. For more information, see Supporting Patch Metal Shapes in *Virtuoso Space-based Router User Guide*.

© 2020 All Rights Reserved.

dbGetPatchType

```
dbGetPatchType(
     d_pathSegId
     )
     => t_patchType / nil
```

Description

(ICADVM20.1 Only) Returns the patch type for the specified path segment.

Arguments

d_pathSegId	Specifies the path segment ID.

Value Returned

t_patchType	Returns the patch type for the specified path segment. The patch type can be any of the following: `notPatch`, `beginStyle`, `endStyle`, `bothStyle`, `future`, `beginFull`, `endFull`, `bothFull`.
`nil`	Returns `nil` if an error occurs (for example, invalid pathSeg ID).

Example

```
ps = dbCreatePathSeg(cv "metal1" list(-1 0) list(0 1) .05 "truncate" "truncate")
t db:0x1f7b9c1a
dbGetPatchType(ps)
"notPatch"
```


© 2020 All Rights Reserved.

dbSetBeginPatch

```
dbSetBeginPatch(
     d_pathSegId
     g_value
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Adds or removes the `beginStyle` patch type for the specified path segment depending on the argument `g_value`.

Arguments

d_pathSegId	The database ID of the path segment.
g_value	Value of the patch type. Set to `true` to set the `beginStyle` for the path segment. Else, set to `false` to remove the patch type..

Value Returned

`t`	The patch type for the path segment is successfully set to `beginStyle`.
`nil`	The patch type is not successfully set.

Example

```
ps = dbCreatePathSeg(cv "metal1" list(-1 0) list(0 1) .05 "truncate" "truncate")
db:0x1f7b9c1a
dbSetBeginPatch(ps t) => t
dbGetPatchType(ps) => beginStyle
dbSetEndPatch(ps t) => t
dbGetPatchType(ps) => bothStyle
dbSetBeginPatch(ps nil) => t
dbGetPatchType(ps) => endStyle
```


© 2020 All Rights Reserved.

dbSetEndPatch

```
dbSetEndPatch(
     d_pathSegId
     g_value
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Adds or removes the `endStyle` patch type for the specified path segment depending on the argument `g_value`.

Arguments

d_pathSegId	The database ID of the path segment.
g_value	Value of the patch type. Set to `true` to set the `endStyle` for the path segment. Else, set to `false` to unset it.

Value Returned

`t`	The patch type for the path segment was successfully set to `endStyle`.
`nil`	`nil` is returned if the patch type was not successfully set.

Example

```
ps = dbCreatePathSeg(cv "metal1" list(-1 0) list(0 1) .05 "truncate" "truncate")
db:0x1f7b9c1a
dbSetEndPatch(ps t) => t
dbGetPatchType(ps) => endStyle
dbSetBeginPatch(ps t) => t
dbGetPatchType(ps) => bothStyle
dbSetEndPatch(ps nil) => t
dbGetPatchType(ps) => BeginStyle
```


© 2020 All Rights Reserved.

dbSetFullPatch

```
dbSetFullPatch(
     d_pathSegId
     g_value
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Adds or removes the full style patch type for the specified path segment depending on the argument `g_value`. Path segment needs to have at least one property among `beginStyle` or `endStyle` or both to set the full style patch type.

Arguments

d_pathSegId	The database ID of the path segment.
g_value	Value of the patch type. Set to `true` to set the full style for the path segment. Else, set to `false` to unset it.

Value Returned

`t`	The patch type for the path segment was successfully set to full style.
`nil`	`nil` is returned if the patch type was not successfully set.

Example

```
ps = dbCreatePathSeg(cv "metal1" list(-1 0) list(0 1) .05 "truncate" "truncate")
db:0x1f7b9c1a
dbSetBeginPatch(ps t) => t
dbGetPatchType(ps) => beginStyle
dbSetEndPatch(ps t) => t
dbGetPatchType(ps) => bothStyle
dbSetFullPatch(ps t) => t
dbGetPatchType(ps) => bothFull
```


© 2020 All Rights Reserved.

dbSetPatchType

```
dbSetPatchType(
     d_pathSegId
     t_patchType
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the patch type for the specified path segment.

Arguments

d_pathSegId	The database ID of the path segment.
t_patchTypeurpose	Specifies the patch type for the path segment. The valid patch types are given below. `notPatch`, `beginStyle`, `endStyle`, `bothStyle`, `future`, `beginFull`, `endFull`, `bothFull`.

Value Returned

`t`	The patch type for the path segment was successfully set.
`nil`	`nil` is returned if the patch type was not set.

Example

```
ps = dbCreatePathSeg(cv "metal1" list(-1 0) list(0 1) .05 "truncate" "truncate")
db:0x1f7b9c1a
dbSetPatchType(ps "endStyle")
t
```


© 2020 All Rights Reserved.

Marker Functions

This section describes marker SKILL functions.

A marker is an object which indicates violations between objects in the database. Different types of markers exist for different types of violations. This area of the marker is specified by a point array.

© 2020 All Rights Reserved.

dbCreateMarker

```
dbCreateMarker(
     d_cellviewId
     t_msg
     t_tool
     l_points
     [ ld_objects ]
     [ g_isVisible ]
     [ g_isClosed ]
     [ t_severity ]
     [ t_shortMsg ]
     [ t_deleteWhen ]
     )
     => d_markerId / nil
```

Description

Creates a marker object in the specified cellview.

Arguments

d_cellviewId	The database ID of the cellview.
t_msg	Message associated with the marker.
t_tool	Name value of the application associated with this marker.
l_points	Specifies the list of points for the marker. For more information, see Type l_point.
ld_objects	Optional argument that specifies the list of database ID's of the objects associated with the marker. Default Value: `nil`

g_isVisible	Optional boolean argument that indicates if the marker is visible
	Valid values:
	`t`, `nil`
	Default Value:
	`t`

© 2020 All Rights Reserved.

g_isClosed	Optional argument that specifies if the pointArray of the marker is closed.
	Valid values:
	`t`, `nil`
	Default Value:
	`t`

t_severity	Optional argument that specifies the severity of the violation shown by the marker. Valid values: `annotation`, `info`, `acknowledgedWarning`, `warning`, `signedOffError`, `error`, `signedOffCriticalError`, `criticalError`, `fatalError`, `unsupported`. See the attribute severity for more information. Default Value: `annotation`
t_shortMsg	Optional argument that specifies the `shortMsg` associated with the marker. Default Value: `""`
t_deleteWhen	Optional argument that specifies when markers are automatically deleted. Valid values: `deleteNever`, `deleteFirst`, `deleteLast`, `deleteModify`. See the attribute deleteWhen for more information. Default Value: `"deleteNever"`

Value Returned

d_markerId	The database ID of the marker if the creation is successful.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbAddObjectsToMarker

```
dbAddObjectsToMarker(
     d_markerId
     ld_objectId
     )
     => t / nil
```

Description

Adds a list of objects to a specified marker.

Arguments

d_markerId	The database ID of the marker.
ld_objectId	The database ID's of the list of objects to be added to the marker.

Value Returned

`t`	The list is added successfully.
`nil`	`nil` is returned if the marker ID or object IDs are invalid.

© 2020 All Rights Reserved.

dbRemoveObjectsFromMarker

```
dbRemoveObjectsFromMarker(
     d_markerId
     ld_objectId
     )
     => t / nil
```

Description

Removes a list of objects from a specified marker.

Arguments

d_markerId	The database ID of the marker.
ld_objectId	The database ID's of the list of objects to be removed from the specified marker.

Value Returned

`t`	The list is removed successfully.
`nil`	`nil` is returned if the marker ID or object IDs are invalid.

© 2020 All Rights Reserved.

dbGetMarkerName

```
dbGetMarkerName(
     d_markerId
     )
     => t_markerName / nil
```

Description

Retrieves the name assigned to the marker associated with the given marker ID.

Arguments

d_markerId	Database ID of a marker.

Value Returned

t_markerName	Name assigned to the specified marker. Returns an empty string for unnamed markers.
nil	Marker name was not retrieved.

Example

```
m1 = list( (-2.0:-2.0) (-2.0:2.0) (2.0:2.0) (2.0:-2.0) (-2.0:-2.0) )
marker1 = dbCreateMarker(cv "marker1 - annotation type marker" "layout" m1 nil t t
"annotation" "annotation marker")
dbGetMarkerName(marker1)
=> ""

dbSetMarkerName(marker1 "foo")
dbGetMarkerName(marker1)
=> "foo"
```

Creates a new marker with the specified list of points. Retrieving the name of the unnamed marker returns an empty string. You can assign a specific name to this marker, which can be then retrieved persistently.

© 2020 All Rights Reserved.

dbSetMarkerName

```
dbSetMarkerName(
     d_markerId
     t_markerName
     )
     => t / nil
```

Description

Assigns the specified name to the marker associated with the given marker ID. The name can be an empty string.

Arguments

d_markerId	Database ID associated with the marker to which you want to assign a name.
t_markerName	Name to be assigned to the marker ID.

Value Returned

`t`	Marker name was successfully assigned.
`nil`	Marker name was not assigned.

Example

```
m1 = list( (-2.0:-2.0) (-2.0:2.0) (2.0:2.0) (2.0:-2.0) (-2.0:-2.0) )
marker1 = dbCreateMarker(cv "marker1 - annotation type marker" "layout" m1 nil t t
"annotation" "annotation marker")
dbSetMarkerName(marker1 "foo")
=> t
```

Creates a new marker with the specified list of points. You can assign a specific name to this marker.

© 2020 All Rights Reserved.

Ruler Functions

dbCreateRuler

```
dbCreateRuler(
     d_cellviewId
     l_points
     )
     => d_rulerId / nil
```

Description

Creates a dbRuler object in the design.

Arguments

d_cellviewId	Cellview Id of the design.
l_points	List of ruler points. For more information, see Type l_point.

Value Returned

d_rulerId	The Id of the dbRuler object, if successful.
nil	The creation of ruler object failed.

Example

```
dbCreateRuler (cvId list (list (0 0) list (10:0))
```


© 2020 All Rights Reserved.

Query Functions

This section describes query SKILL functions.

dbBoundaryQuery

```
dbBoundaryQuery(
     d_cellviewId
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_boundary / nil
```

Description

Returns a list of all the boundaries in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.

© 2020 All Rights Reserved.

x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Optional argument. Specifies the boundary filter size. The query produces only boundaries with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.
l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_boundaryId	Returns a list identifying all the boundaries produced during the query.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbFigGroupQuery

```
dbFigGroupQuery(
     d_cellView
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => l_figGroupId / nil
```

Description

Returns a list of all the figGroups in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

`d_cellView`	The database ID of the cellview in which the design hierarchy will be traversed by the query.
`l_bBox`	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
`x_startLevel`	Optional argument. The level of the design hierarchy in which to start the querying process. If the `startLevel` argument is not specified, the start level defaults to 0.
`x_stopLevel`	Optional argument. The level of the design hierarchy in which to stop the querying process. If the `stopLevel` argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.

© 2020 All Rights Reserved.

n_filterSize	Optional argument. Specifies the figGroup filter size. The query produces only figGroups with a size larger than the *filterSize*. If no filterSize argument is specified, the blockage filter size defaults to 0.
l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

l_figGroupId	Returns a list of all the figGroups produced during the query.
nil	The cellview ID is invalid.

© 2020 All Rights Reserved.

dbGuideQuery

```
dbGuideQuery(
     d_cellviewId
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_guide / nil
```

Description

Returns a list of all the guides in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Optional argument. Specifies the guide filter size. The query produces only guides with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.

© 2020 All Rights Reserved.

l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_boundaryId	Returns a list identifying all the guides produced during the query.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbInstQuery

```
dbInstQuery(
     d_cellviewId
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_inst / nil
```

Description

Returns a list of all the instances in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Optional argument. Specifies the instance filter size. The query produces only instances with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.

© 2020 All Rights Reserved.

l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_instanceId	Returns a list identifying all the instances produced during the query.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbLayerBlockageQuery

```
dbLayerBlockageQuery(
     d_cellviewId
     tx_layer
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_layerBlockage / nil
```

Description

Returns a list of all the layer blockages in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
tx_layer	Specifies the layer name or number of the blockage.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.

© 2020 All Rights Reserved.

n_filterSize	Optional argument. Specifies the blockage filter size. The query produces only blockages with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.
l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_layerBlockageId	Returns a list identifying all the layer blockages produced during the query.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbMarkerQuery

```
dbMarkerQuery(
     d_cellviewId
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_marker / nil
```

Description

Returns a list of all the markers in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Optional argument. Specifies the marker filter size. The query produces only markers with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.

© 2020 All Rights Reserved.

l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_markerId	Returns a list identifying all the markers produced during the query.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbPlacementBlockageQuery

```
dbPlacementBlockageQuery(
     d_cellviewId
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_via / nil
```

Description

Returns a list of all the placement blockages in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Optional argument. Specifies the placement blockage filter size. The query produces only blockages with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.

© 2020 All Rights Reserved.

l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_placementBlockageId

Returns a list identifying all the placement blockages produced during the query.

`nil` `nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbRowQuery

```
dbRowQuery(
     d_cellviewId
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_row / nil
```

Description

Returns a list of all the rows in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Optional argument. Specifies the row filter size. The query produces only rows with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.

© 2020 All Rights Reserved.

l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_rowId	Returns a list identifying all the rows produced during the query.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbShapeQuery

```
dbShapeQuery(
     d_cellviewId
     l_layerPurposePair
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => l_occShapes / nil
```

Description

Returns a list of all the shapes in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_layerPurposePair	Layer purpose pair. All shapes on this layer purpose pair will be queried and returned.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.

© 2020 All Rights Reserved.

n_filterSize	Optional argument. Specifies the shapes filter size. The query produces only shapes with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.
l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

© 2020 All Rights Reserved.

Value Returned

l_occShapes `list((d_shapeId | l_occShape)…)`

where,

- *d_shapeId* is the database ID of a shape.
- *l_occShape*
  is
  `list((d_refId | l_mosaicElem)… d_shapeId)`

  where,

  - *d_refId*
    The database ID of the non-mosaic instance or via.
  - *l_mosaicElem*
    is
    `list(d_mosaicId x_row x_col)`
    Represents a mosaic element in the hierarchy path.
    where,

    - *d_mosaicId*
      The database ID of the mosaic instance.
    - *x_row*
      Row in which *d_shapeId* occurs inside this mosaic instance.
    - *x_col*
      Column in which *d_shapeId* occurs inside this mosaic instance.
  - *d_shapeId*
    The database ID of the shape.

`nil` `nil` is returned if the cellview ID is invalid.

Example

```
shapes = dbShapeQuery( deGetCellView() list("M1" "drawing") list(0:0 0:0) )
```

Returns a list containing shapes or lists representing occurrence shapes that are on `M1 drawing`, overlapping the `(0:0 0:0)` point.

© 2020 All Rights Reserved.

dbSteinerQuery

```
dbSteinerQuery(
     d_cellviewId
     tx_layer
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => ld_steiner / nil
```

Description

Returns a list of all the steiners in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
tx_layer	Specifies the layer name or number of the blockage.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.

© 2020 All Rights Reserved.

n_filterSize	Optional argument. Specifies the steiner filter size. The query produces only steiners with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.
l_transform	Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

d_steinerId	Returns a list identifying all the steiners produced during the query.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbViaQuery

```
dbViaQuery(
     d_cellviewId
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => l_hiervia / nil
```

Description

Returns a list of all the vias in a cellview whose bounding boxes overlap the region specified by `l_bBox`.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Arguments

d_cellviewId	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
l_bBox	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
x_startLevel	Optional argument. The level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Optional argument. The level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Optional argument. Specifies the via filter size. The query produces only vias with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.

© 2020 All Rights Reserved.

l_transform Optional argument which describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

l_hiervia Returns a list of hierarchical vias

```
l_hierVias  list((d_viaId | l_hierVia)…)
```

where,

- `d_viaId` is the database ID of a via.

```
l_hierVia is
     list((d_refId | l_mosaicElem)? d_viaId)
     where,
```

  - `d_refId` is the database ID of the non-mosaic instance.
  - *l_mosaicElem* is `list((d_refId | l_mosaicElem)? d_viaId)` representing a mosaic element in the hierarchy path, where,
    - *d_mosaicId* is the database ID of a mosaic instance.
    - *x_row* is the row in which *d_viaId* occurs inside the mosaic instance.
    - *x_col* is the column in which *d_viaId* occurs inside the mosaic instance.
  - *d_viaId* is the database ID of a via.

`nil` `nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

Row Functions

This section describes row SKILL functions.

A row represents a location for the placement of standard cells, macros, or devices. The element definitions, which are usually defined in a DEF file, include the name of a valid site in which to place the elements. Rows are defined as an area associated with a single site. A site has a width and a height as well as symmetry information, allowing the specification of rotation and symmetry of elements within a given row. A row is a repetition of the site element in a given direction (horizontal or vertical).

In order to allow placement of different types of elements, different types of rows need to be created. Rows can be created overlapping each other.

© 2020 All Rights Reserved.

dbCreateRow

```
dbCreateRow(
     d_cellViewId
     d_siteDef
     t_name
     l_point
     x_numSites
     [ t_siteOrient ]
     [ t_rowOrient ]
     )
     => d_rowId / nil
```

Description

Creates a row in the specified cellview.

Arguments

d_cellViewId	The database ID of the cellview.
d_siteDef	The `siteDef` to be used for this row.
t_name	Specifies the name of the row.
l_point	The origin point for the row. For more information, see Type l_point.
x_numSites	The number of site placements in the row.
t_siteOrient	Specifies the orientation of the site placements of the row. Default value: `R0`
t_rowOrient	Specifies the orientation of the row. Default value: `R0`

Value Returned

d_rowId	The database ID of the row if creation is successful.
`nil`	`nil` is returned if the cellview ID is invalid.

© 2020 All Rights Reserved.

dbFindRowByName

```
dbFindRowByName(
     d_cellViewId
     t_name
     )
     => d_rowId / nil
```

Description

Finds the specified row.

Arguments

d_cellViewId	The database ID of the cellview.
t_name	Specifies the name of the row.

Value Returned

d_rowId	The database ID of the row when successful.
nil	The specified row does not exist in the given cellview.

© 2020 All Rights Reserved.

Row Header Functions

This section describes row header SKILL functions.

A row header contains information about a particular `siteDef` including the rows using the `siteDef`.

© 2020 All Rights Reserved.

dbFindRowHeader

```
dbFindRowHeader(
     d_cellViewId
     t_siteDefName
     )
     => d_rowHeaderId / nil
```

Description

Finds the specified row header.

Arguments

d_cellViewId	The database ID of the cellview in which to search for the row header.
t_siteDefName	Name of the `siteDef` associated with the row header.

Value Returned

d_rowHeaderId	The database ID of the row header when successful.
`nil`	The specified row header does not exist in the given cellview.

© 2020 All Rights Reserved.

Steiner Functions

This section describes Steiner SKILL functions.

© 2020 All Rights Reserved.

dbCreateSteiner

```
dbCreateSteiner(
     d_cellViewId
     l_bBox
     [ tx_layer ]
     )
     => d_steinerId / nil
```

Description

Creates a Steiner object in the given cellview.

Arguments

d_cellviewId	Database ID of the cellview.
l_bBox	List of bBox for the Steiner to be created.
tx_layer	(Optional) Layer associated with the created tap steiner.

Value Returned

d_steinerId	ID of the created Steiner.
nil	Returned if the Steiner cannot be created.

Example

```
steiner1 = dbCreateSteiner(cv list( 4:4 8:8))
```


© 2020 All Rights Reserved.

dbCreateTapSteiner

```
dbCreateTapSteiner(
     d_cellViewId
     l_bBox
     [ tx_layer ]
     )
     => d_steiner / nil
```

Description

Creates a tap steiner object with the specified bounding box in the given cellview.

Arguments

d_cellviewId	Database ID of a cellview.
l_bBox	List of bBox for the Steiner to be created.
tx_layer	Layer associated with the created tap steiner.

Value Returned

d_steinerId	ID of the created tap steiner.
nil	Returned if the tap steiner cannot be created.

Example

```
dbCreateTapSteiner(cv list(-5:-5 0:0) "metal3")
```


© 2020 All Rights Reserved.

dbIsTapSteiner

```
dbIsTapSteiner(
     d_object
     )
     => t / nil
```

Description

Checks whether the specified ID is a tap steiner ID.

Arguments

d_object	ID of an object.

Value Returned

`t`	The object is a tap steiner object.
`nil`	The object is not a tap steiner object.

Example

```
dbCreateSteiner(cv list(-10:-10 0:0) "metal2")
> db:0x08cbab12
dbCreateTapSteiner(cv list(-5:-5 0:0) "metal3")
> db:0x2926a51b
(mapcar 'dbIsTapSteiner cv~>steiners)
> (nil t)
dbIsTapSteiner(cv)
> nil
```


© 2020 All Rights Reserved.

Supply and Ground Sensitivity Functions

This section describes supply and ground sensitivity SKILL functions.

© 2020 All Rights Reserved.

dbGetTermGroundSensitivity

```
dbGetTermGroundSensitivity(
     d_termId
     )
     => d_groundTermId / nil
```

Description

Returns the terminal ground sensitivity attributes for the given terminal ID.

The terminal ID is validated as a single bit terminal and a check is done for the property `groundSensitivity` on the terminal.

Arguments

d_termId	The database ID of the terminal.

Value Returned

d_groundTermId	The property value is retrieved. This terminal is searched in the cellview by the name. If found and the terminal is of single bit, the terminal ID is returned.
`nil`	`nil` is returned if the property `groundSensitivity` does not exist on the terminal.

© 2020 All Rights Reserved.

dbGetTermSupplySensitivity

```
dbGetTermSupplySensitivity(
     d_termId
     )
     => d_supplyTermId / nil
```

Description

Returns the terminal supply sensitivity attributes for the given terminal ID.

The terminal ID is validated as a single bit terminal and a check is done for the property `supplySensitivity` on the terminal.

Arguments

d_termId	The database ID of the terminal.

Value Returned

d_supplyTermId	The property value is retrieved. This terminal is searched in the cellview by the name. If found and the terminal is of single bit, the terminal ID is returned.
`nil`	`nil` is returned if the property `supplySensitivity` does not exist on the terminal.

© 2020 All Rights Reserved.

dbSetTermGroundSensitivity

```
dbSetTermGroundSensitivity(
     d_termId
     d_groundTermId
     )
     => t / nil
```

Description

Specifies the terminal ground sensitivity attributes for the given terminal ID.

Both terminals are validated as single bit terminals and a check is done to ensure that both terminals exist in the same cellview. The ground terminal name is obtained and stored as a property on the terminal. If a property already exists on the terminal, it is replaced.

Arguments

d_termId	The database ID of the terminal figure.
d_groundTermId	The database ID of the ground terminal.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if either of the terminal IDs are invalid.

© 2020 All Rights Reserved.

dbSetTermSupplySensitivity

```
dbSetTermSupplySensitivity(
     d_termId
     d_supplyTermId
     )
     => t / nil
```

Description

Specifies the terminal supply sensitivity attributes for the given terminal ID.

Both terminals are validated as single bit terminals and a check is done to ensure that both terminals exist in the same cellview. The supply terminal name is obtained and stored as a property on the terminal. If a property already exists on the terminal, it is replaced.

Arguments

d_termId	The database ID of the terminal figure.
d_groundTermId	The database ID of the supply terminal.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if either of the terminal IDs are invalid.

© 2020 All Rights Reserved.

Terminal Ordering Functions

This section describes terminal ordering SKILL functions.

© 2020 All Rights Reserved.

dbCreateInstTermByPosition

```
dbCreateInstTermByPosition(
     d_netId
     d_instId
     x_position
     )
     => d_instTermId / nil
```

Description

Creates an instTerm connecting to a terminal at the given position and to the specified master instance.

instTerms associated with an instance must either all connect by name/reference or all connected by position.

Arguments

d_netId	The database ID of the net to be used. `nil` can be specified.
d_instId	The instance ID of the master in which the terminal is to be connected.
x_position	The position of a terminal in the instance master.

Value Returned

d_instTermId	Returns `t` if the instTerm is successfully create based on the position.
`nil`	`nil` is be returned if the instance is using connected by name/reference.

Example

```
dbCreateInstTermByPosition(net i1 0)
```

Creates an instTerm for terminal at 0 position for the instance i1.

© 2020 All Rights Reserved.

dbFindTermByPosition

```
dbFindTermByPosition(
     d_cellviewId
     x_position
     )
     => d_termId / nil
```

Description

Finds the terminal assigned to specific position.

Arguments

d_cellViewId	The database ID of the cellview in which to search for the terminal.
x_position	The position of the terminal.

Value Returned

d_termId	Returns the terminal ID for the specified position.
`nil`	`nil` is returned if the cellview ID is invalid.

Example

```
dbFindTermByPosition(cv 0)
```


© 2020 All Rights Reserved.

Terminal Antenna Functions

© 2020 All Rights Reserved.

dbDeleteAllTerminalAntennaRules

```
dbDeleteAllTerminalAntennaRules(
     d_termId
     t_modelName
     )
     => t / nil
```

Description

Deletes all antenna rules from the specified terminal.

Arguments

d_termId	The database ID of the terminal.
t_modelName	The name of the antenna model.

Value Returned

`t`	The figure was added successfully.
`nil`	An error occurred during execution.

Example

```
dbDeleteAllTerminalAntennaRules(termA "defaultAntennaModel")

dbDeleteAllTerminalAntennaRules(termA "secondAntennaModel")
```


© 2020 All Rights Reserved.

dbDeleteTerminalAntennaRuleOnLayer

```
dbDeleteTerminalAntennaRuleOnLayer(
     d_termId
     t_modelName
     t_dataName
     tx_layer
     )
     => t / nil
```

Description

Deletes the antenna rule of the terminal according to the specified antenna model name, antenna data name, and layer.

Arguments

d_termId	The database ID of the terminal.
t_modelName	Specifies the model name. Valid values: `"defaultAntennaModel"`, `"secondAntennaModel"`, `"thirdAntennaModel"`, `"fourthAntennaModel"`
t_dataName	Specifies the data name. Valid values: `"ANTENNAGATEAREA"` `"ANTENNADIFFAREA"` `"ANTENNAPARTIALMETALAREA"` `"ANTENNAPARTIALMETALSIDEAREA"` `"ANTENNAPARTIALCUTAREA"` `"ANTENNAMAXAREACAR"` `"ANTENNAMAXSIDEAREACAR"` `"ANTENNAMAXCUTCAR"`
tx_layer	The layer name.

© 2020 All Rights Reserved.

Value Returned

`t`	The antenna rule was removed successfully.
`nil`	An error occurred during execution.

Example

```
dbDeleteTerminalAntennaRuleOnLayer(
    termA "defaultAntennaModel" "ANTENNAGATEAREA" "M4")
```


© 2020 All Rights Reserved.

dbDeleteTerminalAntennaRules

```
dbDeleteTerminalAntennaRules(
     d_termId
     t_modelName
     t_dataName
     l_antennaValueLayerPairs
     )
     => t / nil
```

Description

Delete the antenna rules of a terminal according to the model name, data name, and value layer pairs.

Arguments

d_termId The database ID of the terminal.

t_modelName Specifies the model name. Valid values:
`"defaultAntennaModel"`, `"secondAntennaModel"`, `"thirdAntennaModel"`, `"fourthAntennaModel"`

t_dataName Specifies the data name. Valid values:

```
"ANTENNAGATEAREA"
"ANTENNADIFFAREA"
"ANTENNAPARTIALMETALAREA"
"ANTENNAPARTIALMETALSIDEAREA"
"ANTENNAPARTIALCUTAREA"
"ANTENNAMAXAREACAR"
"ANTENNAMAXSIDEAREACAR"
"ANTENNAMAXCUTCAR"
```

l_antennaValueLayerPairs

A list of the antenna value layer pairs.

`list(list(g_value tx_layer)...)`

© 2020 All Rights Reserved.

Value Returned

`t`	The antenna rule was removed successfully.
`nil`	An error occurred during execution.

Example

```
dbDeleteTerminalAntennaRules(termA "defaultAntennaModel" "ANTENNAGATEAREA"
'( '(22.2 "M2")))
```


© 2020 All Rights Reserved.

dbGetTerminalAntennaRuleOnLayer

```
dbGetTerminalAntennaRuleOnLayer(
     d_termId
     t_modelName
     t_dataName
     tx_layer
     )
     => g_value / nil
```

Description

Returns an antenna rule of the terminal according to the model name, data name, and layer.

Arguments

d_termId	The database ID of the terminal.
t_modelName	Specifies the model name. Valid values: `"defaultAntennaModel"`, `"secondAntennaModel"`, `"thirdAntennaModel"`, `"fourthAntennaModel"`
t_dataName	Specifies the data name. Valid values: `"ANTENNAGATEAREA"` `"ANTENNADIFFAREA"` `"ANTENNAPARTIALMETALAREA"` `"ANTENNAPARTIALMETALSIDEAREA"` `"ANTENNAPARTIALCUTAREA"` `"ANTENNAMAXAREACAR"` `"ANTENNAMAXSIDEAREACAR"` `"ANTENNAMAXCUTCAR"`
tx_layer	The layer name.

Value Returned

g_value	The value of the antenna rule is returned.
`nil`	An error occurred during execution.

Example

```
dbGetTerminalAntennaRuleOnLayer(termA "defaultAntennaModel" "ANTENNAGATEAREA"
"Metal1")
```


© 2020 All Rights Reserved.

dbGetTerminalAntennaRules

```
dbGetTerminalAntennaRules(
     d_termId
     t_modelName
     t_dataName
     )
     => l_antennaValueLayerPairs / nil
```

Description

Returns a list of value layer pairs for an antenna rule of the specified terminal according to model name and data name.

Arguments

d_termId	The database ID of the terminal.
t_modelName	Specifies the model name. Valid values: `"defaultAntennaModel"`, `"secondAntennaModel"`, `"thirdAntennaModel"`, `"fourthAntennaModel"`
t_dataName	Specifies the data name. Valid values: `"ANTENNAGATEAREA"` `"ANTENNADIFFAREA"` `"ANTENNAPARTIALMETALAREA"` `"ANTENNAPARTIALMETALSIDEAREA"` `"ANTENNAPARTIALCUTAREA"` `"ANTENNAMAXAREACAR"` `"ANTENNAMAXSIDEAREACAR"` `"ANTENNAMAXCUTCAR"`

Value Returned

l_antennaValueLayerPairs	A list of value layer pairs for an antenna rule.
`nil`	An error occurred during execution.

Example

```
dbGetTerminalAntennaRules(termA "defaultAntennaModel" "ANTENNAGATEAREA")
```


© 2020 All Rights Reserved.

```
=> ((12.0 "M2") 50.0)
```


© 2020 All Rights Reserved.

dbSetTerminalAntennaRuleOnLayer

```
dbSetTerminalAntennaRuleOnLayer(
     d_termId
     t_modelName
     t_dataName
     g_value
     tx_layer
     )
     => t / nil
```

Description

Sets an antenna rule to a specified terminal according to model name, data name, and value layer pair.

Arguments

d_termId	The database ID of the terminal.
t_modelName	Specifies the model name. Valid values: `"defaultAntennaModel"`, `"secondAntennaModel"`, `"thirdAntennaModel"`, `"fourthAntennaModel"`
t_dataName	Specifies the data name. Valid values: `"ANTENNAGATEAREA"` `"ANTENNADIFFAREA"` `"ANTENNAPARTIALMETALAREA"` `"ANTENNAPARTIALMETALSIDEAREA"` `"ANTENNAPARTIALCUTAREA"` `"ANTENNAMAXAREACAR"` `"ANTENNAMAXSIDEAREACAR"` `"ANTENNAMAXCUTCAR"`
g_value	The value of the antenna rule.
tx_layer	The layer name.

© 2020 All Rights Reserved.

Value Returned

`t`	The antenna rule was set successfully.
`nil`	An error occurred during execution.

Example

```
dbSetTerminalAntennaRuleOnLayer(termA "defaultAntennaModel" "ANTENNAGATEAREA" 30
"M3")
```


© 2020 All Rights Reserved.

dbSetTerminalAntennaRules

```
dbSetTerminalAntennaRules(
     d_termId
     t_modelName
     t_dataName
     l_antennaValueLayerPairs

     )
     => t / nil
```

Description

Sets an antenna value layer pair list to a terminal according to model name and data name.

Arguments

d_termId	The database ID of the terminal.
t_modelName	Specifies the model name. Valid values: `"defaultAntennaModel"`, `"secondAntennaModel"`, `"thirdAntennaModel"`, `"fourthAntennaModel"`
t_dataName	Specifies the data name. Valid values: `"ANTENNAGATEAREA"` `"ANTENNADIFFAREA"` `"ANTENNAPARTIALMETALAREA"` `"ANTENNAPARTIALMETALSIDEAREA"` `"ANTENNAPARTIALCUTAREA"` `"ANTENNAMAXAREACAR"` `"ANTENNAMAXSIDEAREACAR"` `"ANTENNAMAXCUTCAR"`
l_antennaValueLayerPairs	A list of the antenna value layer pairs. `list(list(g_value tx_layer)...)`

© 2020 All Rights Reserved.

Value Returned

`t`	The antenna rule was set successfully.
`nil`	An error occurred during execution.

Example

```
dbSetTerminalAntennaRules(termA "defaultAntennaModel" "ANTENNAGATEAREA"
'('(22.2 "M2")))
```

```
=> t
```


© 2020 All Rights Reserved.

Track Pattern Functions

This section describes track pattern SKILL functions.

Track patterns are objects which provide guide lines for routing. The line segments of track patterns are arranged in uniformly-spaced orthogonal patterns extending across a base array.

© 2020 All Rights Reserved.

dbCreateTrackPattern

```
dbCreateTrackPattern(
     d_cellviewId
     g_isHorizontal
     n_startCoord
     x_numTracks
     n_trackSpace
     [ tx_layer ]
     )
     => d_trackPatternId / nil
```

Description

Creates a track pattern in the specified cellview.

(ICADVM20.1 Only) To color the track pattern, use dbSetTrackPatternFirstTrackColor or dbSetTrackPatternColorAlternating.

Arguments

d_cellviewId	The database ID of the cellview.
g_isHorizontal	Specifies a horizontal or vertical track pattern. Valid values: `t`: track pattern is horizontal. `nil`: track pattern is vertical.
n_startCoord	Specifies the start coordinate of the track pattern.
x_numTracks	Specifies the number of track lines of the track pattern.
n_trackSpace	Specifies the tracking space between two track lines.
tx_layer	Optional argument that specifies routing layer of the track pattern.

Value Returned

d_TrackPatternId	The database ID of the track pattern if the creation is successful.
`nil`	`nil` is returned if the cellview ID is invalid or one of the given arguments is invalid.

© 2020 All Rights Reserved.

GCell Pattern Function

This section describes GCell pattern SKILL function.

© 2020 All Rights Reserved.

dbCreateGCellPattern

```
dbCreateGCellPattern(
     d_cellviewId
     g_isHorizontal
     n_startCoord
     x_count
     n_spacing
     [ tx_layer ]
     )
     => d_GCellPatternId / nil
```

Description

Creates a GCell pattern in the specified cellview.

Arguments

d_cellviewId	The database ID of the cellview.
g_isHorizontal	Specifies a horizontal or vertical GCell pattern. Valid values: `t`: GCell pattern is horizontal. `nil`: GCell pattern is vertical.
n_startCoord	Specifies the starting coordinate of the GCell pattern.
x_count	Specifies the number of edges for the pattern.
n_spacing	Specifies the spacing between edges.
tx_layer	Optional argument that specifies the layer of the GCell pattern.

Value Returned

d_TrackPatternId	The database ID of the GCell pattern if the creation is successful.
`nil`	`nil` is returned if the cellview ID is invalid or one of the given arguments is invalid.

© 2020 All Rights Reserved.

Via Functions

This section includes the following topics:

- Via GenVia Functions
- Via Header Function

© 2020 All Rights Reserved.

Via GenVia Functions

This section describes the SKILL functions that enable creating and editing of cdsGenVia.

- dbCreateGenViaVariant
- dbIsCellViewGenViaMaster
- dbIsGenVia
- dbIsGenViaHeader
- dbIsGenViaVariant
- dbSetGenViaParamDefault

© 2020 All Rights Reserved.

dbCreateGenViaVariant

```
dbCreateGenViaVariant(
     d_cellviewId
     t_name
     t_viaDefName
     l_paramList
     )
     => viaVariantId / nil
```

Description

Creates a genViaVariant object in the specified cellview according to the specified viaDef and parameter list.

Arguments

d_cellviewId	ID of the cellview where the object is created.
t_name	Name for the genViaVariant object.
t_viaDefName	Name of the viaDef to which the genViaVariant object is associated.
l_paramList	Parameter list specified as a list of lists. Each parameter is specified as a keyword-value pair. All parameters are required to be specified. Valid values: `layer1Purpose`, `layer2Purpose`, `layer1Enclosure`, `layer2Enclosure`, `cutPurpose`, `cutHeight`, `cutWidth`, `cutRows`, `cutColumns`, `alignment`, `originOffset`, `cutPattern`, `cutArrayPatternX`, `cutArrayPatternY`, `cutSpacing`, `layer1ExtraParams`, `layer2ExtraParams`, `cutLayerExtraParams`, and `version`.

Value Returned

viaVariantId	Database ID of the viaVariant that was created.
`nil`	An error occurred.

© 2020 All Rights Reserved.

Example

The following example creates a viaVariant `genVia1` using viaDef `testGV` in cellview `cv` with the specified parameters.

```
gvv= dbCreateGenViaVariant(cv "testGV" "genVia1" list(list('layer1Purpose "net")
list('layer2Purpose "pin") list('cutWidth 3) list('cutPurpose "fill")
list('cutHeight 2) list('cutColumns 5) list('cutRows 10) list('cutSpacing 0.3 0.4)
list('layer1Enclosure list(0.15 0.25 0.5 0.1)) list('layer2Enclosure list(0.25
0.55 0.8 0.2)) list('cutPattern list(list(1 0 1 0) list(1 1 0 1) list(1 0 1 1)))
list('alignment "offset") list('originOffset list(1.1 2)) list('cutArraySpacing
0.7 1) list('version 1) list('layer1ExtraParams list( list( list('purpose "net")
list('enc list(0.6 0.6 0.7 0.7)))))     list('cutLayerExtraParams list(list(
list('enc list(0 2 4 6)) list('purpose "pin")))) list('layer2ExtraParams list(
list(list('enc list(0.1 0.3 0.8 0.7)) list('purpose "drawing"))))
list('cutArrayPatternY list(5 3 1 1)) list('cutArrayPatternX list(2 1 2))))
```


© 2020 All Rights Reserved.

dbIsCellViewGenViaMaster

```
dbIsCellViewGenViaMaster(
     d_cellviewId
     )
     => t / nil
```

Description

Checks whether the specified cellview ID is of a genViaMaster object.

Arguments

d_cellviewId	Database ID of the input cellview.

Value Returned

`t`	The specified cellview is a genViaMaster object.
`nil`	The specified cellview is not a genViaMaster object.

Example

```
dbIsCellViewGenViaMaster(cv)
```

Checks if the cellview `cv` is a genViaMaster object.

© 2020 All Rights Reserved.

dbIsGenVia

```
dbIsGenVia(
     d_objId
     )
     => t / nil
```

Description

Checks whether the specified database object is a genVia object.

Arguments

d_objId	Database ID of the input object.

Value Returned

`t`	Specified object is a genVia object.
`nil`	Specified object is not a genVia object.

Example

```
dbIsGenVia(objId)
```

Checks if the object `objId` is a genVia object.

© 2020 All Rights Reserved.

dbIsGenViaHeader

```
dbIsGenViaHeader(
     d_objId
     )
     => t / nil
```

Description

Checks whether the specified database object is a genViaHeader object.

Arguments

d_objId	ID of the input object

Value Returned

`t`	Specified object is a genViaHeader object.
`nil`	Specified object is not a genViaHeader object.

Example

```
dbIsGenViaHeader(objId)
```

Checks if the object `objId` is a genViaHeader object.

© 2020 All Rights Reserved.

dbIsGenViaVariant

```
dbIsGenViaVariant(
     d_objId
     )
     => t / nil
```

Description

Checks whether the specified database object is a genViaVariant object.

Arguments

d_viaId	ID of the input object

Value Returned

`t`	Specified object is a genViaVariant object.
`nil`	Specified object is not a genViaVariant object.

Example

```
dbIsGenViaVariant(objId)
```

Checks if the object `objId` is a genViaVariant object.

© 2020 All Rights Reserved.

dbSetGenViaParamDefault

```
dbSetGenViaParamDefault(
     d_viaID
     t_name
     )
     => t / nil
```

Description

Sets the specified parameter to its default value based on the given genVia ID and parameter name.

Arguments

d_viaId	ID of a genVia object whose parameter is to be reset.
t_name	Name of the parameter to be reset to the default value.

Value Returned

`t`	The specified parameter is reset to the default value.
`nil`	The specified parameter could not be reset to the default value.

Example

```
dbSetGenViaParamDefault(viaID "cutWidth")
```

Sets the parameter `cutWidth` to its default value for genVia object `viaID`.

© 2020 All Rights Reserved.

Via Header Function

This section describes the following via header SKILL function:

- dbFindViaHeaderByName

Via header objects hold information that is common to all vias referring to a specific `viaDef`. Via header objects reside in cellviews and are created and automatically managed by the database system. Users can only access via headers and their attributes but are not allowed to change them. Based on the two types of via definitions available in the technology database, there are two types of via headers: standard viaHeaders and custom viaHeaders. Standard viaHeaders contain information common to all standard vias in a route that refer to a specific standard via definition. Custom viaHeaders contain information common to all custom vias in a route that refer to a specific custom via definition in the technology database.

If the master cellview associated with a `viaDef` is custom parameterized, custom viaHeaders are further classified as via superHeaders or via subHeaders. Via superHeaders represent all of the instances of a pCell master, regardless of parameter settings. Via subHeaders represent all of the instances that have one specific set of parameter values. Every custom via of a pCell has one subHeader, which, in turn, has one superHeader. CustomViaHeaders for instances that are not pCells are neither via subHeaders nor via superHeaders. Standard viaHeaders always have a fixed set of parameters associated with them and thus always fall into either the superHeader or subHeader category.

© 2020 All Rights Reserved.

dbFindViaHeaderByName

```
dbFindViaHeaderByName(
     d_cellviewId
     t_viaDefName
     )
     => d_viaHeaderId / nil
```

Description

Takes a viaDef name and returns a viaHeader that references that viaDef. The returned viaHeader is always the superHeader. The only exception is for a custom viaDef whose master is not parameterized.

Arguments

d_cellviewId	The database ID of the cellview.
t_viaDefName	A string identifying a viaDef name.

Value Returned

d_viaHeaderId	The database ID of the viaHeader.
`nil`	`nil` is returned if the cellview ID is invalid or there are no viaHeaders corresponding to the specified viaDef.

Example

```
viaHeader = dbFindViaHeaderByName(cvId "viaArray5")
where cvId is a cellview dbId and "viaArray5" identifies a viaDef.
```


© 2020 All Rights Reserved.

Optical Data Functions

When there is at least one net that has its sigType set to an optical sigType, the data is referred to as optical data.

Supported functions include:

- dbCellViewHasOpticalData
- dbIsNetOptical
- dbListTypes

© 2020 All Rights Reserved.

dbCellViewHasOpticalData

```
dbCellViewHasOpticalData(
     d_cellviewId
     )
     => t / nil
```

Description

Checks whether the specified cellview has optical data.

Arguments

d_cellviewId	The database ID of a cellview.

Value Returned

`t`	The cellview contains optical data.
`nil`	The cellview does not contain optical data

Example

```
dbCellViewHasOpticalData(cv_id) => t / nil
```


© 2020 All Rights Reserved.

dbIsNetOptical

```
dbIsNetOptical(
     d_netId
     )
     => t / nil
```

Description

Checks whether the specified net or signal has optical data.

Arguments

d_netId	The database ID of a net.

Value Returned

t	The object is a net or a signal that has optical data.
nil	The specified net or a signal does not contain optical data.

Note: A warning is displayed if the specified ID is not a valid database ID of a net.

Example

```
dbIsNetOptical(netID) => t / nil
```


© 2020 All Rights Reserved.

dbListTypes

```
dbListTypes(
     t_typeName
     )
     => l_enumValues / nil
```

Description

Lists the enumerated values for a specified database enumerated type.

Arguments

t_typeName	Specifies the name of the DB enumerated type. **Note:** Currently, only `dbSigType` is supported.

Value Returned

l_enumValues	List of the enum values.
`nil`	An invalid type name is specified for the DB enumerated type.

Example

```
dbListTypes("dbSigType")
(
"unknown" "signal" "ground" "supply" "clock"  "testLatch" "analog" "tieOff" "tieHi"
"tieLo"    "scan"  "reset" "optical" "singleModeOptical" "multiModeOptical"
)
```

Lists the enum values for the dbSigtype enumerated type.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

3

Cellview Input/Output Database Access Functions

Before you can access the database, you need to set the `cds.lib` file so the database can locate the cells in your design. You can then open various cellviews in different libraries located in directories listed in the `cds.lib` file. When you open a cellview, you can use other database functions to examine and modify the contents of the cellview. After you finish with a cellview, close it so the memory associated with that cellview can be reused.

This chapter describes the following cellview input/output database SKILL functions:

- dbAllCellViews
- dbCellViewHasVirtHier
- dbCheckRecursion
- dbClose
- dbComputeBBox
- dbComputeBBoxNoNLP
- dbCopyCellView
- dbCreateLib
- dbDefineProc
- dbFeaturePrintInfo
- dbFindOpenCellView
- dbFlattenInst
- dbFlattenInst2
- dbFullLibPath
- dbFullPath

© 2020 All Rights Reserved.

- dbGetAnyInstSwitchMaster
- dbGetCellViewDdId
- dbGetLibDataCompressionLevel
- dbGetOpenCellViews
- dbHasAutoSavedFile
- dbHasPanicFile
- dbIsNonSKILLPcell
- dbOpenAutoSavedCellView
- dbOpenCellViewByType
- dbOpenHier
- dbOpenPanicCellView
- dbOpenParamCellView
- dbProduceMemName
- dbPurge
- dbRefreshCellView
- dbRefreshLibDataCompressionLevel
- dbRegPostSaveAsTrigger
- dbRegPostSaveTrigger
- dbRegPurgeTrigger
- dbRegSaveTrigger
- dbReopen
- dbRestoreAndOpenAutoSavedFile
- dbRestoreAndOpenPanicFile
- dbSave
- dbSearchCDF
- dbSetAutoSave
- dbSetLibDataCompressionLevel

© 2020 All Rights Reserved.

- dbUndefineProc
- dbUnregPostSaveAsTrigger
- dbUnregPostSaveTrigger
- dbUnregPurgeTrigger
- dbUnregSaveTrigger
- dbUnsetLibDataCompressionLevel
- dbWriteSkill
- dbWriteSkillWithLib

© 2020 All Rights Reserved.

dbAllCellViews

```
dbAllCellViews(
     g_lib
     t_cellName
     [ d_contextCellView ]
     )
     => l_viewName / nil
```

Description

Lists all the view names of a cell.

The cell name is defined by *t_cellName*. If *g_lib* is `nil`, then search the library of the *d_contextCellView*. Once the cell is found, the names of the views associated with that cell are returned.

Arguments

g_lib	The `ddId` of the library to search for the cell.
t_cellName	The cell name.
d_contextCellView	A library's `dbCellViewId` to search for the cell if *g_lib* is not specified.

Value Returned

l_viewName	A list of view names.
`nil`	The view names are not listed.

Example

```
dbAllCellViews(libId "74LS169")
=> ("schematic" "symbol")
```


© 2020 All Rights Reserved.

dbCellViewHasVirtHier

```
dbCellViewHasVirtHier(
     cellviewID
     )
     => t / nil
```

Description

Determines whether the specified cellview contains virtual hierarchy data.

Arguments

cellviewd	Database ID of a cellview.

Value Returned4

t	Cellview contains virtual hierarchy data.
nil	Cellview does not contain virtual hierarchy data.

Example

```
dbCellViewHasVirtHier(cellView)
```


© 2020 All Rights Reserved.

dbCheckRecursion

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

Description

Checks for occurrence of the given top cellview in the hierarchy of the `instHeader` masters or any other recursion inside the `instHeader` master's hierarchy.

So whether the hierarchy is A->B->C->A or A->B->C->B, `dbCheckRecursion(A)` catches the recursion.

Arguments

d_cellViewId	Top cellview to check if there is recursion.

Value Returned

`t`	There is no recursion.
`nil`	There is recursion along with warning messages.

Example

```
dbCheckRecursion( topCellView )
```


© 2020 All Rights Reserved.

dbClose

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

Description

Closes a cellview.

When the number of closes to the cellview equals the number of opens, the cellview is purged from virtual memory, freeing the memory so it can be reused in subsequent `dbOpenCellViewByType` calls. Internally, the system maintains the count for the number of openings and closings.

Only one user at a time can open a cellview for append or write. It is recommended that you close all cellviews that you open.

`dbClose` will close the cellview twice when the reference count is two (2), thereby removing the cellview from virtual memory.

Arguments

d_cellViewId	The cellview to be closed.

Value Returned

`t`	The cellview is closed.
`nil`	The cellview is not closed.

Example

Save the changes and close the cellview.

```
dbSave(cellview)
dbClose(cellview)
```


© 2020 All Rights Reserved.

dbComputeBBox

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

Description

Recomputes the given cellview's bounding box.

Arguments

d_cellViewId	The cellview ID.

Value Returned

`t`	The computation is successful
`nil`	The given cellview id is invalid

Example

```
cv = dbOpenCellViewByType( "test" "myBBox" "layout" )
dbComputeBBox(cv) => t
```


© 2020 All Rights Reserved.

dbComputeBBoxNoNLP

```
dbComputeBBoxNoNLP(
     d_cellViewId
     )
     => l_bBox / nil
```

Description

Computes the given cellview's bounding box, but excludes all the bounding boxes of the `nlp` labels in the given cellview itself and its reference cellviews.

Arguments

d_cellViewId	The cellview ID.

Value Returned

l_bBox	A bBox.
`nil`	Errors were encountered.

Example

```
cv = dbOpenCellViewByType ("test" "hasNLP" "layout" "maskLayout" "w")
myLabel = dbCreateLabel (cv 67 5:-21 "label1" "lowerLeft" "RO" "stick" 0.75 )
myLabel~>labelType = "NLPLabel"
myBoxNoNLP = dbComputeBBoxNLP(cv)
```


© 2020 All Rights Reserved.

dbCopyCellView

```
dbCopyCellView(
     d_srcCVId
     t_libName
     t_cellName
     t_viewName
     [ t_version
     [ g_contextId
     [ g_overwrite ] ] ]
     )
     => d_cellViewId / nil
```

Description

Copies a cellview to a destination cellview.

© 2020 All Rights Reserved.

Arguments

d_srcCVId	Source cellview to be copied.
t_libName	Destination library to which the file is to be copied.
t_cellName	Destination cell name.
t_viewName	Destination view name.
t_version	This argument is preserved for compatibility with earlier versions of the software and should be `nil` if specified.
g_contextId	This argument is preserved for compatibility with earlier versions of the software and its contents is ignored.
g_overwrite	Specifies whether a cellview in the destination library is to be over written. If this argument is `nil` (the default value), no cellviews are overwritten. If this argument is `t`, the cellviews are overwritten.

Value Returned

d_cellViewId	The destination cellview ID.
`nil`	The cellview is not copied.

Example

In the following example, cellview A is copied to library test8, cell A2, view layout. If cellview A already exists in the destination library, it is overwritten. The return value is the destination cellview ID.

```
dbCopyCellView(A "test8" "A2" "layout" nil nil t)
=> db:0x16683812
```


© 2020 All Rights Reserved.

dbCreateLib

```
dbCreateLib(
     t_libName
     [ t_libPath ]
     )
     => ddId_libId / nil
```

Description

Creates a library named *t_libName*.

If *t_libPath* is specified, the *t_libPath* is appended to the *t_libName.* This path is created in the *cds.lib* file. No *t_libName* directory is created. If *t_libPath* is omitted, a directory named *t_libName* is created, but its *libPath* in `cds.lib` does not include the *t_libName* itself.

Arguments

t_libName	The library name to be created.
t_libPath	Library path to be appended to the library name.

Value Returned

ddId_libId	If the library is created successfully, the *ddId* of the library is returned.
`nil`	Otherwise `nil` is returned.

Examples

```
libId = dbCreateLib( "myLibrary" )
libId = dbCreateLib( "myLibrary" "/tmp" )
```


© 2020 All Rights Reserved.

dbDefineProc

```
dbDefineProc(
     d_cellView
     t_fileName
     )
     = t / nil
```

Description

Changes the cellview specified by the *d_cellView* to be a parameterized cell. The specified file name should be the name of a file containing un-encrypted SKILL code defining procedure pcGenCell. The cellview should have a hierarchical property named parameters, which should be a hierarchical property list containing the default values for the parameters.

Arguments

d_cellView	The cellview ID.
t_fileName	The file which contains unencrypted SKILL code defining procedure pcGenCell.

Value Returned

t	The cellview gets changed to a parameterized cell successfully.
nil	There is a severe error in the SKILL procedure or the property parameters are illegal.

© 2020 All Rights Reserved.

dbFeaturePrintInfo

```
dbFeaturePrintInfo(
     d_cellView
     [ g_incompatibleOnly ]
     )
     => t / nil
```

Description

Lists information about the features that are being used in the specified cellview.

Arguments

d_cellView Database ID of a cellview.

g_incompatibleOnly

A Boolean value that:

- when set to `'t`, lists information about the features that are incompatible with the current version of the software. This is the default value.
- when set to `nil`, lists information about all features used in the given cellview.

Value Returned

`t`	At least one feature is being used in the cellview.
`nil`	No features are being used in the cellview.

Examples

The following example checks if the cellview `a_cellViewId` comprises features incompatible with the current version of Virtuoso,:

```
dbFeaturePrintInfo(a_cellViewId t)
```

The following example checks what features are in the cellview `a_cellViewId`:

```
dbFeaturePrintInfo(a_cellViewId nil)
```


© 2020 All Rights Reserved.

dbFindOpenCellView

```
dbFindOpenCellView(
     g_libId
     t_cellName
     t_viewName
     [ t_version ]
     )
     => d_cellViewId / nil
```

Description

Finds an opened cellview.

Arguments

g_libId	Specifies the *ddId* of the library that contains the specified cellview.
t_cellName	Cell name of the cellview.
t_viewName	View name of the cellview.
t_version	This argument is preserved for compatibility with earlier versions of the software and should be `nil` if specified.

Value Returned

d_cellViewId	The cellview ID of the cellview with the specified *libId*, cell, and view if the cellview has been opened.
`nil`	The cellview is not found open.

Example

```
cv = dbFindOpenCellView( lib "abc" "schematic" )
```

Finds the schematic cellview `abc` from the library `lib`.

© 2020 All Rights Reserved.

dbFlattenInst

```
dbFlattenInst(
     d_instId
     x_levels
     [ g_flattenPCells ]
     [ g_preservePins ]
     [ g_preserveRODobjs ]
     [ g_delDetachedBlockages ]
     [ g_preservePinFigs ]
     [ g_preserveTermName ]
     [ g_flattenVias ]
     )
     => t / nil
```

Description

Flattens instance *d_instId* up through *x_levels* of hierarchy. The `dbFlattenInst` function flattens mosaics as well as normal instances. To flatten a mosaic, pass the `mosaicId` to `dbFlattenInst`.

When you flatten a mosaic, flattening one level splits the mosaic into a set of normal instances.

Note: (ICADVM20.1 Only) In color-aware designs, effective coloring on both locked and unlocked shapes is preserved. However, if the coloring engine is enabled during flattening, unlocked shapes might be recolored after flattening is completed.

© 2020 All Rights Reserved.

Arguments

d_instId
Database ID of the instance to flatten. To flatten a mosaic, specify the `mosaicId`.

x_levels
Number of levels of hierarchy through which to flatten the instance.Valid Values: an integer between 0 and 32

g_flattenPCells
Specifies whether `pcells` should be flattened. `t` indicates that pcells should be flattened.
Valid Values: `t`, `nil`
Default: `nil`

g_preservePins
Specifies whether the pin information should be preserved when flattening. When set to `t`, dbFlattenInst preserves the connectivity information that, which includes pin properties, such as placement status, access direction and pin figures. In addition, to avoid conflicts, this function prefixes the hierarchical name to the original pin names instead of preserving the original pin names.
Valid Values: `t`, `nil`
Default: `nil`

g_preserveRODobjs
Specifies whether ROD properties should be preserved.
Valid Values: `t`, `nil`
Default: `nil`

g_delDetachedBlockages
Specifies whether the detached blockages should be deleted.
Valid Values: `t`, `nil`
Default: `nil`

g_preservePinFigs
Specifies whether the pin figures should be preserved. If set to `t`, the function preserves the pin figures.
Valid Values: `t`, `nil`
Default: `t`

g_preserveTermName
Specifies whether the terminal name should be preserved. If set to t, the function preserves the terminal name.
Valid Values: `t`, `nil`
Default: `nil`

© 2020 All Rights Reserved.

g_flattenVias	Specifies whether the vias should be flattened or not. If set to `t`, the function flattens the vias. Values: `t, nil` Default: `t`

Value Returned

`t`	The instance is flattened.
`nil`	The instance is not flattened.

Examples

```
dbFlattenInst( instance1 2 nil )
```

Flattens the instance `instance1` through two levels of the hierarchy, preserves pin figures and flattens vias (the default value is `t`). However, does not flatten any `pcells` contained in the instance.

```
dbFlattenInst(instId 1 t t nil nil t t)
```

Flattens the instance `instId` and any `pcells` contained in the instance, preserves pin information, pin figures, and terminal names.

© 2020 All Rights Reserved.

dbFlattenInst2

```
dbFlattenInst2(
     d_instId
     x_levels
     [ g_options ]
     )
     => t / nil
```

Description

Flattens an instance up through `x_levels` of hierarchy along with an optional set of flatten options. `dbFlattenInst2` flattens mosaics as well as normal instances.

Note: (ICADVM20.1 Only) In color-aware designs, effective coloring on both locked and unlocked shapes is preserved. However, if the coloring engine is enabled during flattening, unlocked shapes might be recolored after flattening is completed.

Arguments

d_instId

Specifies the database ID of the instance to flatten.

Valid value: A valid instance `db_Id`.

x_levels

Number of levels of hierarchy through which to flatten the instance.

Valid value: An integer between `0` and `32`.

g_options

(Optional) Specifies a SKILL disembodied property list with the following fields:

- `flattenPCells`: Specifies whether pcells are flattened. Valid Values: `t`, `nil`. Default: `nil`.
- `preservePins`: Specifies whether pin information is preserved. Valid Values: `t`, `nil`. Default: `nil`.
- `preserveRodObjs`: Specifies whether ROD properties are preserved. Valid Values: `t`, `nil`. Default: `nil`.
- `preserveTermName`: Specifies whether terminal name is preserved. Valid Values: `t`, `nil`. Default: `nil`.
- `delDetachedBlockages`: Specifies whether detached blockages are deleted. Valid Values: `t`, `nil`. Default: `nil`.
- `preservePinFigs`: Specifies whether pin figures are preserved. Valid Values: `t`, `nil`. Default: `t`.

© 2020 All Rights Reserved.

- `flattenVias`: Specifies whether vias are flattened. Valid Values: `t`, `nil`. Default: `t`.
- `hierSeparator`: Specifies value of hierarchy separator as a string of characters. Valid Values: String of characters. Default: none.
- `skipShapesOnReservedLayer`: When this option is specified the following shapes, if associated with a reserved layer, would be skipped while copying:
  - `oaArc`
  - `oaDonut`
  - `oaDot`
  - `oaEllipse`
  - `oaLine`
  - `oaPath`
  - `oaPathseg`
  - `oaPolygon`
  - `oaRect`
  - `oaText`
- `skipSteiners`: When specified, skips Steiners while while flattening an instance. Values: `t`, `nil`. Default: `nil`
- `skipFluidShapes`: When specified, skips fluid shapes while while flattening an instance. Values: `t`, `nil`. Default: `nil`
- `preserveConn`: Specifies whether to preserve only nets and not terminals or pins. Values: `t`, `nil`. Default: `nil`
- `useViaPurpose`: Sets the purpose of the copied shapes to the purpose of the parent via. Values: `t`, `nil`. Default: `nil`
- `mergePRBoundary`: Merges the PR Boundary of the flattened cellviews to the PR Boundary of the top level cellview. Valid Values: `t`, `nil`. Default: `nil`.

© 2020 All Rights Reserved.

- `hierNetNamesWhenPreserveConn`: Includes the instance hierarchy in the name of nets that are flattened to the top level. This works only when the `preserveConn` option is set. Valid Values: `t`, `nil`. Default: `nil`.

Value Returned

`t`	Returns `t` if the instance is succesfully flattened.
`nil`	Returns `nil` if the instance is not flattened.

Example 1

```
dbFlattenInst2( instance1 2)
```

Flattens the instance instance1 through two levels of the hierarchy. It uses default values for `g_options` argument.

Example 2

```
options = list(nil)
options->flattenPCells = t
options->preservePinFigs = nil
options->preservePins = t
options->preserveRodObjs = t
options->preserveTermName = t
options->flattenVias = nil
options->delDetachedBlockages = t
options->hierSeparator = "#"
options->skipSteiners = nil
options->skipFluidShapes = nil
options->preserveConn = t


dbFlattenInst2(instance2   3   options)
```

Flattens the instance instance2 through three levels of the hierarchy. It uses values as provided by the user for the g_options argument.

Example 3

```
options = list(nil)
options->skipSteiners = t

dbFlattenInst2(instId 2 options)
```

Flattens the instance `instId` at two levels, and a Steiner, if any, is not copied to the parent cell.

Example 4

© 2020 All Rights Reserved.

```
options = list(nil)
options->flattenPCells = t
options->useViaPurpose = t

dbFlattenInst2(instId 2 options)
```

Flattens the pcells. For vias, the purpose of the copied shapes will be set to the purpose of the parent via, if any.

Note: Vias are flattened since `flattenVias` option is true by default.

© 2020 All Rights Reserved.

dbFullLibPath

```
dbFullLibPath(
     t_libName
     )
     => t_fullLibPath / nil
```

Description

Returns the full library path of the specified library.

Arguments

d_libName	The name of the library.

Value Returned

t_fullPath	Full path for library.
nil	There is an error.

Example

```
dbFullLibPath(myLibrary ) /myDir/myLibrary
```


© 2020 All Rights Reserved.

dbFullPath

```
dbFullPath(
     d_cellView
     )
     => t_fullPath / nil
```

Description

Returns the full path of a cellview.

Arguments

d_cellView	The cellview.

Value Returned

t_fullPath	Full path for cellview *d_cellView.*
nil	There is an error.

Example

Get the full path of a cellview through `dbFullPath(cellView)` and `cellview~>fileName`, which should yield identical results.

```
dbFullPath(cellview)
cellview~>fileName
```


© 2020 All Rights Reserved.

dbGetAnyInstSwitchMaster

```
dbGetAnyInstSwitchMaster(
     d_anyInst
     t_viewList
     )
     => d_cellView / nil
```

Description

Switches into a different view of a master cell instance and opens the corresponding cellview.

The cell and library names are the same as the ones for the master of instance *d_anyInst.* If the cellview to switch into is not a parameterized cell, this function behaves the same way as opening that cellview through `dbOpenCellViewByType`. If it is a parameterized cell, the cellview returned has the same values as *d_anyInst.*

Arguments

d_anyInst	Instance of a master from which a different view is desired.
t_viewList	A string listing view names to switch into.

Value Returned

d_cellView	A different view of a master cell instance if successful.
`nil`	No other view exists or an error occurs.

Example

Switch into a schematic view from a symbol.

```
schCV = dbGetAnyInstSwitchMaster(inst "schematic")
```


© 2020 All Rights Reserved.

dbGetCellViewDdId

```
dbGetCellViewDdId(
     d_dbCellViewId
     )
     => ddId_cellViewId / nil
```

Description

Returns a *ddId* associated with the *cellViewId*. This *ddId* contains the path to the library, cell, view, and file name.

Arguments

d_dbCellViewId	A database cellview ID.

Value Returned

ddId_cellViewId	A *ddId* associated with the *cellViewId*.
nil	No cellview ID exists or if an error occurs.

Example

```
dbGetCellViewDdId( d_cvId ) => ddId
```


© 2020 All Rights Reserved.

dbGetLibDataCompressionLevel

```
dbGetLibDataCompressionLevel(
     dd_libId
     )
     => value/ nil
```

Description

Returns the data compression level set on the specified library ID.

Arguments

dd_libId	The ID of the library whose data compression level must be returned.

Value Returned

value	Returns the value of data compression level.
`nil`	Returns `nil` if an error occurs during execution.

Example

```
dbGetLibDataCompressionLevel( libId ) => 1
```


© 2020 All Rights Reserved.

dbGetOpenCellViews

```
dbGetOpenCellViews(
     )
     => (d_cv d_cv d_cv ...) / nil
```

Description

Returns all opened cellviews in virtual memory.

Arguments

None

Value Returned

d_cv d_cv ...	All opened cellviews in virtual memory.
`nil`	No cellview is opened.

© 2020 All Rights Reserved.

dbHasAutoSavedFile

```
dbHasAutoSavedFile(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Determines whether the specified cellview has an auto-saved file.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cellview.
t_viewName	Name of a view.

Value Returned

`t`	The auto-save file is present in the specified cellview.
`nil`	The auto-save file is not present in the specified cellview.

Example

```
;Check if auto-save file is present in cellview testing1
dbHasAutoSavedFile("designLib" "testing1" "layout1")
t
; Open the auto-save file
CV1 = dbOpenAutoSavedCellView("designLib" "testing1" "layout1")
db:0x08a1ec92
```


© 2020 All Rights Reserved.

dbHasPanicFile

```
dbHasPanicFile(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Determines whether the specified cellview has a panic file.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cellview.
t_viewName	Name of a view.

Value Returned

`t`	Panic file is present in the specified cellview.
`nil`	Panic file is not present in the specified cellview.

Example

```
;Check if panic file is present in cellview testing1
dbHasPanicFile("designLib" "testing1" "layout1")
t
;Open the panic file.
panicCV1 = dbOpenPanicCellView("designLib" "testing1" "layout1")
db:0x08a1ec92
```


© 2020 All Rights Reserved.

dbIsNonSKILLPcell

```
dbIsNonSKILLPcell(
     d_cellviewId
     )
     => t / nil
```

Description

Checks whether a pcell is not a SKILL pcell.

Arguments

d_cellviewId	Database ID of a cellview.

Value Returned

`t`	The pcell is not a SKILL pcell.
`nil`	The pcell is a SKILL pcell.

Example

```
dbIsNonSKILLPcell(cvId)=>nil
```


© 2020 All Rights Reserved.

dbOpenAutoSavedCellView

```
dbOpenAutoSavedCellView(
     t_libName
     t_cellName
     lt_viewName
     [ t_version ]
     )
     => d_cellView / nil
```

Description

Reads the auto-saved file.

The auto-save file cannot be opened if the cellview is open. This file can only be opened for reading. If you want to save the data, use `dbReopen` to change to edit mode to save the file back to disk. The auto-save file is the cellview file name appended by a plus (+) sign.

Arguments

t_libName	Library name.
t_cellName	Cell name.
lt_viewName	One or more view names in one of two forms: *l_viewName* is a SKILL list of view names, such as `list("viewA" "viewB")` *t_viewName* is a string containing one or more view names separated by spaces, such as "*viewA viewB*". **Note:** Cellviews of the cell *t_cellName* are searched for a name matching one from the list. If a match is found, that cellview is opened.
t_version	This argument is preserved for compatibility with earlier versions of the software and should be `nil` if specified.

© 2020 All Rights Reserved.

Value Returned

d_cellView	The *dbObject* of the open cellview.
nil	An error is detected.

Example

Open the auto-saved file for the `layout` view of `cellA` from the `test` library.

```
cellview = dbOpenAutoSavedCellView("test" "cellA" "layout")
```


© 2020 All Rights Reserved.

dbOpenCellViewByType

```
dbOpenCellViewByType(
     { gt_lib | nil }
     t_cellName
     lt_viewName
     [ t_viewTypeName
     [ t_mode
     [ d_contextCellView ] ] ]
     )
     => d_cellView / nil
```

Description

Opens a cellview.

The cellview can be opened for `read`, `append`, `write`, or `scratch` mode. If you open the cellview for the *read* mode, the cellview must exist. if you open the cellview for other modes, the cellview is created if it does not exist.

Arguments

gt_lib | nil	Specifies the *ddId* of a library, the name of a library *t_lib*, or `nil`. If it is `nil`, then a search is made for the library of context cellview *d_contextCellView*. If the name of a library is given, the library must exist in the *libList*.
t_cellName	Specifies the cell name.
lt_viewName	Specifies one or more view names in one of the following two forms: ■ In the *l_viewName* form It is a SKILL list that has *t_viewNames*, such as `list("viewA" "viewB")`. ■ In the *t_viewName* form It is a string containing one or more view names separated by spaces, such as `"viewA viewB"`. Cellviews of cell *t_cellName* are searched for a name matching one from the list. If a match is found, that cellview is opened.

© 2020 All Rights Reserved.

t_viewTypeName

Specifies the view type. If it is omitted or entered as `nil` or `"*.cdb"`, the cellview must exist.

This argument supports different types of cellviews, some of which are specific to certain flows. Some of the commonly used cellViewTypes are: `"maskLayout"`, `"schematic"`, `"schematicSymbol"`, and `"netlist"`.

Note: Schematics use cellViewType `"schematic"and` schematic symbols use cellViewType `"schematicSymbol"`.

You need to specify a cellViewType to create a new cellview. An error occurs if you try to create a cellview with an unsupported type.

t_mode

Specifies the access mode to the cellview. The mode can be one of the following values:

- `r`
  Opens the cellview in read mode. The cellview must already exist.
- `a`
  Opens the existing cellview in append mode. If the cellview does not exist, it is created.
- `w`
  Opens the cellview in write mode. If the cellview does not exist, it is created. If the cellview already exists, its contents are removed from the virtual memory.
- `s`
  Opens the cellview in scratch mode. If the cellview does not exist, this mode is the same as "w" mode , which means that the cellview will have the same state and content as a cellview opened in 'w' mode. However, the cellview cannot be saved to disk.

© 2020 All Rights Reserved.

- `wc`
  Same as "`w`", except that the cellview is not created as the master. This mode creates the cellview even if there is already a master under the cell and view.

  This option is useful in scenarios where users create cellviews that do not have Virtuoso database files as the master for the new cellviews.

  For example, consider that a user creates a cellview using a DFII-based simulator and the cellview has a text-based netlist file as its master.

  To create a Virtuoso representation of this cellview, you might want to create a database file in the same cellview directory but leave the netlist as the master or source of the cellview data.

  In this scenario, the user can initially create the cellview with the simulator and then later, create the same cellview in Virtuoso to represent the netlist.

- `wd`
  Same as `"wc"`, except that the context cellview must be specified. *`LibName`* is ignored if specified. The library is derived from the context cellview.

- `ac`
  Same as "`a`" if the cellview already exists. If the cellview does not exist, this mode is the same as "`wc`" mode.

- `ad`
  Same as "`a`" if the cellview already exists. If the cellview does not exist, this mode is the same as "`wd`" mode.

- `sc`
  Same as "`s`" if the cellview already exists. If the cellview does not exist, this mode is the same is "`wc`" mode, which means that the cellview is in the same state and with the same contents. However, the cellview cannot be saved to disk.

© 2020 All Rights Reserved.

- `sd`
  Same as "`s`" if the cellview already exists. If the cellview does not exist, this mode is the same as "`wd`" mode, except the cellview cannot be saved to disk.

`d_contextCellView` Specifies the context cellview. The context cellview specification is preserved for compatibility with earlier versions of the software.

The specified libraries are searched for the cellview, according to the following search rules

- If both *`gt_lib`* and *`d_contextCellView`* are specified, library *`gt_lib`* is ignored. The library where the context cellview resides is searched.
- If only *`gt_lib`* is specified, only that library is searched. The named library must exist in the *`libList.`*
- If only *`d_contextCellView`* is specified, only the library of the context cellview is searched.

Value Returned

`d_cellView`	The *`dbObject`* of the cellview opened.
`nil`	Any error is detected.

Examples

If cell "cellA" with view "layout" exists in library "test," open the cellview for read.

```
cellview = dbOpenCellViewByType("test" "cellA" "layout")
```

Open cell "cellA" with view "layout" in library "test" for "append" mode. Create the cellview if it does not exist.

```
cellview = dbOpenCellViewByType("test" "cellA" "layout" "maskLayout" "a")
```

Open cell "cellA" with view "layout" in library "test" for "append" mode only if the cellview already exists.

```
cellview = dbOpenCellViewByType("test" "cellA" "layout" "" "a")
```

Open cell "cellA" with view "layout" in library "test" for "scratch" mode. Create the cellview if it does not exist.

```
cellview = dbOpenCellViewByType("test" "cellA" "layout" "maskLayout" "s")
```


© 2020 All Rights Reserved.

dbOpenHier

```
dbOpenHier(
     d_cellView
     x_numLevels
     )
     => t
```

Description

Opens and binds masters of instances that constitute a design hierarchy.

The starting point is the cellview specified by *d_cellView*. If *x_numLevels* is 0 (zero), no action is taken and *t* is returned. If *x_numLevels* is 1, masters of instances contained in cellview *d_cellView* are opened and bound. If *x_numLevels* is 2, masters of instances contained in each of the masters in level 1 are opened and bound. And so on for each level specified.

Arguments

d_cellView	Cellview that represents level 0 (zero) of a design hierarchy.
x_numLevels	Number of design hierarchy levels to open.

Value Returned

`t`	Always returns `t` and reports errors as encountered.

Example

`dbOpenCellViewByType` does not automatically open and bind the masters.

```
cellview = dbOpenCellViewByType("design_gpdk090" "ADDFX2" "layout")
```

Opens the design hierarchy up to and including level one, which means open and bind the master of instances contained in the given cellview.

```
dbOpenHier( cellview 1 )
```

Note: `dbOpenHier()` does not switch views. It only opens views that are directly instantiated.

© 2020 All Rights Reserved.

dbOpenPanicCellView

```
dbOpenPanicCellView(
     gt_lib
     t_cellName
     t_viewName
     [ t_version
     [ t_mode
     [ d_contextCellView ] ] ]
     )
     => d_cellView / nil
```

Description

Opens a saved panic cellview from a physical panic file with extension `.oa-`. The cellview `.oa-` opens in read mode only. You can then use the `dbSave()` function to save it to a new cellview, or call the `dbReopen()` function to reopen it in the append mode and then call the `dbSave()` function to save the cellview back to disk.

Arguments

gt_lib	Specifies the *ddId* of a library, the name of a library *t_lib*. Specifies which library to search the cellview in. It must exist in the `cds.lib` file.
t_cellName	Specifies the cell name.
t_viewName	Specifies the view name.
t_version	This argument is preserved for compatibility with earlier versions of the software and should be `nil` if specified.
t_mode	This argument is preserved for compatibility with earlier versions of the software and should be `r` if specified. The cellview opened by this function is always in "read" mode.
d_contextCellView	This argument is preserved for compatibility with earlier versions of the software and its content is ignored.

Value Returned

d_cellView	The `dbObject` of the cellview opened.
`nil`	An error is detected.

© 2020 All Rights Reserved.

Example

If panic cellview exists in the "layout" view of "cellA" in library "test," open the cellview for read.

```
cellview = dbOpenPanicCellView("test" "cellA" "layout")
dbSave(cellview "test" "cellA" "layout_panic")
dbReopen(cellview "a")
dbSave(cellview)
```


© 2020 All Rights Reserved.

dbOpenParamCellView

```
dbOpenParamCellView(
     t_libName
     t_cellName
     t_viewName
     [ l_params ]
     )
     => d_cellView / nil
```

Description

Opens submaster cellview of the Pcell supermaster specified by the library, cell, and view names and the parameter values.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cellview.
t_viewName	Name of a view.
l_params	List of *l_param,* where *l_param* is *list(t_propName t_propType g_value*). If parameter values are not specified, the default values are used.

Value Returned

d_cellView	Database ID of the cellview that is opened.
nil	The operation fails.

Example

Opens submaster cellview of the Pcell supermaster in the `symbol` view of `nand` cell in the library `basic` with property name `w` (width), property type `float`, and value `2.0`.

```
dbOpenParamCellView("basic" "nand" "symbol" list(list("w" "float" 2.0) list("l"
"float" 5.0) t))
```


© 2020 All Rights Reserved.

dbProduceMemName

```
dbProduceMemName(
     t_name
     )
     => list( <memName> ... <memName> ) / nil
```

Description

Returns a list of the member names for the given name.

Arguments

t_name	Name whose members will be generated.

Value Returned

`list`	Returns list of the member names.
<memName>	Member names.
`nil`	There are errors.

Example

```
dbProduceMemName("7")
 ("7")

dbProduceMemName("(X)")
 ("X")

dbProduceMemName("(7)")
 ("7")

dbProduceMemName("()")
 nil

dbProduceMemName("x(a)")
 ("x#28a#29")

dbProduceMemName("x(3)")
```


© 2020 All Rights Reserved.

```
 ("x(3)")

dbProduceMemName("(a)x")
 nil

dbProduceMemName("(2)x")
 nil

dbProduceMemName("<*3>(0)")
 ("0" "0" "0")

dbProduceMemName("<*3>(A)")
 ("A" "A" "A")

dbProduceMemName("a(7),x(1),2(1),(5),(y)")
 ("a(7)" "x(1)" "2(1)" "5" "y")

dbProduceMemName("c<0*2,(0)*3>")
 ("c<0>" "c<0>" "c<0>" "c<0>" "c<0>")

dbProduceMemName("A<2:0>,B,C<3:4>")
=> ("A<2>" "A<1>" "A<0>" "B" "C<3>" "C<4>")
```


© 2020 All Rights Reserved.

dbPurge

```
dbPurge(
     d_cellView
     )
     => t / nil
```

Description

Forces a cellview to close and removes from virtual memory.

The `dbPurge` function should be used carefully because all changes made, but not saved, are lost. Furthermore, in the graphics environment, purging a cellview empties any window in which the cellview is being displayed.

Note: When you use `dbPurge`, undo checkpoints for all edit actions before the purge operation are cleared. To avoid losing the undo checkpoint information when purging a cellview, set the `dbUndoAcrossPurge` cdsenv variable value to `t`.

Arguments

d_cellView	Cellview to purge.

Value Returned

`t`	The cellview is closed.
`nil`	The cellview is not closed.

Example

Purge this cellview.

```
dbPurge(cv)
```


© 2020 All Rights Reserved.

dbRefreshCellView

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

Description

Refreshes the cellview in virtual memory with the most current copy from disk. If there were any changes to the cellview, this function purges the cellview from virtual memory and reopens the cellview.

Arguments

d_cellViewId	The database ID of the cellview.

Value Returned

t	The cellview was refreshed.
nil	The cellview was not refreshed.

© 2020 All Rights Reserved.

dbRefreshLibDataCompressionLevel

```
dbRefreshLibDataCompressionLevel(
     dd_libId
     )
     => t / nil
```

Description

Refreshes the data compression level for the specified library ID from the disk.

Refreshing the compression level of a library only affects the designs that are saved to the library after the refresh has occurred.

Note: A translator running in another session can change the compression level of a library that is in use in the Virtuoso design environment. If Virtuoso does not refresh the data compression level from the disk, it will not be aware of the change made by the translator.

Arguments

dd_libId	The ID of the library whose data compression level must be refreshed.

Value Returned

`t`	Returns `t` if the data compression level of the specified library ID is successfully refreshed.
`nil`	Returns `nil` if the data compression level of the specified library ID could not be refreshed.

Example

```
dbRefreshLibDataCompressionLevel( dd_libId ) => t
```


© 2020 All Rights Reserved.

dbRegPostSaveAsTrigger

```
dbRegPostSaveAsTrigger(
     S_triggerFunc
     [ x_priority ]
     )
     => t / nil
```

Description

Allows users to register a trigger function that is called after a postSaveAs or Save Copy operation.

Arguments

S_triggerFunc	User-defined trigger function.
x_priority	Priority of the trigger functions registered. This argument is optional. The default is the lowest priority. The priority argument enables users to control which trigger function is to be launched first. It should be specified as a positive number, and `0` is the highest priority.

Value Returned

`t`	The trigger function gets registered successfully.
`nil`	The registration fails.

Example

```
dbRegPostSaveAsTrigger("myPostSaveAsTrigger1" 0)
```


© 2020 All Rights Reserved.

dbRegPostSaveTrigger

```
dbRegPostSaveTrigger(
     S_triggerFunc
     [ x_priority ]
     )
     => t / nil
```

Description

Allows users to register a trigger function that is to be called after `dbSave`. If `dbSave` fails, this trigger will not be launched. This trigger will be called only after saving the db cellviews. The post save trigger will be called with the db cellview ID.

Note: The trigger function *must not* call `dbSave` again or it will cause infinite looping.

Arguments

S_triggerFunc	User-defined trigger function.
x_priority	Priority of the trigger functions registered. This argument is optional. The default is the lowest priority. The priority argument enables users to control which trigger function is to be launched first. It should be specified as a positive number, and 0 is the highest priority.

Note: The trigger function can be defined after calling `dbRegPostSaveTrigger`, but, in that case, you will get a warning message.

Value Returned

`t`	The trigger function gets registered successfully.
`nil`	The registration fails.

© 2020 All Rights Reserved.

dbRegPurgeTrigger

```
dbRegPurgeTrigger(
     S_triggerFunc
     [ x_priority ]
     )
     => t / nil
```

Description

Registers a user-defined trigger function.

The trigger functions registered are invoked when a cellview is purged.

Arguments

S_triggerFunc	User-defined trigger function.
x_priority	Priority of the trigger functions registered. This argument is optional. The default is the lowest priority.

Value Returned

`t`	The trigger function gets registered successfully.
`nil`	The registration fails.

Example

```
procedure( myTrigFunc(cv) printf("cellview - %d\n", cv) )
dbRegPurgeTrigger('myTrigFunc 10)
```


© 2020 All Rights Reserved.

dbRegSaveTrigger

```
dbRegSaveTrigger(
     S_triggerFunc
     [ x_priority ]
     )
     => t / nil
```

Description

Registers a SKILL preSave trigger. The trigger function does not need to exist at the time of register, a warning will be issued though. If the function is bound, it is called before the cellview is saved.

Arguments

S_triggerFunc	SKILL trigger function. The function must be defined as: *funcName*(*d_cellviewId*)
x_priority	Optional priority. Defaults to `INT_MAX` (lowest priority).

Value Returned

`t`	The trigger was successfully registered.
`nil`	An error occurred during execution.

© 2020 All Rights Reserved.

dbReopen

```
dbReopen(
     d_cellViewId
     t_mode
     )
     => t / nil
```

Description

Changes the mode of the specified cellview to the given mode.

Arguments

d_cellViewId	A cellview ID.
t_mode	A character representing the new mode. Valid values: `r` (read), `a` (append), `w` (write), or `s` (scratch). ■ `r` (read) - Reopens the cellview in read mode. When you use the function to change from edit to read-only mode, your changes are discarded immediately. ■ `a` (append) - Reopens the cellview in append mode. When you use the function to change from read-only mode or scratch mode to append mode, a conditional refresh might be performed. For example, if there are timestamp changes on the disk file, a refresh action is performed. Otherwise, the cellview data is kept as it is. ■ `w` (write) - Reopens the cellview in write mode, in which the contents of the cellview are truncated. ■ `s` (scratch) - Reopens the cellview in scratch mode.

© 2020 All Rights Reserved.

Value Returned

`t`	The mode change is successful.
`nil`	The mode change is not successful.

Example

```
cv = dbOpenCellViewByType("lib" "decoder" "schematic" "" "r")
```

Re-open the cellview for append mode.

```
dbReopen(cv "a")
cv~>mode
=> "a"
```

Re-open the cellview for scratch mode.

```
dbReopen(cv "s")
cv~>mode
=> "s"
```


© 2020 All Rights Reserved.

dbRestoreAndOpenAutoSavedFile

```
dbRestoreAndOpenAutoSavedFile(
     t_libName
     t_cellName
     t_viewName
     )
     => d_cellviewID / nil
```

Description

Restores and opens the auto-saved file for the specified cellview.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cellview.
t_viewName	Name of a view.

Value Returned

d_cellviewID	Database ID of the cellview that opens.
nil	Returned in case of failure.

Example

```
; Check if cellview has an auto-save file
dbHasAutoSavedFile("designLib" "testing1" "layout1")
t
;Restore and open the auto-save file
CV2 = dbRestoreAndOpenAutoSavedFile("designLib" "testing1" "layout1")
db:0x08a1ec92
```


© 2020 All Rights Reserved.

dbRestoreAndOpenPanicFile

```
dbRestoreAndOpenPanicFile(
     t_libName
     t_cellName
     t_viewName
     )
     => d_cellviewID / nil
```

Description

Restores and opens the panic file for the specified cellview.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cellview.
t_viewName	Name of a view.

Value Returned

d_cellviewID	Database ID of the cellview that opens.
nil	Returned in case of failure.

Example

```
; Check if cellview has a panic file
dbHasPanicFile("designLib" "testing1" "layout1")
t
;Restore and open the panic file
panicCV2 = dbRestoreAndOpenPanicFile("designLib" "testing1" "layout1")
db:0x08a1ec92
```


© 2020 All Rights Reserved.

dbSave

```
dbSave(
     d_cellview
     [ d_destCellView | t_libName t_cellName t_viewName ]
     )
     => t / nil
```

Description

Saves the changes made to a specified cellview opened in write or append mode. By default the cellview is saved to the same database ID; however, you can optionally specify a different destination cellview ID or create a new cellview in a different library. A cellview opened in read mode can only be saved as a new cellview with the different name.

The specification for the destination cellview is identical to `dbOpenCellViewByType`, but `mode` is omitted. A cellview that has been opened in read mode cannot be saved or overwritten.

Note: When you use `dbSave`, undo checkpoints for all edit actions before the save operation are cleared. To avoid losing the undo checkpoint information while saving a cellview, set the `dbUndoAcrossSave` cdsenv variable value to `t`.

Arguments

d_cellView	The cellview to be saved.
d_destCellView	Specifies a cellview to save the data into. In the first form, this destination cellview is specified by a *dbObject.*
t_libname	The library name.
t_cellName	Specifies the cell name.
t_viewName	Specifies one or more view names in the *l_viewName* or *t_cellName* form.

© 2020 All Rights Reserved.

Value Returned

`t`	The cellview is saved.
`nil`	The cellview is not saved.

Examples

Overwrites the specified cellview.

```
dbSave(cellview)
```

Saves the changes to an open destination cellview.

```
destCV = dbOpenCellViewByType(lib "cellName" "schematic"
"schematic" "w")
dbSave(cellview destCV)
```

Saves the changes to a destination cellview in a different library.

```
dbSave(cellview "libName" "cellName" "viewName")
```


© 2020 All Rights Reserved.

dbSearchCDF

```
dbSearchCDF(
     g_status
     )
     => t
```

Description

Sets and un-sets a flag that will cause CDF to be searched for parameter values.

Arguments

g_status	Sets the flag. Valid Values: `t` (sets), `nil` (un-sets the flag)

Value Returned

`t`	The operation was successful.

© 2020 All Rights Reserved.

dbSetAutoSave

```
dbSetAutoSave(
     g_isAutoSaveSet
     [ u_saveInterval ]
     )
     => t / nil
```

Description

Enables or disables the automatic save feature.

With the automatic save feature enabled, all cellviews that have been modified and not saved are auto-saved after the specified time interval has passed. Auto-saved cellviews have the `.oa+` file extension.

To minimize the impact on Virtuoso session performance, auto-save checking is done on a user-event such as clicking on a menu item or typing something into the CIW. This means that if you make an edit to a layout cellview and check the UNIX file time stamp of the corresponding layout `.oa+` file, it will not be updated. The file time stamp will change only after a user-event.

Arguments

g_isAutoSaveSet	Specifies whether or not to set the automatic save feature. The default value is `nil`, which means the feature is not set; `t` sets the save feature.
u_saveInterval	Specifies the time interval in seconds after which the cell is saved automatically. The default time interval is 300 seconds or five minutes. **Note:** The interpretation of the `u_saveInterval` changed in the IC6.1.8 and ICADVM20.1 base release. Now, this value is interpreted as a time interval in seconds.

© 2020 All Rights Reserved.

Value Returned

`t`	The automatic save feature has been set successfully.
`nil`	The automatic save feature has not been set successfully.

Examples

Disable the automatic save option.

```
dbSetAutoSave(nil)
=> t
```

Set the automatic save option so that data is saved automatically after every 100 seconds.

```
dbSetAutoSave(t 100)
=> t
```


© 2020 All Rights Reserved.

dbSetAutoSaveCallback

```
dbSetAutoSaveCallback(
     t_callback
     )
     => 't / nil
```

Description

Enables a user-defined callback function to control the frequency at which large designs are auto-saved. When set, the user-defined callback function is called just before the design is about to be auto-saved.

Note: The user-defined function can be registered only once, usually in the start-up script before calling the dbSetAutoSave function.

Arguments

t_callback	Name of the callback function.

Value Returned

`'t`	The user-defined callback function is set.
`nil`	The user-defined callback function is not set and returns an error when: ■ A user-defined callback function is already set. ■ No user-defined callback function is found. ■ The user-defined callback function is incorrectly defined.

Example

In this example, the `as_func` callback function is defined. This function returns `t` every time it is invoked. Next, the auto-save feature is enabled by registering this callback function.

```
as_var = 0

 (defun as_func ( design )

   ( let (res remnt)
```


© 2020 All Rights Reserved.

```
    as_var = as_var + 1

    res = nil

    remnt = as_var - (as_var / 2 * 2)

    (if (remnt == 0) res = t)

    res))

dbSetAutoSave(t)

 dbSetAutoSaveCallback("as_func")
```

Here, the auto-save period of a single design increases by factor a of 2.

Note: You should examine the design size and configure a delay in auto-save only for those designs that larger then several megabytes.

© 2020 All Rights Reserved.

dbSetLibDataCompressionLevel

```
dbSetLibDataCompressionLevel(
     dd_libId
     [ n_compressionLevel ]
     )
     => t/ nil
```

Description

Sets the data compression level for designs that are saved to the specified library ID.

A compression level indicates the amount of effort the compression algorithm uses when compressing data. A higher level does not necessarily correspond to better compression efficiency. The supported levels range from `1` to `9`. The default level `1` provides an optimized balance between efficiency and performance.

Note: Changing the compression level of the library only affects the designs that are saved to the library after the compression level is set. It does not recompress the existing designs in the library.

Arguments

dd_libId	The ID of the library whose data compression level must be set.
n_compressionLevel	The data compression level that must be set on the specified library ID. The compression level must be an integer between `1` and `9` (inclusive). If this optional argument is not specified, the OpenAccess default compression level is used.

Value Returned

`t`	Returns `t` if the data compression level is successfully set on the specified library ID.
`nil`	Returns `nil` if the data compression level could not be set.

Example

```
dbSetLibDataCompressionLevel( dd_libId 1 ) => t
```


© 2020 All Rights Reserved.

dbUndefineProc

```
dbUndefineProc(
     d_cellView
     )
     = t / nil
```

Description

Changes the parameterized cell specified by the *d_cellView* to be a regular cellview.

Arguments

d_cellView	Specifies the cellview ID.

Value Returned

t	The parameterized cell gets changed to a regular cell successfully.
nil	Errors are encountered.

© 2020 All Rights Reserved.

dbUnregPostSaveAsTrigger

```
dbUnregPostSaveAsTrigger(
     S_triggerFunc
     )
     => t / nil
```

Description

Allows users to unregister the trigger function called after a postSaveAs operation

Arguments

S_triggerFunc	User-defined trigger function that you want to unregister

Value Returned

t	The trigger function is successfully unregistered
nil	The trigger function is not unregistered

Example

```
dbUnregPostSaveAsTrigger("myPostSaveAsTrigger1")
```


© 2020 All Rights Reserved.

dbUnregPostSaveTrigger

```
dbUnregPostSaveTrigger(
     s_triggerFunc
     )
     => t / nil
```

Description

Un-registers the post save trigger function from `dbSave`.

Arguments

`s_triggerFunc`	User-defined trigger function.

Value Returned

`t`	The function is un-registered.
`nil`	The function is not un-registered.

© 2020 All Rights Reserved.

dbUnregPurgeTrigger

```
dbUnregPurgeTrigger(
     s_triggerFunc
     )
     => t / nil
```

Description

Unregisters a user-defined trigger function.

Arguments

s_triggerFunc	User-defined trigger function.

Value Returned

`t`	The trigger function gets unregistered successfully.
`nil`	The un-registration fails.

Example

```
dbUnregPurgeTrigger('myTrigFunc)
```


© 2020 All Rights Reserved.

dbUnregSaveTrigger

```
dbUnregSaveTrigger(
     S_triggerFunc
     )
     => t / nil
```

Description

This function unregisters the SKILL preSave trigger function specified by the given name.

Arguments

S_triggerFunc	SKILL trigger function. The function must be defined as: *funcName*(*d_cellviewId*)

Value Returned

t	The trigger was successfully unregistered.
nil	An error occurred during execution.

© 2020 All Rights Reserved.

dbUnsetLibDataCompressionLevel

```
dbUnsetLibDataCompressionLevel(
     dd_libId
     )
     => t/ nil
```

Description

Removes the data compression level of the specified library.

After the data compression level is removed, the data of the designs saved in that library will no longer be compressed. However, this removal does not change the compression level of the cellviews that were already saved in the library. To uncompress all the cellviews in a library, use the oazip utility.

Arguments

dd_libId	The ID of the library whose data compression level must be removed.

Value Returned

`t`	Returns `t` if the data compression level is successfully removed for the specified library ID.
`nil`	Returns `nil` if the data compression level could not be removed.

Example

```
dbUnsetLibDataCompressionLevel( dd_libId ) => t
```


© 2020 All Rights Reserved.

dbWriteSkill

```
dbWriteSkill(
     d_cellViewId
     t_fileName
     t_mode
     t_release
     [ g_conn ] db
     [ g_ref ]
     [ g_lppString ]
     )
     => t / nil
```

Description

Creates a file that contains all the SKILL commands needed to recreate a cellview, so that the design can be recreated in the current or previous release by loading the file in the current or previous release, respectively.

To avoid painfully traversing the customer design hierarchy, the *g_ref* option allows you to traverse the cellview hierarchy from a source cellview and to generate the SKILL functions to recreate all the database objects of all the reference cellviews which the source cellview directly and indirectly referenced.

`dbWriteSkill` creates two files for a procedural pcell—one with the cell's design data and the other with the procedure.

Note: Functions `dbWriteSkill` and dbWriteSkillWithLib do not replicate all database attributes of all objects, and should not be viewed as a way to completely recreate a given design. In previous releases, these functions served purposes related to migration of basic objects from older releases to newer versions, but are not intended to fully support newer and expanded capabilities and methods used by the latest versions, and latest features in the OA database. For example, a partial list of limitations include, layer coloring information, blockages, and physical-only attributes.

Arguments

d_cellViewId	ID of the source cellview.
t_fileName	Target SKILL filename.
t_mode	Modes.
	Valid Values: `w` (overwrite), `a` (append)
	Using `a` mode, you can dump a few designs into one file.

© 2020 All Rights Reserved.

t_release	The release in which the file will be loaded.
g_conn	The default is to dump out everything. If this argument is set to `TRUE`, then only the connectivity of the given cellview will be written out.
g_ref	The default is to not dump the reference cellviews. If this argument is set to `t`, then all the cellviews that the source cellview directly or indirectly referenced will be written out.
g_lppString	Layer purpose name or number. The default is to dump `lpp` as number (for backward compatibility.) If set to `t`, `lpp` is dumped as a string.

Value Returned

`t`	The function was successful.
`nil`	The function failed.

Example 1

In the 5.0 release, do

```
dbWriteSkill(cvId "/tmp/bar.il" "w" "5.0")
```

If you are dumping with the reference cellview option (g_ref), a SKILL procedure is defined for each cellview to modularize all the database objects of each cellview. Because the number of database objects is unlimited but the line number of a SKILL procedure is limited, split the procedure if the line number in each procedure is more than PROCMAXLINE (that is, 2000 lines). The procedure name is based on the ID of the cellview in memory. Append the numbers starting from 0 when splitting multiple procedures.

For example:

```
procedure( dbD_10056748Ref0() ... )
procedure( dbD_10055724Ref0() ... )
procedure( dbD_10055724Ref1() ... )
procedure( dbD_10055724Ref2() ... )
procedure( dbD_10055724Ref3() ... )
```

Note: The sequence of recreating the cellviews in the hierarchy is important. Create the leaf node cellviews first, then go up to the source cellview. This ensures that when any database object that references another cellview is created, it will not fail.

For example, create cellview A with the following reference cellviews:

© 2020 All Rights Reserved.

```
              A
            /     \
          B        C
         / \      / \
       D    E   /    F
      /    / \/
   G     H   I
```

In this example, the sequence of creating cellviews is G, D, H, I, E, B, F, C, and A.

Limitation

There is a size limitation with the *g_ref* option when you load a huge SKILL file that is generated from a big design. In this case, avoid dumping from the top level cellview, and try to dump from lower level cellviews.

Example 2

Given below is an example for a procedural `pCell`.

```
pcell cvId = dbOpenCellViewByType("myLib" "myPcell" "layout")
dbWriteSkill(cvId "/tmp/dumpFile.il" "w" "5.0")
```

creates two files in `/tmp`:

```
dumpFile.il
myLib_myPcell_layout_pCell.il
```

`dumpFile.il` contains design data for the cell.

`myLib_myPcell_layout_pCell.il` contains the `pCell` procedure.

Follow these steps, substituting your names for the example names:

1. Execute `dbWriteSkill`.
2. Run the SKILL Code Checker on `dumpFile.il`.
3. Make the necessary changes to `myLib_myPcell_layout_pCell.il`.
4. Recreate the `pCell`.

Note: Do not use the load command or the pcell compiler to re-create the cell. You lose graphical data if you use load and you lose parameters if you recompile. For information about loading pcells, refer to the *Virtuoso Parameterized Cell Reference*. For information about loading techfile devices, refer to the *Virtuoso Technology Data User Guide.*

© 2020 All Rights Reserved.

dbWriteSkillWithLib

```
dbWriteSkillWithLib(
     d_cellviewId
     t_fileName
     t_mode
     t_release
     [ g_conn ]
     [ g_ref ]
     )
     => t / nil
```

Description

Creates a file that contains all the SKILL commands needed to recreate a cellview.

The difference between `dbWriteSkill` and `dbWriteSkillWithLib` is that library name is not written in the SKILL code when using `dbWriteSkillWithLib`. Instead a SKILL variable, *`dbD_NewLibNameForDump`*, is written whose value can be used to set the name of the target library before loading the SKILL file.

Note: Functions dbWriteSkill and dbWriteSkillWithLib do not replicate all database attributes of all objects, and should not be viewed as a way to completely recreate a given design. In previous releases, these functions served purposes related to migration of basic objects from older releases to newer versions, but are not intended to fully support newer and expanded capabilities and methods used by the latest versions, and latest features in the OA database. For example, a partial list of limitations include, layer coloring information, blockages, and physical-only attributes.

© 2020 All Rights Reserved.

Arguments

d_topCellViewId	The top design cellview ID.
t_fileName	Target SKILL filename.
t_mode	Modes. Valid Values: `w` (overwrite) or `a` (append) Using `a` mode, you can dump a few designs into one file.
t_release	The release in which the file will be loaded.
g_conn	The default is to dump out everything. If this argument is set to `TRUE`, then only the connectivity of the given cellview will be written out.
g_ref	The default is to not dump the reference cellviews. If this argument is set to `t`, then all the cellviews that the source cellview directly or indirectly referenced will be written out.

Value Returned

`t`	The file was created.
`nil`	The file was not created.

Example

```
cvId2 = dbOpenCellViewByType("dummyLib" "cell2" "layout" nil "r")

dbWriteSkillWithLib( cvId2 "z4.il" "w" "6.1")
```

Creates the file `z4.il` that contains the SKILL commands to recreate the `cell2/layout` cellview in a library given by the SKILL variable `dbD_NewLibNameForDump`.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

4

Instance Creation and Retrieval Database Access Functions

The following functions create and retrieve instances and their attributes from the database.

- dbCreateInst
- dbCreateParamInst
- dbCreateExtParamInst
- dbCreateExtParamInstByMasterName
- dbCreateXformPCell
- dbDumpPcDefinePcell
- dbCheckParamCell
- dbEvalParamCell
- dbFixParamCell
- dbHasInstance
- dbIsCellViewModified
- dbReplaceParam
- dbReplaceInstParamList
- dbFindAnyInstByName
- dbFindMemInstByName
- dbGetHierPathTransform
- dbGetInstByName
- dbGetInstanceByName
- dbGetInstNameNumInst

© 2020 All Rights Reserved.

- dbGetInstTransform
- dbIsAnyInstBaseNameUsed
- dbCreateInstByMasterName
- dbCreateParamInstByMasterName
- dbDumpPcell
- dbDumpPcellWithLib
- dbCreateViaShapeArray
- dbSetInstHeaderMasterName
- dbIsSameMaster
- dbIsObjectBound
- dbGetCellViewSymmetry
- dbSetCellViewSymmetry
- dbGetCellViewEEQMaster
- dbSetCellViewEEQMaster
- dbIsInstTransparent
- dbSetInstTransparent

© 2020 All Rights Reserved.

dbCreateInst

```
dbCreateInst(
     d_cellView
     d_master
     t_name
     l_point
     t_orient
     [ x_stopIndex ]
     [ g_physOnly]
     )
     => d_inst / nil
```

Description

Creates an instance (of cellview *d_master*) in the cellview *d_cellView* with an origin and orientation specified by *l_point* and *t_orient*.

© 2020 All Rights Reserved.

Arguments

d_cellView	Cellview where the instance is placed.
d_master	Master cellview of the instance to be created. This parameter must be a super master; use of a submaster can cause unexpected behavior.
t_name	If specified, the instance is named by *t_name*; if `nil` the program assigns a unique name to the instance. *t_name* can be a unique name, such as "`Inst1`" or, a hierarchy name, such as `"moduleA`moduleB`Inst"` with at least one hierarchy delimiter (`). It must obey the syntax of instance names as defined at the beginning of this chapter.
l_point	Origin of the new instance.
t_orient	Orientation of the new instance.
x_stopIndex	Number of instances; When specified and when its value is greater than `0`, the starting index value is `0`, and therefore, the resulting width of the instance is `stopIndex +1`. Default value: `'nil` **Note:** If `t_name` is given, any value specified by *x_stopIndex* is overridden. (that is, the number of instances is derived from the name, instead of being taken from the *x_stopIndex* argument).
g_physOnly	Optional boolean argument that specifies whether cellview data is physical or both physical and logical. A cellview may contain both physical and logical data when it contains EMH type data. When `t`, the cellview is physical only. Default is `nil`.

Value Returned

d_inst	The *dbObject* of the instance.
`nil`	The instance is not created.

Example

```
dbCreateInst(inCellView masterCV "inst1" list(0 0) "R0" 1)
dbCreateInst(inCellView masterDesign "A2`B2`Ins3" list(0 0) "R0")
```


© 2020 All Rights Reserved.

dbCreateParamInst

```
dbCreateParamInst(
     d_cellView
     d_master
     t_name
     l_point
     t_orient
     [ x_numInst
     [ l_params
     [ g_physOnly ] ] ]
     )
     => d_inst / nil
```

Description

Creates a parameterized cell instance (of cellview *d_master*) in the cellview *d_cellView* with an origin and orientation specified by *l_point* and *t_orient*.

The values and types of the parameters for this instance may be optionally specified in *l_params.* Parameters not specified in *l_params* are defaulted to the ones defined in the master cellview *d_master.* Similarly, if *l_params* is not specified, the instance is created with the same parameters defined in its master.

© 2020 All Rights Reserved.

Arguments

d_cellView	Cellview the instance is placed in.
d_master	Parameterized cell master cellview of the instance to create. This parameter must be a super master; use of a submaster can cause unexpected behavior.The rest of the arguments are identical to those of `dbCreateInst` except *l_params.*
t_name	If specified, the instance is named by *t_name*; if `nil` the program assigns a unique name to the instance. *t_name* must obey the syntax of instance names as defined at the beginning of this chapter.
l_point	Origin of the new instance.
t_orient	Orientation of the new instance.
x_numInst	Number of instances; if *t_name* is given, it overrides any value specified by *x_numInst* (that is, the number of instances is derived from the name instead of taken from the *x_numInst* argument). If not specified, the default is one.
l_params	List of *l_param,* where *l_param* is *list(t_propName t_propType g_value*).
g_physOnly	Optional boolean argument that specifies whether the instance is only in physical domain when the argument is `t`; or it is in both physical and module domain when the argument is `nil`. Default is `nil`.

Value Returned

d_inst	The *dbObject* of the instance.
`nil`	The instance is not created.

Example

```
dbCreateParamInst(inCellView masterCV "inst1" list(0 0) "R0" 1 list(list("w"
"float" 2.0) list("l" "float" 5.0) t))
```


© 2020 All Rights Reserved.

dbCreateExtParamInst

```
dbCreateExtParamInst(
     d_cellView
     d_master
     t_name
     l_point
     t_orient
     [ x_numInst
     [ l_params
     [ g_physOnly ] ] ]
     )
     => d_inst / nil
```

Description

Creates a parameterized cell instance (of cellview d_master) in the cellview d_cellView with an origin and orientation specified by l_point and t_orient. The parameters in the l_params list may be CDF parameters whose values match the CDF parameter defaults.

© 2020 All Rights Reserved.

Arguments

d_cellView	Cellview in which the instance is placed.
d_master	Parameterized cell master cellview of the instance that is to be created. This parameter must be a super master; use of a submaster can cause unexpected behavior.
t_name	The parametrized cell instance is named by t_name; if nil the program assigns a unique name to the instance. t_name must obey the syntax of instance names as defined at the beginning of this chapter.
l_origin	Origin of the new instance.
t_orient	Orientation of the new instance.
x_numinst	(Optional) Number of instances; if t_name is given, it overrides any value specified by x_numInst (that is, the number of instances is derived from the name instead of taken from the x_numInst argument). If not specified, the default is one.
l_params	(Optional) Specifies a list of `l_param`, where *`l_param`* is `list(`*`t_propName t_propType g_value`*`)`. If *`l_params`* is not specified the default is `nil`.
g_physonly	(Optional) Specifies whether cellview data is physical or both physical and logical. A cellview may contain both physical and logical data when it contains EMH type data. When `t`, the cellview is physical only. Default is `nil`.

Value Returned

d_inst	Returns dbObject of the instance.
nil	Returns nil if the instance is not created.

Example

```
i3 = dbCreateExtParamInst(cell master "I3" 0:0 "R0" 1 list(list("aInt" "int" 11))
t)
```


© 2020 All Rights Reserved.

dbCreateExtParamInstByMasterName

```
dbCreateExtParamInstByMasterName(
     d_cellView
     t_libName
     t_cellName
     t_viewName
     t_name
     l_origin
     t_orient
     [ x_numInst
     [ l_params
     [ g_physOnly ] ] ]
     )
     => d_inst / nil
```

Description

Creates an instance of the specified Pcell master in the specified cellview. The parameters in the parameter list may be CDF parameters. In this case, for each parameter the function creates an explicit local user property on the instance, even if its value is the same as the default on the master.

Arguments

d_cellView	Cellview in which the instance is created.
t_libName	Library name of the master cellview.
t_cellName	Cell name of the master cellview.
t_viewName	View name of the master cellview.
t_name	Name of the new Pcell instance. If `nil` the program assigns a unique name to the instance. The name must obey the syntax of instance names as defined at the beginning of this chapter.
l_origin	Origin of the new instance.
t_orient	Orientation of the new instance.
x_numinst	(Optional) Number of instances; if *t_name* is given, it overrides any value specified by *x_numInst* (that is, the number of instances is derived from the name instead of taken from the *x_numInst* argument). If not specified, the default is one.

© 2020 All Rights Reserved.

l_params	(Optional) Specifies a list of *l_param*, where *l_param* is `list(`*t_propName t_propType g_value*`)`. If *l_params* is not specified the default is nil.
g_physonly	(Optional) Specifies whether cellview data is physical or both physical and logical. A cellview may contain both physical and logical data when it contains EMH type data. When `t`, the cellview is physical only. Default is `nil`.

Value Returned

d_inst	Database ID of the instance.
`nil`	The instance was not created.

Example

```
dbCreateExtParamInstByMasterName(cvId "libMaster" "cellMaster" "viewMaster" "I1"
0:0 "R0" 1list(list("aa" "int" 9) list("bb" "string" "right")))
```

Creates the Pcell instance `I1` of the specified Pcell master (`"libMaster" "cellMaster" "viewMaster"`) in the cellview `cvId` with the specified parameters.

© 2020 All Rights Reserved.

dbCreateXformPCell

```
dbCreateXformPCell(
     d_cellViewId
     t_libName
     )
     => d_cellViewId / nil
```

Description

Creates a new Pcell supermaster with the additional parameters: *mag* and *angle*. The new Pcell is used in emulating magnification and any-angle rotation of instances. If the source cellview is `maskLayout`, the Pcell cell name is same as that of the source cellview, with `_xform` appended to it, otherwise, the viewName is similarly renamed.
This Pcell creates an instance of the source cellview, flattens it, and then transforms all the resultant shapes by adding the parameters *mag* and *angle*.
If the Pcell supermaster already exists, it is opened as read-only, and the cellview ID is returned.

Arguments

d_cellViewId	The database cellview ID.
t_libName	The name of the alternate library. This argument is useful when the source library that contains the source cellview is not writable. In this case, the transformed Pcell is created in the alternate library specified by this argument.

Value Returned

d_cellViewId	The Pcell super master cellview ID that was created.
`nil`	An error occurred.

© 2020 All Rights Reserved.

dbDumpPcDefinePcell

```
dbDumpPcDefinePcell(
     d_cellViewId
     t_pCellFile
     )
     => t / nil
```

Description

Writes out three lists to the file specified: The first list is a list of library names, cell names, and view names of the given cellview ID; the second list is of the parameter value; the last list is the SKILL procedure attached to the given cellview ID.

Arguments

d_cellViewId	The cellview ID. Must be a parameterized cell.
t_pCellFile	Name of the output file.

Value Returned

`t`	The output file is successfully created.
`nil`	With warning messages if any errors are encountered.

© 2020 All Rights Reserved.

dbCheckParamCell

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

Description

Checks if the specified cellview is a valid parameterized cell. If there was an error during the reading, binding, or evaluation of the specified cellview, a description of the error is returned.

Note: Errors are returned by printing messages to CIW and by placing a string property named *errorDesc* in the subMasters that had evaluation issues. The value of the string property is the error message.

Arguments

d_cellViewId	The database cellview ID.

Value Returned

t	The specified cellview is a valid parameterized cell.
nil	The specified cellview is not a valid parameterized cell.

Examples

```
dbCheckParamCell(pcell)
```


© 2020 All Rights Reserved.

dbEvalParamCell

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

Description

Evaluates the supermaster; the parameters on all the instance headers are fixed, and the submasters are evaluated.

Arguments

d_cellViewId	The database cellview ID.

Value Returned

`t`	The super master is successfully evaluated.
`nil`	The function is not successfully executed.

© 2020 All Rights Reserved.

dbFixParamCell

```
dbFixParamCell(
     d_cellViewId
     t_fileName
     )
     => t / nil
```

Description

Fixes the parameterized cell for which the reading, binding, or evaluation has failed, by re-defining it with a different SKILL procedure defined in `t_fileName`.

Arguments

`d_cellViewId`	The cellview for which the reading, binding, or evaluation has failed.
`t_fileName`	The source file that contains the SKILL procedure used in the re-definition.

Value Returned

`t`	The re-definition succeeded.
`nil`	The re-definition did not succeed or there was nothing to fix.

Example

```
dbFixParamCell(pcell "newPcell.il")
```


© 2020 All Rights Reserved.

dbHasInstance

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

Description

Checks whether the specified cellview contains any instances.

Arguments

d_cellViewId	The database cellview ID.

Value Returned

`t`	The cellview contains instances.
`nil`	The cellview does not contain instances.

© 2020 All Rights Reserved.

dbIsCellViewModified

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

Description

Checks whether the specified cellview has been modified since the last open or last save.

When all changes to a cellview have been undone using the *Undo* command, the `dbIsCellViewModified` function returns `nil`.

Arguments

`d_cellViewId`	The database cellview ID.

Value Returned

`t`	The cellview has been modified.
`nil`	The cellview has not been modified.

© 2020 All Rights Reserved.

dbReplaceParam

```
dbReplaceParam(
     d_cellViewId
     t_name
     t_type
     g_value
     )
     => d_prop / nil
```

Description

Add or modifies a cellview parameter if the cellview is a pcell super master. If the cellview is not a super master, `dbReplaceParam` gives a warning stating that it cannot define a parameter for a non super master, and quits.
This function is only used on pcell super master cellviews.

Arguments

d_cellViewId	The cellview whose parameter is to be changed.
t_name	The name of the parameter.
t_type	The type of the parameter.
g_value	The value of the parameter.

Value Returned

d_prop	The property ID of the replaced parameter.
`nil`	The parameter was not successfully replaced.

© 2020 All Rights Reserved.

dbReplaceInstParamList

```
dbReplaceInstParamList(
     d_instId
     l_paramList
     )
     => t / nil
```

Description

Atomically replaces all the Pcell instance parameters with the given parameter list. The resulting Pcell subMaster is evaluated only once.

Arguments

d_instId	Specifies the ID of the Pcell instance whose parameters are to be changed.
l_paramList	Specifies the list of parameters to be replaced.

Value Returned

t	The instance parameter list was successfully replaced.
nil	The parameter was not successfully replaced.

Example

```
dbReplaceInstParamList(pInst1 list(list("metal2Layer" "string" "Metal5")
list("boolVal1" "boolean" nil) list("intLength" "integer" 10)))
```


© 2020 All Rights Reserved.

dbFindAnyInstByName

```
dbFindAnyInstByName(
     d_cellView
     t_name
     )
     => d_inst / nil
```

Description

Retrieves an instance or mosaic.

The use of this function is more efficient than searching through the list of instances on the cellview for the correct one and is preferred when a specific instance is needed.

Note: `dbFindAnyInstByName` is also defined as `dbGetInstanceByName` for compatibility with earlier versions of software.

Arguments

d_cellView	Name of the cellview.
t_name	Name of the instance.

Value Returned

d_inst	The *dbObject* of the instance.
nil	The instance or mosaic is not retrieved.

Examples

```
dbFindAnyInstByName(inCellView "inst1")
dbFindAnyInstByName(inCellView "myInv<0:1>")
```


© 2020 All Rights Reserved.

dbFindMemInstByName

```
dbFindMemInstByName(
     d_cellView
     t_name
     )
     => ( d_inst x_index ) / nil
```

Description

Retrieves an instance or mosaic and returns its member index.

Unlike the `dbFindAnyInstByName` function described earlier, `dbFindMemInstByName` returns not only the instance *dbObject*, but also the member index of the instance to which the name applies. This member index is applicable to iterated instances.

Arguments

d_cellView	Name of cellview.
t_name	Name of the instance.

Value Returned

d_inst x_index	The *dbObject* and the member index of the instance as a list.
nil	The instance or mosaic is not retrieved.

Examples

```
inst = dbCreateInst(cvSch cvSym "myInst" list(0 0) "R0" 2)
dbFindMemInstByName(cvSch "myInst") => (db:<dbId#> 0)
dbFindMemInstByName(cvSch "myInst<1>") => nil
```

```
inst = dbCreateInst(cvSch cvSym "myInst" list(0 0) "R0")
dbFindMemInstByName(cvSch "myInst") => (db:<dbId#> 0)
dbFindMemInstByName(cvSch "myInst<1>") => nil
```

```
inst = dbCreateInst(cvSch cvSym "m<0:3>" list(0 0) "R0")
dbFindMemInstByName(cvSch "m<0>") => (db:<dbId#> 0)
dbFindMemInstByName(cvSch "m<1>") => (db:<dbId#> 1)
dbFindMemInstByName(cvSch "m<3>") => (db:<dbId#> 3)
dbFindMemInstByName(cvSch "m<4>") => nil
dbFindMemInstByName(cvSch "m") => nil
```

```
inst = dbCreateInst(cvSch cvSym "m<3>" list(0 0) "R0")
dbFindMemInstByName(cvSch "m<0>") => nil
dbFindMemInstByName(cvSch "m") => nil
```


© 2020 All Rights Reserved.

```
dbFindMemInstByName(cvSch "m<2>") => nil
dbFindMemInstByName(cvSch "m<3>") => (db:<dbId#> 0)
```


© 2020 All Rights Reserved.

dbGetHierPathTransform

```
dbGetHierPathTransform(
     l_dbInstId
     )
     => x_InstTransform
```

Description

Returns the concatenated transform of all instance or mosaic IDs present in the nested list passed as the parameter. The nested list represents the path to an object, such as the lists returned by `dbGetTrueOverlaps` or `dbGetOverlaps`.

The first elements of the list represent the instances and/or the mosaics traversed to get the object. The last element of the list is the object itself.

Note: If the object is an instance or a mosaic, its transform is not concatenated to the resulting transform. To include concatenation of the last object transform (if it is an instance or a mosaic), use `dbGetInstTransform`.

You can pass the resulting transform to the `dbCopyFig` or `dbMoveFig` functions as their third argument, in order to copy or move the shape to the cellview passed to `dbGetTrueOverlaps` (as its first argument).

Arguments

l_dbInstId	List of instance IDs of the `hierPath`.

Value Returned

x_InstTransform	Concatenated transform of the instance IDs passed.

Examples

```
objectsPaths = dbGetOverlaps(geGetEditCellView() geGetEditCellView()~>bBox t 32 t)
foreach(objectPath objectsPaths println(dbGetHierPathTransform(objectPath)))
```


© 2020 All Rights Reserved.

dbGetInstByName

```
dbGetInstByName(
     d_cellView
     t_name
     )
     => d_inst / nil
```

Description

Retrieves an instance or mosaic.

`dbGetInstByName` is maintained only for backward compatibility; you should now use `dbFindAnyInstByName`.

Arguments

d_cellView	Name of the cellview.
t_name	Name of the instance.

Value Returned

d_inst	The database object of the instance.
nil	The instance or mosaic is not retrieved.

© 2020 All Rights Reserved.

dbGetInstanceByName

```
dbGetInstanceByName(
     d_cellView
     t_name
     )
     => d_inst / nil
```

Description

Retrieves an instance or mosaic.

`dbGetInstanceByName` is maintained only for backward compatibility; you should now use `dbFindAnyInstByName`.

Arguments

d_cellView	Name of the cellview.
t_name	Name of the instance.

Value Returned

d_inst	The database object of the instance.
nil	The instance or mosaic is not retrieved.

© 2020 All Rights Reserved.

dbGetInstNameNumInst

```
dbGetInstNameNumInst(
     t_name
     )
     => x_numInst / nil
```

Description

Retrieves the number of instances of an instance or mosaic

Arguments

t_name	Name of the instance.

Value Returned

x_numInst	The number of instances.
nil	The number cannot be retrieved.

Examples

```
dbGetInstNameNumInst("i") 1
dbGetInstNameNumInst("i<3>") 1
dbGetInstNameNumInst("i<0:3>") 4
```


© 2020 All Rights Reserved.

dbGetInstTransform

```
dbGetInstTransform(
     l_dbInstId
     )
     => x_InstTransform
```

Description

Returns the concatenated transform of all instance or mosaic IDs present in the nested list passed as the parameter. The nested list represents the path to an object, such as the lists returned by `dbGetTrueOverlaps` or `dbGetOverlaps`.

The first elements of the list represent the instances and/or the mosaics traversed to get the object. The last element of the list is the object itself.

Note: If the object is an instance or a mosaic, its transform is also concatenated to the resulting transform. To get rid off the concatenation of the last object transform, use `dbGetHierPathTransform`.

Arguments

l_dbInstId	List of instance IDs of the `hierPath`.

Value Returned

x_InstTransform	Concatenated transform of all instance IDs passed.

Examples

```
objectsPaths = dbGetOverlaps(geGetEditCellView() geGetEditCellView()~>bBox t 32 t)
foreach(objectPath objectsPaths println(dbGetInstTransform(objectPath)))
```


© 2020 All Rights Reserved.

dbIsAnyInstBaseNameUsed

```
dbIsAnyInstBaseNameUsed(
     d_cellView
     t_name
     )
     => t / nil
```

Description

Check if the name is used as instance basename in the cellview.

Arguments

d_cellView	Cellview within which to check the base name.
t_name	Base name of the instance.

Value Returned

t	The name is used as instance base name in the cellview.
nil	The name is not used as instance base name in the cellview.

Example

```
dbIsAnyInstBaseNameUsed(cv "myInst")
```


© 2020 All Rights Reserved.

dbCreateInstByMasterName

```
dbCreateInstByMasterName(
     d_cellView
     t_libName
     t_cellName
     t_viewName
     t_instName
     l_origin
     t_orient
     [ x_numInst ]
     [ g_physOnly ]
     )
     => d_inst / nil
```

Description

Creates an instance (of cellview—*t_libName/t_cellName/t_viewName*) in *d_cellView* with an origin and orientation specified by *l_point* and *t_orient*.

© 2020 All Rights Reserved.

Arguments

d_cellView	Cellview where the instance is placed.
t_libName	Library name of the master cellview.
t_cellName	Cell name of the master cellview.
t_viewName	View name of the master cellview.
t_instName	If specified, the instance is named by *t_name*; if `nil` the program assigns a unique name to the instance. *t_name* must obey the syntax of instance names as defined at the beginning of this chapter.
l_origin	Origin of the new instance.
t_orient	Orientation of the new instance.
x_numInst	Number of instances; if *t_name* is given, it overrides any value specified by *x_numInst* (that is, the number of instances is derived from the name instead of taken from the *x_numInst* argument). If not specified, the default is one.
g_physOnly	Optional boolean argument that specifies whether cellview data is physical or both physical and logical. A cellview may contain both physical and logical data when it contains EMH type data. When `t`, the cellview is physical only. Default is `nil`.

Value Returned

d_inst	The *dbObject* of the instance.
`nil`	The instance is not created.

Example

```
dbCreateInstByMasterName(cv "basic" "nand2" "symbol" "inst1" list(0 0) "R0" 1)
```


© 2020 All Rights Reserved.

dbCreateParamInstByMasterName

```
dbCreateParamInstByMasterName(
     d_cellView
     t_libName
     t_cellName
     t_viewName
     t_name
     l_origin
     l_orient
     [ x_numInst
     [ l_params
     [ g_physOnly ] ] ]
     )
     => d_inst / nil
```

Description

Creates an instance of the specified Pcell master in the specified cellview. The parameters in the parameter list may be CDF parameters. In this case, when you create a Pcell instance and pass a parameter, with the same value as the default value, `dbCreateParamInstByMasterName()` does not create an explicit local user property on the instance because the value is the same as the default. If the value is different from the default, the function stores the value as a local property on the instance.

Arguments

d_cellView	Cellview in which the instance is created.
t_libName	Library name of the master cellview.
t_cellName	Cell name of the master cellview.
t_viewName	View name of the master cellview.
t_instName	Name of the new Pcell instance. If `nil` the program assigns a unique name to the instance. The name must obey the syntax of instance names as defined at the beginning of this chapter.
l_origin	Origin of the new instance.
l_orient	Orientation of the new instance.
x_numInst	Number of instances; if *t_name* is given, it overrides any value specified by *x_numInst* (that is, the number of instances is derived from the name instead of taken from the *x_numInst* argument). If not specified, the default is one.

© 2020 All Rights Reserved.

l_params	List of *l_param*, where *l_param* is `list(` *t_propName t_propType g_value*`)`.
g_physOnly	Optional boolean argument that specifies whether the instance is only in physical domain when the argument is `t`; or it is in both physical and module domain when the argument is `nil`. Default is `nil`.

Value Returned

d_inst	Database ID of the instance.
`nil`	The instance was not created.

Example

```
dbCreateParamInstByMasterName(inCellView "basic" "nand" "symbol" "inst1"
list(0 0) "R0" 1 list(list("w" "float" 2.0) list("l" "float" 5.0) t))
```


© 2020 All Rights Reserved.

dbDumpPcell

```
dbDumpPcell(
     d_cvId
     t_outputFile
     t_pCellFile
     )
     => t / nil
```

Description

The `dbDumpPcell` function is used in the implementation of the *keepPcell* option, during streamIn and streamOut. Pcells are preserved by writing the pcells to SKILL files (during StreamOut) which are loaded back into CDB/Virtuoso (during StreamIn).

Note: Dumping parameterized cell devices defined in a technology file is not supported in this function.

Arguments

d_cvId	Must be a valid cellview ID and pcell.
t_outputFile	Output file which can be reloaded to recreate the pcell.
t_pCellFile	Output file for the SKILL procedure attached to the pcell, this file will be given to `dbDefineProc` when reloading `outputFile`.

Value Returned

`t`	The files are dumped successfully.
`nil`	The files are not dumped successfully.

© 2020 All Rights Reserved.

dbDumpPcellWithLib

```
dbDumpPcellWithLib(
     d_cvId
     t_outputFile
     t_pCellFile
     )
     => t / nil
```

Description

Same functionality as `dbDumpPcell`, which preserves pcells through StreamIn and StreamOut by writing the pcells to SKILL files. `dbDumbPcellWithLib` does not directly write the library name in the SKILL code, instead writes a SKILL variable, *`dbD_NewLibNameForDump`* whose value can be set to the name of the target library before loading the SKILL file.

Note: Dumping parameterized cell devices defined in a technology file is not supported in this function.

Arguments

`d_cvId`	Must be a valid cellview ID and pcell.
`t_outputFile`	Output file which can be reloaded to recreate the pcell.
`t_pCellFile`	Output file for the SKILL procedure attached to the pcell, this file will be given to `dbFefineProc` when reloading outputFile.

Value Returned

`t`	The files are dumped successfully.
`nil`	The files are not dumped successfully.

© 2020 All Rights Reserved.

dbCreateViaShapeArray

```
dbCreateViaShapeArray(
     d_techCVId
     g_viaLayer
     [ d_netId ]
     f_x
     f_y
     f_xPitch
     f_yPitch
     f_w
     f_length
     x_row
     x_column
     )
     => l_xy / nil
```

Description

Creates an array of via shapes (rectangles) on the given layer in the technology cellview and adds all the shapes to the given net.

Note: Specify `nil` as `d_netId`, if you do not want via shapes to be added to a net.

Arguments

d_techCVId	ID of the technology cellview.
g_viaLayer	Layer on which to create the via shape array.
d_netId	ID of the net to add the via shapes to.
f_x	X coordinate of the starting point.
f_y	Y coordinate of the starting point.
f_xPitch	Length along the X axis allocated to one shape, including the width of the rectangle and the space along the X axis between two adjacent rectangles.
f_yPitch	Length along the Y axis allocated to one shape, including the length of the rectangle and the space along the Y axis between two adjacent rectangles.
f_w	Width (along the X axis) of the rectangle.
f_length	Length (along the Y axis) of the rectangle.
x_row	Number of rows along the Y axis.

© 2020 All Rights Reserved.

x_column	Number of columns along the X axis.

Value Returned

l_xy	List of X and Y. x = x + xPitch * column and y = y + yPitch * row.
nil	The via shape array is not created successfully.

Example

```
dbCreateViaShapeArray( cvId "contact" netId 10.2 10.1 5.5 6.7 3.1 4.9 2 3 )
==>list(26.7 23.5)
```


© 2020 All Rights Reserved.

dbSetInstHeaderMasterName

```
dbSetInstHeaderMasterName(
     d_instHeader
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Sets the master of individual instance on the instHeader to the new master. All the instances are removed from the old instHeader and move to the new instHeader, which may exist or not. When an instHeader is empty, meaning it does not have any instance on this instHeader, the instHeader is removed and the ID is no longer valid.

Arguments

d_instHeader	Instance header whose master library name, cell name, and view name are being replaced by the specified names.
t_libName	New master library name. If it is `nil`, use the one from the `instHeader`.
t_cellName	New master cell name. If it is `nil`, use the one from the `instHeader`.
t_viewName	New master view name. If it is `nil`, use the one from the `instHeader`.

Value Returned

`t`	The library name/cell name/view name of the master is set successfully.
`nil`	The library name/cell name/view name of the master is not set successfully.

Example

```
dbSetInstHeaderMasterName(instH1 "basic" "nand2" "symbol")
```


© 2020 All Rights Reserved.

dbIsSameMaster

```
dbIsSameMaster(
     d_anyInst1
     d_anyInst2
     )
     => t / nil
```

Description

Determines if two instances or mosaics are instantiated from the same cellview master.

Both the instances passed in should be instantiated in the same cellview. If two instances from different cellviews are passed in, even if they are instances of the same master, `nil` is returned.

Arguments

d_anyInst1	First instance.
d_anyInst2	Second instance.

Value Returned

`t`	The two instances are in the same cellview and have the same master.
`nil`	The two instances are not in the same cellview or if they do not have the same master.

Example

```
dbIsSameMaster(instId1 instId2)
```


© 2020 All Rights Reserved.

dbIsObjectBound

```
dbIsObjectBound(
     g_object
     )
     => t / nil
```

Description

Determines whether or not an object is bound. If the master cellview of the object is opened, the object is bound. Otherwise it is not bound.

Arguments

g_object	An instance, a mosaic, a mosaic instance, an instance header, an instance pin, or an instance terminal.

Value Returned

`t`	The object is bound.
`nil`	The object is not bound.

© 2020 All Rights Reserved.

dbGetCellViewSymmetry

```
dbGetCellViewSymmetry(
     d_cellView
     )
     => t_symmetry / nil
```

Description

Returns the symmetry of the specified cellview.

Arguments

d_cellviewId	The database ID of the cellview.

Value Returned

t	Returns the symmetry value of the cellview.
nil	The cellview ID is invalid.

© 2020 All Rights Reserved.

dbSetCellViewSymmetry

```
dbSetCellViewSymmetry(
     d_cellViewID
     t_symmetry
     )
     => t / nil
```

Description

Sets the symmetry (or orientation) on a cellview (for which the ID is specified). The symmetry of the cellview can be set to a string value, such as X, Y, XY, any, or none, where X mirrors the cellview about the x axis and Y mirrors the cellview about the y axis.

Arguments

d_cellviewID	The database ID of the cellview.
t_symmetry	String value specifying the symmetry of the cellview. You can also use a string value from the following list: `X` for symmetry about the x axis, `Y` for symmetry about the y axis, `XY` for symmetry about the x and y axis, `any` for symmetry about any axis, and `no` for no specific orientation.

Value Returned

`t`	The symmetry value is set.
`nil`	The cellview ID is invalid.

Example

```
cv = geGetWindowCellView(hiGetCurrentWindow())
dbSetCellViewSymmetry(cv "X")
dbGetCellViewSymmetry(cv)
```


© 2020 All Rights Reserved.

dbGetCellViewEEQMaster

```
dbGetCellViewEEQMaster(
     d_cellView
     )
     => t_EEQMaster / nil
```

Description

Returns the value of the EEQmaster attribute of the cellview.

The EEQmaster (electrically equivalent master) is stored as a string on the cellview. If the string is empty, `nil` will be returned. EEQ cellviews are devices such as OR-gates or inverters that have several implementations with different shapes, geometries, and orientations.

Arguments

d_cellviewId	The database ID of the cellview.

Value Returned

t_EEQMaster	The the value of the EEQmaster attribute is returned.
`nil`	The cellview ID is invalid. The EEQMaster attribute is not set or the string value is empty.

© 2020 All Rights Reserved.

dbSetCellViewEEQMaster

```
dbSetCellViewEEQMaster(
     d_cellView
     t_EEQMaster
     )
     => t / nil
```

Description

Sets the value of the EEQmaster attribute of the specified cellview.

The EEQmaster (electrically equivalent master) is stored as a string on the cellview. If the string value of *t_EEQMaster* is an empty string, the value is effectively "unset".

Arguments

d_cellviewId	The database ID of the cellview.
t_EEQMaster	String value.

Value Returned

t	The value of the EEQmaster attribute is set.
nil	The cellview ID is invalid. An EEQMaster attribute is not set.

© 2020 All Rights Reserved.

dbIsInstTransparent

```
dbIsInstTransparent(
     d_instanceId
     )
     => t / nil
```

Description

Checks if the specified instance ID is set to be transparent to the Layout XL binder.

Arguments

d_instanceId	Database ID of the instance to be checked.

Value Returned

`t`	The specified instance is set as transparent to the Layout XL binder.
`nil`	The instance is not transparent. The default value for any instance is `nil`.

Example

```
isTransparent = dbIsInstTransparent(instanceId)
```

Returns `t` if the instance `instanceId` is already set as transparent. Otherwise, returns `nil`.

© 2020 All Rights Reserved.

dbSetInstTransparent

```
dbSetInstTransparent
     d_instanceId
     g_transparent
     )
     => t / nil
```

Description

Sets an instance as transparent to the Layout XL binder.

Arguments

d_instanceId	Database ID of the instance to be set as transparent.
g_transparent	A Boolean that determines whether instance should be set as transparent. The default is `nil`.

Value Returned

`t`	The specified instance is set as transparent.
`nil`	The specified instance is not set as transparent.

Example

```
dbSetInstTransparent(inst, t)
```

Makes instance `inst` transparent. When Layout XL binder is run, this instance is considered as transparent, enabling bindings to be made for instances inside the master of this instance.

```
dbSetInstTransparent(inst, nil)
```

Removes transparency from the instance `inst`. Layout XL can no longer bind to the instances inside the master of this instance.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

5

Connectivity Database Access Functions

This chapter includes the following topics:

- Connectivity Creation and Modification Functions
- Equivalent Connectivity Time Functions
- Connectivity Access Functions

© 2020 All Rights Reserved.

Connectivity Creation and Modification Functions

The following database functions create and modify connectivity objects in the database.

- dbCreateNet
- dbMakeNet
- dbMergeNet
- dbCreateUniqueNamedNet
- dbSetNetNameDescend
- dbIsNetNameDescend
- dbNetIsSig
- dbDeleteAllNet
- dbCreateTerm
- dbCreateInstTerm
- dbHasShortedTerminals
- dbCreatePin
- dbCreateConn
- dbCreateConnByName
- dbCreateConnByNewName
- dbSubFigFromNet
- dbAddFigToNet
- dbGetNetVoltageRange
- dbSetNetVoltageRange
- dbUnsetNetVoltageRange
- dbGetNetVoltageRangeSource
- dbSetNetVoltageRangeSource
- dbUnsetNetVoltageRangeSource
- dbGetCellViewNetVoltageRange

© 2020 All Rights Reserved.

- dbSetCellViewNetVoltageRange
- dbIsCellViewNetVoltageRangeSet
- dbIsNetVoltageRangeSet
- dbUnsetCellViewNetVoltageRange
- dbGetNetPowerDomain
- dbSetNetPowerDomain
- dbRenameNet
- dbIsCellViewPhysicalOnly
- dbUnsetCellViewPhysicalOnly

© 2020 All Rights Reserved.

dbCreateNet

```
dbCreateNet(
     d_cellView
     t_name
     [ d_parentNet ]
     )
     => d_net / nil
```

Description

Creates a new net in a cellview. If the net already exists, returns `nil`.

To create a child net, the *`d_parentNet`* object must not be `nil`. If *`t_name`* is an empty string or `nil`, the name of the parent net is used. *`t_name`* can be a unique name, such as "`net1`" or, a hierarchy name, such as `"moduleA`moduleB`Net2"` with at least one hierarchy delimiter (`). However, if you specify a hierarchy name, the *`d_parentNet`* object must be `nil`.

To create a new top-level net, *`t_name`* must be specified as a valid name string and *`d_parentNet`* is either `nil` or unspecified.

Arguments

`d_cellView`	Cellview the net is created within.
`t_name`	Name of the net. For a child net, it can be an empty string or `nil`.
`d_parentNet`	To create a new top-level net, *`d_parentNet`* is either `nil` or unspecified. To create a child net, the *`d_parentNet`* object must not be `nil`.

Value Returned

`d_net`	A new net in the cellview, if one does not already exist.
`nil`	The net is not created or the net already exists.

Examples

Creates a top-level net.

```
parentNet = dbCreateNet(cellview "net")
```


© 2020 All Rights Reserved.

```
hierNet = dbCreateNet(cellview "moduleA`moduleB`hierNet")
```

Creates a child net.

```
childNet = dbCreateNet(cellview "childNet" parentNet)
```


© 2020 All Rights Reserved.

dbCreateNamedSubNet

```
dbCreateNamedSubNet(
     d_parentNet
     t_subNetName
     )
     => t_subnetId / nil
```

Description

Creates a subnet with the specified name. The behavior is different from dbCreateNet because when a root net ID is specified. dbCreateNameSubNet creates an explicitly named scalar net with the specified subnet name.

The following restrictions apply:

- The specified *d_parentNetId* must be a single-bit net.
- The *d_parentNetId* must be a scalar net, which means that it cannot be a bus-bit name.
- The *d_parentNetId* cannot have any subnets with the specified subnet name.
- The *t_subNetName* must be a scalar net, which means that it cannot be a bus-bit name.
- A net in the cellview cannot exist with the subnet name.

Arguments

d_parentNetId	Database ID of the root net with which the subnet will be associated. The *d_parentNetId* also provides the cellview in which the new net is created.
t_subNetName	Name for the new subnet.

Value Returned

t_subnetId	Name of the subnet that is created with the parent net as its root net.
nil	The subnet could not be created or the subnet already exists.

© 2020 All Rights Reserved.

Example

In the following example, generating nets in the cellview `cv` creates the named nets `a` and `sn2`. Generating the subnets of net `a` creates subnets `sn1` and `sn2`.

```
rootNet = dbCreateNet(cv "a")
```

Create subnet `sn1`.

```
    subNet1 = dbCreateNet(cv "sn1" rootNet)
```

Create subnet `sn2`.

```
    subNet2 = dbCreateNamedSubNet(rootNet "sn2")
cv~>nets~>name
("a" "sn2")
rootNet~>subnet~>name
("sn1" "sn2)
```


© 2020 All Rights Reserved.

dbMakeNet

```
dbMakeNet(
     d_cellView
     t_name
     [ d_parentNet ]
     )
     => d_net / nil
```

Description

Creates a new net in a cellview. If the net already exists, it returns the *dbObject* of the net.

To create a child net, this function works identically to `dbCreateNet`. *t_name* can be a unique name, such as "`net1`" or, a hierarchy name, such as `"moduleA`moduleB`Net2"` with at least one hierarchy delimiter (`). However, if you specify a hierarchy name, the *d_parentNet* object must be `nil`.

Arguments

d_cellView	Cellview the net is created within.
t_name	Name of the net. For a child net it can be an empty string or `nil`.
d_parentNet	To create a new top-level net, *d_parentNet* must be either `nil` or unspecified. To create a child net, the *d_parentNet* object must not be `nil`.

Value Returned

d_net	The new net or the existing top-level net *t_name*.
`nil`	The net is not created.

Examples

Make a top-level net.

```
parentNet = dbMakeNet(cellview "net1")
hierNet = dbMakeNet(cellview "moduleA`moduleB`hierNet")
```

Make a child net.

```
childNet = dbMakeNet(cellview "childNet" parentNet)
```


© 2020 All Rights Reserved.

dbMergeNet

```
dbMergeNet(
     d_netId1
     d_netId2
     )
     => dnetId1 / nil
```

Description

Merges one `netId2` into `netId1`. The pins, terminals, connections, figures, properties, and signals of the merging net (`netId2`) are merged into the surviving net (`netId1`), and member nets of `netId2`'s signals become members of `netId1`'s signals. Both of the nets may already be attached to a terminal; both nets must have the same number of bits. If the two nets are in the same net hierarchy, `netId1` must not be a descendant of `netId2`. If the merge is successful, `netId1` is returned; otherwise, `nil` is returned.

Arguments

d_netId1	The surviving net to which the merging net gets merged into.
d_netid2	The net that is going to get merged into the surviving net.

Value Returned

dNetId1	The merge is successful.
`nil`	Otherwise, `nil` is returned.

© 2020 All Rights Reserved.

dbCreateUniqueNamedNet

```
dbCreateUniqueNamedNet(
     d_cellView
     t_prefix
     x_numBits
     )
     => d_net / nil
```

Description

Creates a new net with a unique name.

A unique name for the net is automatically generated. The name is composed of the given *t_prefix* followed by a unique number as specified by *x_numBits*.

Arguments

d_cellView	Cellview within which the net is created.
t_prefix	Prefix used in auto-naming the net. nil or an empty string defaults to dbNet. **Note:** Here, N_ cannot be used as prefix to create bus nets because it conflicts with the prefix of the auto-names.
x_numBits	Integer specifying the number of bits for the net. If *x_numBits* is 1, the name is *t_ prefix* followed by a unique number. If *x_numBits* is greater than 1, *t_prefix* is followed by *<0:numBits-1>.* If *x_numBits* is less than -1, *t_prefix* is followed by *<abs(numBits)-1:0>*.

Value Returned

d_net	The net that is created.
nil	The net is not created.

Examples

Create a single bit net named N*x* where *x* is a unique number.

```
net = dbCreateUniqueNamedNet(cellview "N" 1)=> N1
```


© 2020 All Rights Reserved.

Create a new net named N<0:3>.

```
net = dbCreateUniqueNamedNet(cvId "N" 4)=> N<0:3>
```

Create a bus named B<30:0>.

```
bus = dbCreateUniqueNamedNet(cellview "B" -31)=> B<30:0>
```


© 2020 All Rights Reserved.

dbSetNetNameDescend

```
dbSetNetNameDescend(
     d_cellView
     g_mode
     )
     => t / nil
```

Description

Sets a mode, descending or ascending, which determines the direction of the net name vector expression.

This mode is mainly used in `dbCreateUniqueNameNet` together with the sign of *`numBit`* to determine the direction of the net name vector expression.

Arguments

`d_cellView`	Cellview within which the mode is set.
`g_mode`	Mode to be set. Either `t` or `nil`.

Value Returned

`t`	The mode is set successfully.
`nil`	The function fails.

Example

```
dbSetNetNameDescend(cellview t)
```


© 2020 All Rights Reserved.

dbIsNetNameDescend

```
dbIsNetNameDescend(
     d_cellView
     )
     => t / nil
```

Description

Checks if the net name descend mode is set or not.

Arguments

d_cellView	Cellview within which the mode is checked.

Value Returned

t	The mode is set.
nil	The mode is not set.

Example

```
dbIsNetNameDescend(cellview)
```


© 2020 All Rights Reserved.

dbNetIsSig

```
dbNetIsSig(
     d_netId
     )
     => t / nil
```

Description

Tests whether the specified net ID is also a signal.

Arguments

d_netId	A net ID.

Value Returned

`t`	The specified net ID is also a signal.
`nil`	The specified net ID is not also a signal.

© 2020 All Rights Reserved.

dbDeleteAllNet

```
dbDeleteAllNet(
     d_cellView
     )
     => t / nil
```

Description

Deletes all nets from a cellview.

Arguments

d_cellView	Specifies the cellview.

Value Returned

`t`	The nets are deleted.
`nil`	The nets are not deleted.

Example

```
dbDeleteAllNet(cellview)
```


© 2020 All Rights Reserved.

dbCreateTerm

```
dbCreateTerm(
     d_net
     t_name
     t_direction
     [ x_unused ]
     [ g_physOnly ]
     )
     => d_term / nil
```

Description

Creates a terminal for a net.

The direction of the terminal is specified by a string *t_direction.* A list of valid *t_direction* specifiers is available in the section Access Function Argument Types.

The terminal name must be unique within a given cellview. Member terminals propagating the same signal must have the same name.

© 2020 All Rights Reserved.

Arguments

d_net	Name of the net.
t_name	Name of the terminal to create. If *t_name* is `nil` or an empty string, the default is to use the same name as the net *d_net*.
t_direction	A string specifying the direction of the terminal. See Type t_direction for valid terminal directions.
x_unsed	This argument is for backward compatibility. An integer must be given.
g_physOnly	Optional boolean argument that specifies whether cellview data is physical or both physical and logical. A cellview may contain both physical and logical data when it contains EMH type data. When `t`, the cellview is physical only. Default is `nil`.

Value Returned

d_term	The new terminal.
`nil`	The terminal is not created.

Example

```
netId = dbCreateNet(geGetEditCellView() "CLK")
termId = dbCreateTerm(netId "myTerm" "input" 0 t)
```


© 2020 All Rights Reserved.

dbCreateInstTerm

```
dbCreateInstTerm(
     d_net
     d_inst
     d_term
     )
     => d_instTerm / nil
```

Description

Creates an instance terminal on the instance *d_inst* for the master terminal *d_term.*

The instance *d_inst* cannot be a mosaic instance. Instance terms are not supported in mosaic instances.

Terminal *d_term* must exist in the master cellview of the instance *d_inst.* If *d_net* is `nil`, the instance terminal is created. If *d_net* defines a valid net, then an attempt is made to connect the instance terminal to the net. If the instance terminal already exists, the attempted connection still takes place. There can be only one instance terminal of a given master terminal on an instance of a master cellview.

Arguments

d_net	Net to be connected to the instance terminal. If `nil`, the instance terminal created is left unconnected.
d_inst	Instance of a master cellview where *d_term* is defined.
d_term	Terminal defined in the master cellview of the instance *d_inst.*

Value Returned

d_instTerm	The new or existing instance terminal.
`nil`	The terminal is not created.

Examples

Create a terminal in the master cellview.

```
net = dbCreateNet(master "A")
term = dbCreateTerm(master "" "input")
```


© 2020 All Rights Reserved.

Create an instance of the master.

```
inst = dbCreateInst(cellview master "" list(0 0) "R0" 1)
```

Create a net, an instance terminal of terminal "A", and connect the net to the instance terminal.

```
inNet = dbCreateNet(cellview "inNet")
instTerm = dbCreateInstTerm(inNet inst term)
```


© 2020 All Rights Reserved.

dbHasShortedTerminals

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

Description

Checks whether the specified cellview contains nets that are connected to multiple terminals.

Arguments

d_cellViewId	The database cellview ID.

Value Returned

`t`	The specified cellview has shorted terminals.
`nil`	The cellview ID is invalid or there is an error.

© 2020 All Rights Reserved.

dbCreatePin

```
dbCreatePin(
     d_netId
     d_figId | nil
     [ t_pinName | nil ]
     [ d_termId | nil]

     )
     => d_pin / nil
```

Description

Creates a pin.

The figure of a pin represents physical implementation and graphical rendering for the pin. A pin cannot exist without being attached to a terminal. A terminal cannot exist without being attached to a net. Therefore, if the *root-net* (the top-most parent) of net `d_net` does not have a terminal, one is created automatically.

Note: The access direction of the pin is initialized to `None`.

Arguments

`d_netId`	Net ID for the pin.
`d_figId`	Figure ID of the pin. The figure *purpose* attribute is left unchanged.
`t_pinName`	Pin name `nil` or an empty string produces an automatically generated name. Pin name `t_pinName` is optional. If specified, it must be unique among pins of the same terminal. A unique name is automatically generated if `t_pinName` is not given.
`d_termId`	Term ID for the pin. Term ID `d_termId` is optional.

© 2020 All Rights Reserved.

Value Returned

d_pin	The new pin.
nil	The pin is not created.

Example

```
pin = dbCreatePin(net fig)
```


© 2020 All Rights Reserved.

dbCreateConn

```
dbCreateConn(
     d_net
     d_inst
     d_term
     )
     => d_instTerm / nil
```

Description

Identical to `dbCreateInstTerm`, except the net *d_net* defines a valid net to be connected to the instance terminal of a master terminal *d_term.*

Arguments

d_net	Net to be connected to the instance terminal.
d_inst	Instance of the master cellview where terminal *d_term* is defined. *d_inst* cannot be a mosaic instance. Instance terms are not supported in mosaic instances.
d_term	Terminal defined in the master cellview of instance *d_inst.*

Value Returned

d_instTerm	The connected instance terminal.
`nil`	The instance is not connected.

Example

```
conn = dbCreateConn(net inst term)
```

Creates an instance terminal of master terminal `term` on instance `inst` and connect it to `net.`

© 2020 All Rights Reserved.

dbCreateConnByName

```
dbCreateConnByName(
     d_net
     d_inst
     t_name
     )
     => d_instTerm / nil
```

Description

Similar to `dbCreateConn`, except the master terminal is defined by *`t_name`* instead of being specified by the master terminal's *dbObject.*

Arguments

`d_net`	Net to be connected to the instance terminal.
`d_inst`	Instance of the master cellview where terminal *`d_term`* is defined. *`d_inst`* cannot be a mosaic instance. Instance terms are not supported in mosaic instances.
`t_name`	The name of a terminal defined in the master cellview of instance *`d_inst.`*

Value Returned

`d_instTerm`	The connected instance terminal.
`nil`	The instance terminal is not connected.

Example

```
conn = dbCreateConnByName(net inst "A")
```

Creates an instance terminal of master terminal `A` on instance `inst` and connect it to the `net`.

Note: If you specify `nil` as the `d_net` value, a warning message displays and the instance terminal is not created.

© 2020 All Rights Reserved.

dbCreateConnByNewName

```
dbCreateConnByNewName(
     d_net
     d_inst
     t_name
     )
     => d_instTerm / nil
```

Description

Similar to `dbCreateConnByName`, except *`t_name`* can refer to a master terminal to be created in the master cellview of instance *`d_inst`*.

Note: In that case, the instance terminal created is left *unbound* to its master. If terminal *`t_name`* exists on the instance's master cellview, this function behaves the same as `dbCreateConnByName`.

Arguments

`d_net`	Net to be connected to the instance terminal.
`d_inst`	Instance of the master cellview where terminal *`t_name`* is defined. *`d_inst`* cannot be a mosaic instance. Instance terms are not supported in mosaic instances.
`t_name`	Name of a new or existing terminal in the master cellview of instance *`d_inst`*.

Value Returned

`d_instTerm`	The connected instance terminal.
`nil`	The instance terminal is not created.

Example

Create an instance terminal of a non-existing master terminal named `A`.

```
unbound = dbCreateConnByNewName(net inst "A")
```

Note: If you specify `nil` as the `d_net` value, a warning message displays and the instance terminal is not created.

© 2020 All Rights Reserved.

dbSubFigFromNet

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

Description

Removes the given figure object from the net object with which it is already associated.

Arguments

d_figId	The given figure object.

Value Returned

`t`	The figure object was removed.
`nil`	The figure object was not removed.

Example

```
dbSubFibFromNet(figId)
```

When the above command is successful you can verify the result using

```
netId~>figs=>list(figs...)
```

Where `figId` is not a member of the returned list.

© 2020 All Rights Reserved.

dbAddFigToNet

```
dbAddFigToNet(
     d_figId
     d_netId
     )
     => t / nil
```

Description

Assigns the given figure object to a specified net object. If the figure is already associated with a different net, the figure will be detached from that net before being assigned to the given net object.

Arguments

d_figId	The given figure object.
d_netId	The specified net object.

Value Returned

`t`	The figure object was assigned.
`nil`	The figure object was not assigned.

Example

```
dbAddFigToNet(figId netId)
```

When the above execution is successful, you can verify that the figure is added using

```
netId~>figs=>list(figs...)
```

Where `figId` will be a member of the returned list.

© 2020 All Rights Reserved.

dbGetNetVoltageRange

```
dbGetNetVoltageRange(
     d_netId
     )
     => l_voltages / nil
```

Description

Returns the minimum and maximum voltages on a net.

Arguments

d_netId	The net ID whose voltages must be returned.

Value Returned

l_voltages	The list of floating numbers representing the minimum and maximum voltages.
nil	The specified net ID is incorrect.

Example

```
dbGetNetVoltageRange(netId) => (0.5 2.5)
```


© 2020 All Rights Reserved.

dbSetNetVoltageRange

```
dbSetNetVoltageRange(
     d_netId
     n_minVoltage
     n_maxVoltage
     )
     => t / nil
```

Description

Sets the minimum and maximum voltages on the specified net.

Arguments

d_netId	The net ID whose minimum and maximum voltages needs to be set.
n_minVoltage	The minimum voltage value to be set for the specified net ID.
n_maxVoltage	The maximum voltage value to be set for the specified net ID.

Value Returned

`t`	The specified minimum and maximum voltages was set on the specified net.
`nil`	The specified minimum and maximum voltages was not set on the specified net.

Example

```
dbSetNetVoltageRange(netId 0.5 2.5) => t
```


© 2020 All Rights Reserved.

dbUnsetNetVoltageRange

```
dbUnsetNetVoltageRange(
     d_netId
     )
     => t / nil
```

Description

Resets the minimum and maximum voltages on a net to their default values.

Arguments

d_netId	The net ID whose minimum and maximum voltages must be set to their default values.

Value Returned

`t`	The default voltages were reset on the specified net ID.
`nil`	The default voltages were not reset on the specified net ID.

Example

```
dbUnsetNetVoltageRange(netId) => t
```


© 2020 All Rights Reserved.

dbGetNetVoltageRangeSource

```
dbGetNetVoltageRangeSource(
     d_netId
     )
     => t_source / nil
```

Description

Returns the voltage range source of the specified net ID.

Arguments

d_netId	The net ID whose voltage range source must be returned.

Value Returned

t_source	The text string representing the voltage range source of the net ID.
nil	There is no voltage range source for the specified net ID.

Example

```
dbGetNetVoltageRangeSource(netId) => "user"
dbGetNetVoltageRangeSource(netId) => "simulation"
```


© 2020 All Rights Reserved.

dbSetNetVoltageRangeSource

```
dbSetNetVoltageRangeSource(
     d_netId
     t_source
     )
     => t / nil
```

Description

Sets the voltage range source on the specified net ID.

Arguments

d_netId	The net ID whose voltage range source must be set.
t_source	The voltage range source to be set on the specified net ID.

Value Returned

`t`	The voltage range source was set successfully on the specified net ID.
`nil`	The voltage range source was not set on the specified net ID.

Example

```
dbSetNetVoltageRangeSource(netId "simultion") => t
```


© 2020 All Rights Reserved.

dbUnsetNetVoltageRangeSource

```
dbUnsetNetVoltageRangeSource(
     d_netId
     )
     => t / nil
```

Description

Resets the voltage range source of the specified net ID to its default value.

Arguments

d_netId	The net ID whose voltage range source must be set to its default value.

Value Returned

t	The default voltage range source was reset successfully on the specified net ID.
nil	The default voltage range source was not reset on the specified net ID.

Example

```
dbUnsetNetVoltageRangeSource(netId) => t
```


© 2020 All Rights Reserved.

dbGetCellViewNetVoltageRange

```
dbGetCellViewNetVoltageRange(
     d_cvId
     )
     => l_voltages / nil
```

Description

Returns the default minimum and maximum net voltages on the specified cellview ID.

Arguments

d_cvId	The cellview ID whose minimum and maximum default net voltages must be returned.

Value Returned

l_voltages	The list of two floating numbers representing the minimum and maximum default net voltages on the cellview ID.
nil	The specified cellview ID is incorrect.

Example

```
dbGetCellViewNetVoltageRange(cvId) => (1.9 2.2)
```


© 2020 All Rights Reserved.

dbSetCellViewNetVoltageRange

```
dbSetCellViewNetVoltageRange(
     d_cvId
     n_minVoltage
     n_maxVoltage
     )
     => t / nil
```

Description

Sets the default minimum and maximum net voltages on a cellview ID.

Arguments

d_cvId	The cellview ID whose minimum and maximum default net voltages must be set.
n_minVoltage	The default minimum net voltage value to be set on the specified cellview ID.
n_maxVoltage	The default maximum net voltage value to be set on the specified cellview ID.

Value Returned

`t`	The default minimum and maximum net voltages were successfully set on the specified cellview ID.
`nil`	The default minimum and maximum net voltages were not set on the specified cellview ID.

Example

```
dbSetCellViewNetVoltageRange(cvId 1.5 3.5) => t
```


© 2020 All Rights Reserved.

dbIsCellViewNetVoltageRangeSet

```
dbIsCellViewNetVoltageRangeSet(
     d_cvId
     )
     => t / nil
```

Description

Ensures that the voltage range is set on a cellview.

Arguments

d_cvId	Database ID of a cellview.

Value Returned

t	The voltage range is set on the specified cellview.
nil	The voltage range is not on the specified cellview.

Example

```
dbIsCellViewNetVoltageRangeSet(cvId) => t
```


© 2020 All Rights Reserved.

dbIsNetVoltageRangeSet

```
dbIsNetVoltageRangeSet(
     d_netId
     )
     => t / nil
```

Description

Ensures that the voltage range is set on a net.

Arguments

d_netId	Database ID of a net.

Value Returned

t	The voltage range is set on the specified net.
nil	The voltage range is not set on the specified net.

Example

```
dbIsNetVoltageRangeSet(netId) => nil
```


© 2020 All Rights Reserved.

dbUnsetCellViewNetVoltageRange

```
dbUnsetCellViewNetVoltageRange(
     d_cvId
     )
     => t / nil
```

Description

Resets the voltage range on a cellview to its default value.

Arguments

d_cvId	Database ID of a cellview.

Value Returned

t	The default voltage range is reset on the specified cellview.
nil	The default voltage range is not rest on the specified cellview.

Example

```
dbUnsetCellViewNetVoltageRange(cvID)=> t
```


© 2020 All Rights Reserved.

dbGetNetPowerDomain

```
dbGetNetPowerDomain(
     dd_netId
     )
     => l_list / nil
```

Description

Returns the power domain associated with the specified net. A power domain can be a single string or a list of strings.

Arguments

dd_netId	ID of the net.

Value Returned

l_list	The power domain.
nil	No power domain is associated with the specified net.

Example

```
dbGetNetPowerDomain( netId )
> ("powerDomain1" "powerDomain2")
```


© 2020 All Rights Reserved.

dbSetNetPowerDomain

```
dbSetNetPowerDomain(
     dd_netId
     l_list
     )
     => t / nil
```

Description

Sets the power domain for the specified net.

Arguments

dd_netId	ID of the net.
l_list	The power domain; you can specify a single string or a list of strings.

Value Returned

t	The power domain was set successfully for the specified net.
nil	The power domain was not set for the specified net.

Example

```
dbSetNetPowerDomain( netId "powerDomain" )
> true
dbSetNetPowerDomain( netId list( "powerDomain1"  "powerDomain2" ))
> true
```


© 2020 All Rights Reserved.

dbGetPowerDomainClusters

```
dbGetPowerDomainClusters(
     d_cellViewId
     )
     => l_cluster / nil
```

Description

Retrieves SKILL list of clusters in the specified cellView. Only clusters with clusterBoundary having power domain attribute are returned. Each entry contains the cluster name, cluster boundary name and voltage value.

Arguments

d_cellViewId	Specifies the cellview from which data needs to be retrieved.

Value Returned

l_cluster	`("cluster1" "cluster2"... "clusterN")`
	List containing multiple entries. Each entry is list of:
	`("clusterName" "clusterBoundaryName" "maxVoltage")`

`nil`	The specified cellview contains no cluster with the power domain attribute.

Example

In this example, the design contains three clusters with a cluster boundary. Out of these, two clusters with clusterBoundary have the power domain attribute.

The following are the cluster details:

```
cluster 'powerDomain1' with boundary 'powerDomain1' Voltage: 3.2
cluster 'powerDomain2' with boundary 'powerDomain2' Voltage: 2.5
cluster 'NoPowerDomain' with boundary 'NoPowerDomain'

cv = geGetEditCellView()

dbGetAllPowerDomainClusters(cv)
```


© 2020 All Rights Reserved.

```
(("powerDomain1" "powerDomain1" 3.2)
     ("powerDomain2" "powerDomain2" 2.5)
)
```


© 2020 All Rights Reserved.

dbRenameNet

```
dbRenameNet(
     d_netId
     t_newNetName
     )
     => t / nil
```

Description

This function changes the name of the net pointed by `netId` to the `newNetName`.

Arguments

d_netId	Specifies the ID of the net for which you want to change the name.
t_newNetName	Specifies the new name of the net.

Value Returned

`t`	Returns `t` if the new name is successfully changed.
`nil`	Returns `nil` if the new name cannot be changed.

Example

```
net = dbCreateNet(cv "net1")
dbRenameNet(net "newNet")
```


© 2020 All Rights Reserved.

dbIsCellViewPhysicalOnly

```
dbIsCellViewPhysicalOnly(
     d_cvId
     )
     => t / nil
```

Description

Checks whether the connectivity detail of a cellveiw is visible only to the block domain or to both the block domain and the OpenAccess module domain.

Arguments

d_cvId	Database ID of a cellview.

Value Returned

`t`	The cellview is visible to the block domain only.
`nil`	The cellview is visible to both block and module domains.

Example

```
dbIsCellViewPhysicalOnly(cvId)
nil
```

The cellview `cvid` is visible to the block domain only.

© 2020 All Rights Reserved.

dbUnsetCellViewPhysicalOnly

```
dbUnsetCellViewPhysicalOnly(
     d_cvId
     )
     => t / nil
```

Description

Unsets the physical-only attribute of the specified cellview, allowing connectivity to be visible in the OpenAccess module domain.

Arguments

d_cvId	Database ID of a cellview.

Value Returned

`t`	Physical-only attribute of the specified cellview is unset.
`nil`	The operation fails.

Example

```
dbUnsetCellViewPhysicalOnly(cvId)
t
```

Physical-only attribute of the cellview `cvid` is unset.

© 2020 All Rights Reserved.

Equivalent Connectivity Time Functions

The following database functions perform operations like setting, retrieving, and removing equivalent connectivity time on a cellview.

- dbCellViewHasEquivalentConnectivityTime
- dbGetCellViewEquivalentConnectivityTime
- dbSetCellViewEquivalentConnectivityTime
- dbUnsetCellViewEquivalentConnectivityTime

dbCellViewHasEquivalentConnectivityTime

```
dbCellViewHasEquivalentConnectivityTime(
     d_cvId
     )
     => t / nil
```

Description

Checks whether the equivalent connectivity time is set on the specified cellview.

Arguments

d_cvId	Database ID of a cellview.

Value Returned

`t`	A valid equivalent connectivity time is set on the specified cellview.
`nil`	Equivalent connectivity time is not set on the specified cellview.

Example

```
dbCellViewHasEquivalentConnectivityTime(cvId) => nil
```

Confirms that the equivalent connectivity time is not set on the cellview `cvId`.

© 2020 All Rights Reserved.

dbGetCellViewEquivalentConnectivityTime

```
dbGetCellViewEquivalentConnectivityTime
     d_cvId
     )
     t_timeString / nil
```

Description

Retrieves the equivalent connectivity time that is set on the specified cellview.

Arguments

d_cvId	Database ID of a cellview

Value Returned

t_timeString	Equivalent connectivity time that is set on the cellview.
nil	No equivalent connectivity time is set.

Example

```
dbGetCellViewEquivalentConnectivityTime(cvId) => "Feb 17 12:01:27 2016"
```

Equivalent connectivity time `Feb 17 12:01:27 2016` is returned for the cellview *cvId*.

© 2020 All Rights Reserved.

dbSetCellViewEquivalentConnectivityTime

```
dbSetCellViewEquivalentConnectivityTime
     d_cvId
     t_timestring
     )
     => t / nil
```

Description

Sets the equivalent connectivity time on the specified cellview.

Arguments

d_cvId	Database ID of a cellview
t_timeString	Equivalent connectivity time to be set on the cellview.

Value Returned

`t`	Equivalent connectivity time is set on the specified cellview
`nil`	Failed to set the equivalent connectivity time.

Example

```
dbSetCellViewEquivalentConnectivityTime(cvId, timeVal) => t
```

Equivalent connectivity time `timeVal` is set on the cellview `cvId`.

© 2020 All Rights Reserved.

dbUnsetCellViewEquivalentConnectivityTime

```
dbUnsetCellViewEquivalentConnectivityTime(
     d_cvId
     )
     => t / nil
```

Description

Removes equivalent connectivity time information from the specified cellview.

Arguments

d_cvId	Database ID of a cellview

Value Returned

`t`	Equivalent connectivity time is removed from the specified cellview
`nil`	Failed to remove the equivalent connectivity time setting.

Example

```
dbUnsetCellViewEquivalentConnectivityTime(cvId) => t
```

Equivalent connectivity time information is removed from the cellview `cvId`.

© 2020 All Rights Reserved.

Connectivity Access Functions

A net contains one or more signals. A signal can be in more than one net. A net named *A<0:1>* contains two signals, *A<0>* and *A<1>*, while a net named *A<1*2>* contains two references to one signal, *A<1>*. The syntax for net names is described in more detail earlier in this chapter.

This section contains a group of database functions for accessing the connectivity information between nets, signals, and terminals.

- dbDeleteEmptyNet
- dbFindNetByName
- dbFindTermByName
- dbFindSigByName
- dbFindSigByAlias
- dbGetMemNetSig
- dbGetMemNetSigName
- dbGetNetTerms
- dbGetSigNameMemNets
- dbIsSigNameGlobal
- dbGetMemName
- dbExpandToMemNameExpr
- dbGetMaxNumBit
- dbGetNameNumBit
- dbMergeSignal
- dbMoveChildNet
- dbMovePinToNet

© 2020 All Rights Reserved.

dbDeleteEmptyNet

```
dbDeleteEmptyNet(
     d_netId
     )
     => t / nil
```

Description

Deletes a net that is not associated with other objects.

Arguments

d_netId	The database ID of the net to be deleted.

Value Returned

`t`	The net was successfully deleted.
`nil`	An error occurred during execution.

Example

```
dbDeleteEmptyNet(netId)
```


© 2020 All Rights Reserved.

dbFindNetByName

```
dbFindNetByName(
     d_cellView
     t_name
     )
     => d_net / nil
```

Description

Retrieves a net in a cellview.

Arguments

d_cellView	Cellview within which the net is located.
t_name	Name of the net.

Value Returned

d_net	The net named *t_name* if found.
nil	The net is not retrieved.

Example

```
net = dbFindNetByName(cellview "FindMe")
```


© 2020 All Rights Reserved.

dbFindTermByName

```
dbFindTermByName(
     d_cellView
     t_name
     )
     => d_term / nil
```

Description

Finds a terminal in a cellview.

Arguments

d_cellView	Name of the cellview.
t_name	Name of the terminal to search for.

Value Returned

d_term	The terminal.
nil	The terminal is not found.

Example

```
term = dbFindTermByName(cellview "FindMe")
```


© 2020 All Rights Reserved.

dbFindSigByName

```
dbFindSigByName(
     d_cellView
     t_name
     )
     => d_sig / nil
```

Description

Finds a signal in a cellview.

Arguments

d_cellView	Name of the cellview.
t_name	Name of the signal.

Value Returned

d_sig	The signal found.
nil	The signal is not found.

Example

```
sig = dbFindSigByName(cellview "VDD")
```


© 2020 All Rights Reserved.

dbFindSigByAlias

```
dbFindSigByAlias(
     d_cellView
     t_name
     )
     => d_sig / nil
```

Description

Finds a signal in a cellview by an alias.

Possible alias names of a signal are the corresponding names of the bits of the member nets this signal belongs to. signal aliases might be different from the signal name because two different nets carrying two different sets of signals can be merged into one net.

Arguments

d_cellView	Cellview the signal is residing within.
t_name	The alias name for the signal.

Value Returned

d_sig	The signal found.
nil	The signal is not found.

Example

```
sig = dbFindSigByAlias(cellview "PWR")
```


© 2020 All Rights Reserved.

dbGetMemNetSig

```
dbGetMemNetSig
     dbGetMemNetSig( ( d_net x_index ) )
          => d_sig / nil

     dbGetMemNetSig( d_net x_index )
          => d_sig / nil
```

Description

Retrieves the signal carried by bit number *x_index* of net *d_net*.

The pair (*d_net x_index*) is called the `member net` of the signal. A signal can be carried by one or many `member nets`. The function accepts the arguments separately or as a list.

Arguments

d_net	A net carrying the signal.
x_index	An integer representing a bit number in the net *d_net*.

Value Returned

d_sig	The signal carried by the specified member net.
`nil`	The member net specification is invalid.

Examples

Create a net carrying two signals “a” and “b”.

```
net = dbCreateNet(cellview "a,b")
```

Get signal “a” then signal “b” from net “a, b.”

```
sigA = dbGetMemNetSig(list(net 0))
sigB = dbGetMemNetSig(net 1)
```


© 2020 All Rights Reserved.

dbGetMemNetSigName

```
dbGetMemNetSigName
     dbGetMemNetSigName( d_net x_index )
          => t_name / nil

     dbGetMemNetSigName( ( d_net x_index ) )
          => t_name / nil
```

Description

Similar to `dbGetMemNetSig`, except it returns the name of the signal instead of the signal object.

Arguments

d_net	A net carrying the signal.
x_index	An integer representing a bit number in the net *d_net*.

Value Returned

t_name	The name of the signal carried by the specified member net.
nil	The member net specification is invalid.

Examples

Create a net carrying two signals “a” and “b”.

```
net = dbCreateNet(cellview "a,b")
```

Get signal “a” then signal “b” from net “a, b”.

```
sigNameA = dbGetMemNetSigName(list(net 0))
sigNameB = dbGetMemNetSigName(net 1)
```


© 2020 All Rights Reserved.

dbGetNetTerms

```
dbGetNetTerms(
     d_net
     )
     => d_term / nil
```

Description

Returns the IDs of the terminals attached to the specified net.

Arguments

d_net	The name of the net.

Value Returned

d_term	The ID of any terminal attached to the net.
nil	No terminals are attached to the net.

© 2020 All Rights Reserved.

dbGetSigNameMemNets

```
dbGetSigNameMemNets(
     d_cellView
     t_sigName
     )
     => ( (t_netName x_index) ...) / nil
```

Description

Lists the member nets within the cellview that contain the signal.

The member net is output as a list of lists, each sublist containing the name of the net and the index representing the net member containing the signal.

Arguments

d_cellView	Cellview where signal *t_sigName* resides.
t_sigName	The name of the signal.

Value Returned

t_netName x_index	The name of the net and the index representing the net member containing the signal.
nil	signal *t_sigName* does not exist in cellview *d_cellView*.

Example

Creates a net containing two s “a” and “b.”

```
net = dbCreateNet(cellview "a,b")
```

Gets member net for signal “a.”

```
dbGetSigNameMemNets(cellview "a")
=> (("a,b" 0))
```


© 2020 All Rights Reserved.

dbIsSigNameGlobal

```
dbIsSigNameGlobal(
     d_cellView
     t_name
     )
     => t / nil
```

Description

Tests if the signal *t_name* is a global signal.

Arguments

d_cellView	Cellview where signal *t_name* resides.
t_name	The name of the signal.

Value Returned

`t`	signal *t_name* is global.
`nil`	The signal is not global.

Example

```
isGlobal = dbIsSigNameGlobal(cellview "VDD!")
```


© 2020 All Rights Reserved.

dbGetMemName

```
dbGetMemName(
     t_name
     x_index
     )
     => t_memName
```

Description

Retrieves the name of the *x_index* member from *t_name*.

The members are numbered in increasing order from left to right starting from zero. The name defined by *t_name* is parsed according to the syntax rules of net and terminal names (of which instance names is a subset). If no errors occur during parsing and the name is valid, a string defined by *t_memName* is returned that corresponds to the *n*th member defined in the name (where *n* is *x_index).* In case of an error, the member name returned is composed of *t_name* suffixed by <*x_index*>.

Arguments

t_name	Name string adhering to the net, terminal, or instance name syntax.
x_index	Integer representing member to retrieve.

Value Returned

t_memName	The name of the *x_index* member from *t_name*. In case of an error, the member name returned is composed of *t_name* suffixed by <*x_index*>.

Example

In the following example, member 0 refers to one<2>, member 1 refers to one<3>, and member 2 refers to four<5>.

```
memName = dbGetMemName("one<2:3>,four<5>" 2)
```


© 2020 All Rights Reserved.

dbExpandToMemNameExpr

```
dbExpandToMemNameExpr(
     t_bundleName
     )
     => l_memNameExpr
```

Description

Expands a bundle name to a list of member-name-expression strings.

Member-name-expression strings are returned in the following form:

```
baseName[<vector-expression>]
```

Arguments

t_bundleName	Valid name expression.

Value Returned

l_memNameExpr	List of member-name-expression strings contained in the specified bundle.

Example

```
dbExpandToMemNameExpr("A,B<3:0>,<*2>D<1>,D<2>")
=> ("A" "B<3:0>" "D<1>" "D<2>")
```


© 2020 All Rights Reserved.

dbGetMaxNumBit

```
dbGetMaxNumBit(
     )
     => x_numbit
```

Description

Returns the maximum width allowed for a net.

Arguments

None

Value Returned

x_numbit	The maximum net width, expressed as an integer.

© 2020 All Rights Reserved.

dbGetNameNumBit

```
dbGetNameNumBit(
     t_name
     )
     => x_numBit
```

Description

Parses a string and returns the number of bits in the net, terminal, or instance named.

The name defined by *t_name* is parsed according to the syntax rules of net and terminal names (of which instance names is a subset).

Arguments

t_name	Name string adhering to the net, terminal, or instance name syntax.

Value Returned

x_numBit	Number of bits in *t_name*. Returns zero if an error is detected.

Example

The following name returns 3 as the number of bits.

```
numBit = dbGetNameNumBit("one<2:3>,four<5>"
```


© 2020 All Rights Reserved.

dbMergeSignal

```
dbMergeSignal(
     d_sig1Id
     d_sig2Id
     )
     => d_sig1Id / nil
```

Description

Merges two signals into one signal.

Arguments

d_sig1Id	The database ID of the first signal to be merged.
d_sig2Id	The database ID of the second signal to be merged.

Value Returned

d_sig1Id	Return the database ID of the single merged signal.
nil	An error occurred during execution.

Example

```
dbMergeSignal(sigId1 sigId2)
```


© 2020 All Rights Reserved.

dbMoveChildNet

```
dbMoveChildNet(
     d_childId
     d_parentId
     )
     => t / nil
```

Description

Moves a net to be the child of given parent net, both should be on same terminal.

Arguments

d_childId	The database ID of the net you want to be the child.
d_parentId	The database ID of the net you want to be the parent.

Value Returned

`t`	The net was successfully moved.
`nil`	An error occurred during execution.

Example

```
dbMoveChildNet(d_childId d_parentId)
```


© 2020 All Rights Reserved.

dbMovePinToNet

```
dbMovePinToNet(
     d_pinId
     d_netId
     )
     => t / nil
```

Description

Moves a pin to the specified net.

Arguments

d_pinId	The database ID of the pin you want move.
d_netId	The database ID of the net.

Value Returned

`t`	The pin was successfully moved.
`nil`	An error occurred during execution.

Example

```
dbMovePinToNet(pinId netId)
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

6

Pin Database Access Functions

This chapter includes the following topics:

- Pin Connection Functions
- Pin Group Guide Functions

© 2020 All Rights Reserved.

Pin Connection Functions

Pin models allow multiple figures per pin, as well as multiple pins per terminal.

Figures (instances, shapes and vias) can be added to a pin. Pins can be created with 0 to *n* figures attached. When writing SKILL code, pins can be initially create with no figures. You can then add or remove figures from the pins in subsequent calls.

Important

Do not leave pins without a figure attached. Creating a pin without a figure is intended to be a transitional state, not a permanent, or saved state.

Note: You can indicate a must connect by setting the `mustJoin` attribute on the terminals.

External, Strong, and Weak Pin Connections

In IC6.1.x and later, External, Strong and Weak pin connections are determined by the geometry on the Pin and the associations between the Pins and the Terms, and not by Grouping (as was the case in IC 5.1.x).

A terminal (Term) represents a logical connection. A Pin represents the physical implementation of a logical connection. A terminal may have one or more Pins. A pinFig represents a specific connection point in the implementation of the Pin. Typically, pinFigs are shapes, but they can also be other geometrical objects (such as instances). In the rest of the section, we will use the term pinFigs and shapes interchangeably for simplicity.

All of the pinFigs (shapes) on the same Pin are strongly connected to one another. These shapes represent a set of low resistance connection points to the same logical terminal. The router may chose to route the connection through the strongly connected pinFigs at the router's discretion. A typical example of strongly connected pinFigs is the shapes on either side of a power bus on the top and the bottom of a device.

A terminal may have multiple Pins. The pinFigs (shapes) that are on different Pins, which are on the same terminal are weakly connected to one another. These shapes represent a set of high resistance connection points to the same logical terminal. For example, the two ends of a poly gate of a transistor. The router may not continue a route through weakly connected pinFigs.

© 2020 All Rights Reserved.

External Pin connections are modeled by collections of MustJoin Terms. The router must externally connect together the pinFigs that are on Pins which are on Terms in the same "MustJoin" set.

Figures on the same pin are strongly connected: fig1 and fig2

Figures on different pins but on the same terminal are weakly connected: {fig1 and fig3} {fig2 and fig3}

Figures on terminals with different nets, and the terminals have the mustJoin attribute are must connect: fig4 and fig5

Figures on different terminals on the same net are shorted: fig5 and fig6

This chapter describes the following pin connection database SKILL functions:

- dbCreateStrongGroup
- dbExternallyConnectPins
- dbGetPinConnection
- dbGetStrongGroups
- dbHasExternalConnections
- dbHasMustConnections
- dbMergeStrongGroups
- dbMustConnectPins
- dbStronglyConnectPins
- dbTransformAccessDirection
- dbWeaklyConnectPins

© 2020 All Rights Reserved.

- dbAddFigsToPin
- dbAddFigToPin
- dbGetPinFigAccessDirection
- dbGetPinFigName
- dbGetPinFigPlacementStatus
- dbSetPinFigAccessDirection
- dbSetPinFigName
- dbSetPinFigPlacementStatus
- dbSetTermMustJoin
- dbSubFigFromPin
- dbSubFigsFromPin
- dbUnsetTermMustJoin

© 2020 All Rights Reserved.

dbCreateStrongGroup

```
dbCreateStrongGroup(
     l_pinList
     [ s_groupConnection ]
     )
     => t / nil
```

Description

This function is obsolete in this release. Strong connections are defined exclusively by the geometry on the pin. All the pinFigs on the same pin are strongly connected to one another.

© 2020 All Rights Reserved.

dbExternallyConnectPins

```
dbExternallyConnectPins(
     l_pinList
     )
     => t / nil
```

Description

This function is obsolete in this release. External connections are modeled by pinFigs on Pins on Terms in the same MustJoin set. See dbSetTermMustJoin.

© 2020 All Rights Reserved.

dbGetPinConnection

```
dbGetPinConnection(
     d_pin1
     d_pin2
     )
     => dbcExternallyConnected / dbcWeaklyConnected /
          dbcStronglyConnected / nil
```

Description

Given two pins, it returns a symbol representing the connectivity between two pins.

The return value is one of the following:

0	dbcExternallyConnected
1	dbcWeaklyConnected
2	dbcStronglyConnected
nil	no connection

The connectivity symbols also have numeric values. The higher the value, the stronger the connection.

Arguments

d_pin1, d_pin2 The two pins you are interested in.

Value Returned

dbcExternallyConnected

Symbol returned when *d_pin1* and *d_pin2* are externally (must) connected. This happens when *d_pin1* and *d_pin2* are different pins on different terminals and the terminals are in the same MustJoin set.

dbcWeaklyConnected

Symbol returned when *d_pin1* and *d_pin2* are weakly connected. This happens when *d_pin1* and *d_pin2* are different pins on the same terminal.

© 2020 All Rights Reserved.

dbcStronglyConnected

Symbol returned when *d_pin1* and *d_pin2* are strongly connected. This happens when *d_pin1* and *d_pin2* are the same pin on the same terminal.

nil

Returned when *d_pin1* and *d_pin2* are not connected. This occurs when *d_pin1* and *d_pin2* are on different terminals.

© 2020 All Rights Reserved.

dbGetStrongGroups

```
dbGetStrongGroups(
     d_object
     )
     => ~>name
```

Description

This function is obsolete in this release. Strong connections are modeled by multiple pinFigs on the same pin.

© 2020 All Rights Reserved.

dbHasExternalConnections

```
dbHasExternalConnections(
     d_connectivityObject
     )
     => t / nil
```

Description

Reports if the given net or terminal has one or more pins that are externally connected. `dbHasExternalConnections(connectivityObject)` is equivalent to `dbHasMustConnections(connectivityObject)`.

Arguments

d_connectivityObject

Specifies the *connectivityObject* which can either be a net (*d_net*), a terminal (*d_terminal*), or a pin (*d_pin*).

Value Returned

`t`	The given net or terminal contains one or more externally connected pins.
`nil`	The given net or terminal does not contain one or more externally connected pins.

© 2020 All Rights Reserved.

dbHasMustConnections

```
dbHasMustConnections(
     anObject
     )
     => t
```

Description

Reports if the given net or terminal has one or more pins that are must connected. `dbHasMustConnections(anObject)` is equivalent to `dbHasExternalConnections(connectivityObject)`.

Arguments

anObject	Can be a terminal, a net, or a pin.

Value Returned

t	The given object, `anObject`, has one or more pins that are must connected. `anObject` can be a terminal, a net, or a pin. If `anObject` is a net or a pin, then `anObject`'s terminal is checked for must connections.

© 2020 All Rights Reserved.

dbMergeStrongGroups

```
dbMergeStrongGroups(
     l_pinList
     )
     => t / nil
```

Description

This function is obsolete in this release. Strong connections are modeled by multiple pinFigs on the same pin.

© 2020 All Rights Reserved.

dbMustConnectPins

```
dbMustConnectPins(
     l_pinList
     )
     => t / nil
```

Description

This function is obsolete in this release. External Pin connections are modeled by collections of MustJoin Terms. See dbSetTermMustJoin.

© 2020 All Rights Reserved.

dbStronglyConnectPins

```
dbStronglyConnectPins(
     l_pinList
     )
     => t / nil
```

Description

This function is obsolete in this release. Strong connections are modeled by multiple pinFigs on the same pin.

© 2020 All Rights Reserved.

dbTransformAccessDirection

```
dbTransformAccessDirection(
     x_dirCode
     t_orient
     )
     => x_dirCode / nil
```

Description

Transforms the given direction by the specified orientation and returns the corresponding access direction.

Arguments

x_dirCode	Access direction expressed as an integer. `4` (left), `8` (right), `1` (top), `2` (bottom), any summation of those values, and `0` (none). `nil` means unspecified and not 0.
t_orient	Specifies the orientation.

Value Returned

x_dirCode	Access direction expressed as an integer. `4` (left), `8` (right), `1` (top), `2` (bottom), any summation of those values, and `0` (none). `nil` means unspecified and not 0.
`nil`	Returns `nil` when the direction is unspecified.

Example

```
dbTransformAccessDirection(4 "R270")
```


© 2020 All Rights Reserved.

dbWeaklyConnectPins

```
dbWeaklyConnectPins(
     l_pinList
     )
     => t / nil
```

Description

This function is obsolete in this release. Weak pin connections are modeled by pinFigs on different pins on the same terminal.

© 2020 All Rights Reserved.

dbAddFigsToPin

```
dbAddFigsToPin(
     d_pinId
     ld_figId
     )
     => t / nil
```

Description

Adds a list of figures to the given pin.

Arguments

d_pinId	The database ID of the pin.
ld_figId	Specifies a list of figure database IDs.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if the pin ID or figure IDs are invalid.

© 2020 All Rights Reserved.

dbAddFigToPin

```
dbAddFigToPin(
     d_pinId
     d_figId
     )
     => t / nil
```

Description

Adds the specified figure to the given pin.

If the pinFig has the placement status or access direction set and the pinFig is the first fig of the pin, then the pin is updated with the attributes of its pinFig (if the data is set on the pinFig).

Arguments

d_pinId	The database ID of the pin.
d_figId	The database ID of the figure.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if the pin or figure ID is invalid.

© 2020 All Rights Reserved.

dbGetPinFigAccessDirection

```
dbGetPinFigAccessDirection(
     d_figId
     )
     => t_pinFigAccessDir / nil
```

Description

Returns the access direction for the specified pin figure.

If the access direction of the figure is equivalent to the access direction stored on the pin associated with figure, then a value is not stored on the figure and the returned value is retrieved from the pin.

Arguments

d_figId	The database ID of the figure that is attached to the pin.

Value Returned

t_pinFigAccessDir	A string list indicating the allowed access direction of the pin figure.
nil	nil is returned if the pin or figure IDs are invalid.

Example

```
foreach( fig setof(shape geGetEditCellView()~>shapes shape~>pin != nil)
    printf("Pin fig access direction: %L\n" dbGetPinFigAccessDirection(fig)
)
```


© 2020 All Rights Reserved.

dbGetPinFigName

```
dbGetPinFigName(
     d_figId
     )
     => t_pinFigName / nil
```

Description

Returns the name of the figure specified by the pin figure ID.

If the name of the figure is equivalent to name of the pin associated with figure, then a value is not stored on the figure and the returned value is retrieved from the pin.

Arguments

d_figId	The database ID of the figure that is attached to the pin.

Value Returned

t_pinFigName	A string indicating the name of the pin figure.
`nil`	`nil` is returned if the pin or figure IDs are invalid.

Example

```
foreach( fig setof(shape geGetEditCellView()~>shapes shape~>pin != nil)
   printf("Pin fig name: %s\n" dbGetPinFigName(fig)
   )
)
```


© 2020 All Rights Reserved.

dbGetPinFigPlacementStatus

```
dbGetPinFigPlacementStatus(
     d_figId
     )
     => t_pinFigPlacement / nil
```

Description

Returns the placement status for the specified pin figure.

If the placement status of the figure is equivalent to the placement status stored on the pin associated with figure, then a value is not stored on the figure and the returned value is retrieved from the pin.

Arguments

d_figId	The database ID of the figure that is attached to the pin.

Value Returned

t_pinFigPlacement	A string list indicating the placement status of the pin figure. Valid values: `placed`, `unplaced`, `firm`, `fixed`
`nil`	`nil` is returned if the pin or figure IDs are invalid.

Example

```
foreach( fig setof(shape geGetEditCellView()~>shapes shape~>pin != nil)
   status = dbGetPinFigPlacementStatus(fig)
   if( status != nil
      printf("Pin fig placement status: %s\n" status)
   )
)
```


© 2020 All Rights Reserved.

dbSetPinFigAccessDirection

```
dbSetPinFigAccessDirection(
     d_pinFigId
     lt_accessDir
     )
     => t / nil
```

Description

Specifies the access direction for the given pin figure.

If there is an override value on the figure which is different from value stored on the pin associated with the figure, and the figure is the only figure of its pin then the pin~>accessDir is updated; otherwise, the value is stored locally with the figure.

Arguments

d_figId	The database ID of the figure that is attached to the pin.
d_figId	A string list which indicates one or more access directions for the specified pin. Valid values: `list("left" "right" "bottom" "top")`

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if the pin or figure IDs are invalid.

Example

```
dbSetPinFigAccessDirection(figId list("left" "right")) t
```


© 2020 All Rights Reserved.

dbSetPinFigName

```
dbSetPinFigName(
     d_pinId
     t_pinName
     )
     => t / nil
```

Description

Specifies the name for the given pin figure.

If there is an override value on the figure which is different from the value stored on the pin associated with the figure, the value is stored locally with the figure.

Arguments

d_pinId	The database ID of the figure attached to the pin specified by *t_pinName*.
t_pinName	The database ID of the pin.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if the fig ID is invalid.

Example

```
dbSetPinFigName(figId "MyPin") t
```


© 2020 All Rights Reserved.

dbSetPinFigPlacementStatus

```
dbSetPinFigPlacementStatus(
     d_figId
     g_placementStat
     )
     => t / nil
```

Description

Specifies the placement status for the given pin figure.

If there is an override value on the figure which is different from the value stored on the pin associated with the figure and the figure is the only figure of its pin then the pin~>status is updated; otherwise, the value is stored locally with the figure.

Arguments

d_figId	The database ID of the pin figure that is attached to the pin.
g_placementStat	A string list indicating the placement status for the pin figure. Valid values: `placed`, `unplaced`, `firm`, `fixed`, `none`

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if the pin or figure IDs are invalid.

Example

```
dbSetPinFigPlacementStatus(figId "placed") t
```


© 2020 All Rights Reserved.

dbSetTermMustJoin

```
dbSetTermMustJoin(
     d_termId
     d_mustJoinTermId
     )
     => t / nil
```

Description

Sets the connection status of the terminals specified by the terminal IDs to mustJoin.

Arguments

d_termId	The database ID of the terminal that you want to mustJoin to the terminal specified by *d_mustJoinTermId*.
d_mustJoinTermId	The database ID of the terminal that you want to mustJoin to the terminal specified by *d_termId*.

Value Returned

t	t is returned when successful.
nil	nil is returned if either of the terminal IDs are invalid.

© 2020 All Rights Reserved.

dbSubFigFromPin

```
dbSubFigFromPin(
     d_pinId
     d_figId
     )
     => t / nil
```

Description

Removes the specified figure from the given pin.

If a pin is changed from having two pin figures to a single pin figure, the placement status and access direction of the pin is updated with the attributes of its pinFig if the data is set on the pinFig.

Arguments

d_pinId	The database ID of the pin.
d_figId	The database ID of the figure to be removed from the pin.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if either the pin or fig IDs are invalid.

© 2020 All Rights Reserved.

dbSubFigsFromPin

```
dbSubFigsFromPin(
     d_pinId
     ld_figId
     )
     => t / nil
```

Description

Removes the specified list of figures from the given pin.

Arguments

d_pinId	The database ID of the pin.
ld_figId	Specifies a list of figure to be removed from a pin.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if either the pin or fig IDs are invalid.

© 2020 All Rights Reserved.

dbUnsetTermMustJoin

```
dbUnsetTermMustJoin(
     d_termId
     )
     => t / nil
```

Description

Removes the mustJoin pin connection status from the given terminal.

Arguments

d_termId	The database ID of the terminal.

Value Returned

`t`	`t` is returned when successful.
`nil`	`nil` is returned if the terminal ID is invalid.

© 2020 All Rights Reserved.

dbSetTermMustConnectAllPins

```
dbSetTermMustConnectAllPins(
     d_termId
     g_value { t / nil }
     )
     => t / nil
```

Description

Indicates whether to set the `mustConnectAllPins` attribute on the specified terminal. The terminal must be a single-bit terminal.

Arguments

d_termId	Database ID of a single-bit terminal.
g_value	Boolean value specifying whether the `mustConnectAllPins` attribute should be set on the specified terminal. Valid values: ■ `t`, if the `mustConnectAllPins` attribute should be set. ■ `nil`, if the `mustConnectAllPins` attribute should be not set.

Value Returned

`t`	The `mustConnectAllPins` attribute on the specified terminal is set to the given value.
`nil`	The terminal ID is invalid or a multi-bit terminal is specified.

Example

```
dbSetTermMustConnectAllPins(term t)
```

The `mustConnectAllPins` attribute on the terminal `term` is set to `t`.

© 2020 All Rights Reserved.

dbTermHasMustConnectAllPins

```
dbTermHasMustConnectAllPins(
     d_termId
     )
     => t / nil
```

Description

Checks whether the `mustConnectAllPins` attribute value is set on the specified single-bit terminal.

Arguments

d_termId	Database ID of a terminal.

Value Returned

`t`	The `mustConnectAllPins` attribute value is set on the specified terminal.
`nil`	The terminal ID is invalid or is a multi-bit terminal.

Example

```
dbTermHasMustConnectAllPins(termId)
=> t
```

Confirms that the `mustConnectAllPins` attribute value is set on the terminal `termId`.

© 2020 All Rights Reserved.

Pin Group Guide Functions

Pin Group Guides functions are related to Virtuoso® Floorplanner. These functions let you create pin groups and assign pin constraints to them. This feature is useful for designs with a large number of pins, such as mixed-signal designs, where it would be difficult and time-consuming to create pin constraints on individual pins. You can create pin groups, assign them to full or partial edge(s), and apply process constraints, such as layer range and spacings, to the entire pin group through a dedicated pin group guide constraint.

For more information on Floorplanner, see Creating Pin Group Guides in *Virtuoso Floorplanner User Guide.*

This chapter describes the following pin group guide database SKILL functions:

- dbCreateAreaPinGroupGuide
- dbCreateEdgePinGroupGuide
- dbFindPinGroupGuideByName
- dbGetAreaBoundaryPinGroupGuide
- dbGetAreaPinGroupGuideAreas
- dbGetFigPinGroupGuide
- dbGetPRBoundaryPinGroupGuides
- dbGetPinGroupGuideEdgeBoundary
- dbGetPinGroupGuideEdgeName
- dbGetPinGroupGuideEdgeStartOffset
- dbGetPinGroupGuideEdgeStopOffset
- dbGetPinGroupGuides
- dbGetPinPinGroupGuides
- dbAddAreaToPinGroupGuide
- dbAddFigToPinGroupGuide
- dbAddPinToPinGroupGuide
- dbSubAreaFromPinGroupGuide
- dbSubFigFromPinGroupGuide

© 2020 All Rights Reserved.

- dbSubPinFromPinGroupGuide
- dbRemovePinGroupGuideEdge
- dbSetPinGroupGuideEdge
- dbSetPinGroupGuideEdgeStartOffset
- dbSetPinGroupGuideEdgeStopOffset
- dbGetAreaBoundaryValidRoutingLayers
- dbAreaBoundaryHasLayerPriorityOrder
- dbAreaBoundaryHasValidRoutingLayers
- dbGetAreaBoundaryLayerPriorityOrder
- dbGetPinGroupGuideKeepoutSpacing
- dbGetPinGroupGuideLayerPriorityOrder
- dbGetPinGroupGuideMinPinSpacing
- dbGetPinGroupGuideMinPinWidth
- dbGetPinGroupGuideValidRoutingLayers
- dbGetPinMinPinSpacing
- dbGetPinMinPinWidth
- dbPinGroupGuideHasKeepoutSpacing
- dbPinGroupGuideHasLayerPriorityOrder
- dbPinGroupGuideHasMinPinSpacing
- dbPinGroupGuideHasMinPinWidth
- dbPinGroupGuideHasValidRoutingLayers
- dbPinHasMinPinSpacing
- dbPinHasMinPinWidth
- dbSetAreaBoundaryLayerPriorityOrder
- dbSetAreaBoundaryValidRoutingLayers
- dbSetPinGroupGuideKeepoutSpacing
- dbSetPinGroupGuideLayerPriorityOrder

© 2020 All Rights Reserved.

- dbSetPinGroupGuideMinPinSpacing
- dbSetPinGroupGuideMinPinWidth
- dbSetPinGroupGuideValidRoutingLayers
- dbSetPinMinPinSpacing
- dbSetPinMinPinWidth
- dbUnsetAreaBoundaryLayerPriorityOrder
- dbUnsetAreaBoundaryValidRoutingLayers
- dbUnsetPinGroupGuideKeepoutSpacing
- dbUnsetPinGroupGuideLayerPriorityOrder
- dbUnsetPinGroupGuideMinPinSpacing
- dbUnsetPinGroupGuideMinPinWidth
- dbUnsetPinGroupGuideValidRoutingLayers
- dbUnsetPinMinPinSpacing
- dbUnsetPinMinPinWidth

© 2020 All Rights Reserved.

dbCreateAreaPinGroupGuide

```
dbCreateAreaPinGroupGuide(
     d_cellViewId
     t_name
     )
     => d_pinGroupGuideId / nil
```

Description

Creates an area pin group guide with the specified name in the given cellview.

Arguments

d_cellViewId	The cellview in which the area pin group guide is created.
t_name	The name of the area pin group guide.

Value Returned

d_pinGroupGuideId	The pin group guide ID is created with the specified name in the given cellview.
nil	The pin group guide ID is not created.

Example

```
dbCreateAreaPinGroupGuide(cv "pg1")
=> db:0x13eaeb1b
```


© 2020 All Rights Reserved.

dbCreateEdgePinGroupGuide

```
dbCreateEdgePinGroupGuide(
     d_cellViewId
     t_name
     )
     => d_edgePinGroupGuideId / nil
```

Description

Creates an edge pin group guide with the specified name in the given cellview.

Arguments

d_cellViewId	The ID of the cellview in which the edge pin group guide is created.
t_name	The name of the edge pin group guide.

Value Returned

d_edgePinGroupGuideId	The edge pin group guide ID is created with the specified name in the given cellview.
`nil`	Returns `nil` if either an invalid cellview ID is specified or if the edge pin group guide ID is not created.

Example

```
dbCreateEdgePinGroupGuide(cv "epg")
=> db:0x13eaeb1b
```


© 2020 All Rights Reserved.

dbFindPinGroupGuideByName

```
dbFindPinGroupGuideByName(
     d_cellViewId
     t_name
     )
     => d_pinGroupGuideId / nil
```

Description

Searches for the pin group guide with the specified name in the given cellview.

Arguments

d_cellViewId	The cellview in which the pin group guide is searched.
t_name	The name of the pin group guide.

Value Returned

d_edgePinGroupGuideId	The ID of the pin group guide that is found with the specified name in the given cellview.
nil	The pin group guide ID is not found.

Example

```
dbFindPinGroupGuideByName(cv "pgg1")
```


© 2020 All Rights Reserved.

dbGetAreaBoundaryPinGroupGuide

```
dbGetAreaBoundaryPinGroupGuide(
     d_areaBoundaryId
     )
     => d_pinGroupGuideId / nil
```

Description

Returns the pin group guide associated with the specified area boundary.

Arguments

d_areaBoundaryId	The ID of the specified area boundary whose associated pin group guide needs to be returned.

Value Returned

d_pinGroupGuideId	The pin group guide ID of the specified area boundary.
`nil`	Returns `nil` if no pin group guide ID is associated with the specified area boundary.

Example

```
dbGetAreaBoundaryPinGroupGuide(areaBndyId)
=> db:0x1818de1b
```


© 2020 All Rights Reserved.

dbGetAreaPinGroupGuideAreas

```
dbGetAreaPinGroupGuideAreas(
     d_areaPinGroupGuideId
     )
     => l_areaBoundaryIds / nil
```

Description

Returns the list of area boundaries associated with the specified area pin group guide.

Arguments

d_areaPinGroupGuideId	The ID of the specified area pin group guide whose associated area boundaries need to be returned.

Value Returned

l_areaBoundaryIds	The list of area boundaries associated with the specified area pin group guide.
nil	No area boundaries are associated with the specified area pin group guide.

Example

```
dbGetAreaPinGroupGuideAreas(pinGroupGuideId)
=> list(db:0x1818daaa db:0x1818daae)
```


© 2020 All Rights Reserved.

dbGetFigPinGroupGuide

```
dbGetFigPinGroupGuide(
     d_figId
     )
     => d_pinGroupGuideId / nil
```

Description

Returns the pin group guide associated with the specified figure.

Arguments

d_figId	The ID of the specified figure whose associated pin group guide needs to be returned.

Value Returned

d_pinGroupGuideId	Returns the ID of the pin group guide associated with the specified figure.
`nil`	Returns `nil` if no pin group guide is associated with the specified figure.

Example

```
dbGetFigPinGroupGuide(figId)
=> db:0x1818de1b
```


© 2020 All Rights Reserved.

dbGetPRBoundaryPinGroupGuides

```
dbGetPRBoundaryPinGroupGuides(
     d_prBoundaryId
     )
     => l_pinGroupGuideIds / nil
```

Description

Returns the list of pin group guides associated with the specified prBoundary.

Arguments

d_prBoundaryId	The ID of the specified prBoundary whose associated pin group guides need to be returned.

Value Returned

l_pinGroupGuideIds	Returns the list of the pin group guides associated with the specified prBoundary.
`nil`	Returns `nil` if no pin group guide is associated with the specified prBoundary.

Example

```
dbGetPRBoundaryPinGroupGuides(prBoundaryId)
=> list(db:0x1818df1b db:0x1818de8a)
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideEdgeBoundary

```
dbGetPinGroupGuideEdgeBoundary(
     d_edgePinGroupGuide
     )
     => d_prBoundary / nil
```

Description

Returns the prBoundary associated with the specified edge pin group guide.

Arguments

d_edgePinGroupGuide	The specified edge pin group guide whose associated prBoundary need to be returned.

Value Returned

d_prBoundary	Returns the prBoundary associated with the specified edge pin group guide.
`nil`	Returns `nil` if no prBoundary is associated with the specified edge pin group guide.

Example

```
dbGetPinGroupGuideEdgeBoundary(epgId)
=> db:0x13eaec1a
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideEdgeName

```
dbGetPinGroupGuideEdgeName(
     d_edgePinGroupGuide
     )
     => t_name / nil
```

Description

Returns the edge name of the specified edge pin group guide.

Arguments

d_edgePinGroupGuide	The specified edge pin group guide whose edge name needs to be returned.

Value Returned

t_name	Returns the edge name of the specified edge pin group guide.
`nil`	Returns `nil` on error. For example, when an invalid edge pin group guide is specified.

Example

```
dbGetPinGroupGuideEdgeName(epgId)
=> "Edge1"
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideEdgeStartOffset

```
dbGetPinGroupGuideEdgeStartOffset(
     d_edgePinGroupGuide
     )
     => n_offset / nil
```

Description

Returns the start offset of the edge for the specified edge pin group guide.

Arguments

d_edgePinGroupGuide	The specified edge pin group guide whose start offset of the edge needs to be returned.

Value Returned

n_offset	Returns the start offset of the edge for the specified edge pin group guide.
nil	Returns nil on error. For example, when an invalid edge pin group guide is specified.

Example

```
dbGetPinGroupGuideEdgeStartOffset(epgId)
=> 1
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideEdgeStopOffset

```
dbGetPinGroupGuideEdgeStopOffset(
     d_edgePinGroupGuide
     )
     => n_offset / nil
```

Description

Returns the stop offset of the edge for the specified edge pin group guide.

Arguments

d_edgePinGroupGuide	The specified edge pin group guide whose stop offset of the edge needs to be returned.

Value Returned

n_offset	Returns the stop offset of the edge for the specified edge pin group guide.
nil	Returns nil on error. For example, when an invalid edge pin group guide is specified.

Example

```
dbGetPinGroupGuideEdgeStopOffset(epgId)
=> 2
```


© 2020 All Rights Reserved.

dbGetPinGroupGuides

```
dbGetPinGroupGuides(
     d_cellViewId
     )
     => l_pinGroupGuides / nil
```

Description

Returns the list of pin group guides associated with the specified cellview.

Arguments

d_cellViewId	The ID of the cellview whose associated pin group guides need to be returned.

Value Returned

l_pinGroupGuides	Returns the list of pin group guides associated with the specified cellview.
`nil`	Returns `nil` if no pin group guides are associated with the specified cellview.

Example

```
dbGetPinGroupGuides(cvId)
=> list(db:0x1818df1b db:0x1818de8a)
```


© 2020 All Rights Reserved.

dbGetPinPinGroupGuides

```
dbGetPinPinGroupGuides(
     d_pinId
     )
     => l_pinGroupGuideIds / nil
```

Description

Returns the list of pin group guides associated with the specified pin.

Arguments

d_pinId	The ID of the pin whose associated pin group guides need to be returned.

Value Returned

l_pinGroupGuideIds	Returns the list of pin group guides associated with the specified pin.
`nil`	Returns `nil` if no pin group guides are associated with the specified pin.

Example

```
dbGetPinPinGroupGuides(pinId)
=> list(db:0x1818df1b db:0x1818de8a)
```


© 2020 All Rights Reserved.

dbAddAreaToPinGroupGuide

```
dbAddAreaToPinGroupGuide(
     d_areaPinGroupGuideId
     d_areaBoundaryId
     )
     => t / nil
```

Description

Adds an area boundary to the specified area pin group guide.

Arguments

d_areaPinGroupGuideId	The specified area pin group guide to which the area boundary needs to be added.
d_areaBoundaryId	The ID of the specified area boundary.

Value Returned

`t`	Returns `t` if an area boundary is added successfully to the specified area pin group guide.
`nil`	Returns `nil` if an area boundary was not added to the specified area pin group guide.

Example

```
dbAddAreaToPinGroupGuide(apgId areaBndyId)
=> t
```


© 2020 All Rights Reserved.

dbAddFigToPinGroupGuide

```
dbAddFigToPinGroupGuide(
     d_pinGroupGuideId
     d_figId
     )
     => t / nil
```

Description

This function adds a via or a non-text shape to the specified pin group guide.

Arguments

d_pinGroupGuideId	The specified pin group guide.
d_figId	The via or non-text shape that needs to be added.

Value Returned

`t`	Returns `t` if the given figure is added to the pin group guide.
`nil`	Returns `nil` when one of the following error conditions occurs: ■ an invalid pin group guide ID or figure ID is specified ■ the figure ID refers to a text shape ■ the pin group guide ID and the figure ID exist in different cellviews

Example

```
dbAddFigToPinGroupGuide(epg1 pinFig)
```


© 2020 All Rights Reserved.

dbAddPinToPinGroupGuide

```
dbAddPinToPinGroupGuide(
     d_pinGroupGuideId
     d_pinId
     )
     => t / nil
```

Description

This function adds a pin to the specified pin group guide.

Arguments

d_pinGroupGuideId	The database ID of the pin group guide.
d_pinId	The pin that needs to be added.

Value Returned

`t`	Returns `t` if the pin is added to the specified pin group guide.
`nil`	Returns `nil` on error.

Example

```
dbAddPinToPinGroupGuide(epg1 pin)
```


© 2020 All Rights Reserved.

dbSubAreaFromPinGroupGuide

```
dbSubAreaFromPinGroupGuide(
     d_areaPinGroupGuideId
     d_areaBoundaryId
     )
     => t / nil
```

Description

This function removes an area boundary from the specified area pin group guide.

Arguments

d_areaPinGroupGuideId	The ID of the area pin group guide.
d_areaBoundaryId	The ID of the area boundary that needs to be removed.

Value Returned

`t`	Returns `t` if the area boundary is removed from the specified area pin group guide.
`nil`	Returned `nil` on error.

Example

```
dbSubAreaFromPinGroupGuide(apgId areaBndyId) => t
```


© 2020 All Rights Reserved.

dbSubFigFromPinGroupGuide

```
dbSubFigFromPinGroupGuide(
     d_pinGroupGuideId
     d_figId
     )
     => t / nil
```

Description

This function removes a via or a non-text shape from the specified pin group guide.

Arguments

d_pinGroupGuideId	The ID of the pin group guide.
d_figId	The ID of the via or non-text shape that needs to be removed.

Value Returned

`t`	Returns `t` if the via or the non-text shape is removed from the specified pin group guide.
`nil`	Returns `nil` on error, for example, when the via or the non-text shape is not removed from the specified pin group guide.

Example

```
dbSubFigFromPinGroupGuide(epg1 pinFig) => t
```


© 2020 All Rights Reserved.

dbSubPinFromPinGroupGuide

```
dbSubPinFromPinGroupGuide(
     d_pinGroupGuideId
     d_pinId
     )
     => t / nil
```

Description

This function removes a pin from the specified pin group guide.

Arguments

d_pinGroupGuideId	The ID of the pin group guide.
d_pinId	The ID of the pin that needs to be removed.

Value Returned

`t`	Returns `t` if the pin is removed from the specified pin group guide.
`nil`	Returned if the pin is not removed from the specified pin group guide.

Example

```
dbSubPinFromPinGroupGuide(epg1 pin)
=> t
```


© 2020 All Rights Reserved.

dbRemovePinGroupGuideEdge

```
dbRemovePinGroupGuideEdge(
     d_edgePinGroupGuide
     )
     => t / nil
```

Description

This function removes the prBoundary and edge name specification from the specified edge pin group guide.

Arguments

d_edgePinGroupGuide	The ID of the edge pin group guide.

Value Returned

`t`	Returns `t` if the prBoundary is removed from the given edge pin group guide.
`nil`	Returns `nil` in case of error, that is, when the prBoundary is not removed successfully.

Example

```
dbRemovePinGroupGuideEdge(epgId)
=> t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideEdge

```
dbSetPinGroupGuideEdge(
     d_edgePinGroupGuide
     d_prBoundary
     t_name
     )
     => t / nil
```

Description

This function specifies new or updates existing prBoundary edge information for the edge pin group guide.

Arguments

d_edgePinGroupGuide	Specifies the ID of the edge pin group guide name.
d_prBoundary	Specifies the ID of the prBoundary.
t_name	Specifies the edge name of the prBoundary that is being added. prBoundaries can have multiple edge names.

Value Returned

`t`	Returns `t` if the prBoundary edge specification for the given edge pin group guide is set or updated.
`nil`	Returns `nil` if the prBoundary edge specification for the given edge pin group guide is not set or updated.

Example

```
dbSetPinGroupGuideEdge(epgId prbId "Edge1") => t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideEdgeStartOffset

```
dbSetPinGroupGuideEdgeStartOffset(
     d_edgePinGroupGuide
     n_offset
     )
     => t / nil
```

Description

This function sets the start offset of the edge on the edge pin group guide.

Arguments

d_edgePinGroupGuide	The ID of the edge pin group guide.
n_offset	The start offset of the edge. The default offset value is 0.

Value Returned

`t`	Returns `t` if the start offset of the edge on the edge pin group guide is set.
`nil`	Returns `nil` if the start offset of the edge on the edge pin group guide is not set.

Example

```
dbSetPinGroupGuideEdgeStartOffset(epgId 5)
=> t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideEdgeStopOffset

```
dbSetPinGroupGuideEdgeStopOffset(
     d_edgePinGroupGuide
     n_offset
     )
     => t / nil
```

Description

This function sets the stop offset of the edge on the edge pin group guide.

Arguments

d_edgePinGroupGuide	The ID of the edge pin group guide.
n_offset	The stop offset of the edge.

Value Returned

`t`	Returns `t` if the stop offset of the edge on the edge pin group guide is set.
`nil`	Returns `nil` if the stop offset of the edge on the edge pin group guide is not set.

Example

```
dbSetPinGroupGuideEdgeStopOffset(epgId 10)
=> t
```


© 2020 All Rights Reserved.

dbGetAreaBoundaryValidRoutingLayers

```
dbGetAreaBoundaryValidRoutingLayers(
     d_areaBoundaryId
     )
     => l_layers / nil
```

Description

This function returns the value of the `validRoutingLayers` constraint on the given area boundary.

Arguments

d_areaBoundaryId	The ID of the area boundary.

Value Returned

l_layers	Returns the value of the `validRoutingLayers` constraint if it is set on the specified area boundary.
`nil`	Returns `nil` if no `validRoutingLayers` constraint is set on the specified area boundary.

Example

```
dbGetAreaBoundaryValidRoutingLayers(areaBndyId)
=> ("M1" "M2" "VIA2" "VIA3" "M6")
```


© 2020 All Rights Reserved.

dbAreaBoundaryHasLayerPriorityOrder

```
dbAreaBoundaryHasLayerPriorityOrder(
     d_areaBoundaryId
     )
     => t / nil
```

Description

This function returns a boolean value indicating that the `layerPriorityOrder` constraints set on the specified area boundary.

Arguments

d_areaBoundaryId	The ID of the area boundary.

Value Returned

`t`	Returns `t` if the `layerPriorityOrder` constraint is set on the specified area boundary.
`nil`	Returned if the `layerPriorityOrder` constraint is not set on the specified area boundary.

Example

```
dbAreaBoundaryHasLayerPriorityOrder(areaBndyId)
=> t
```


© 2020 All Rights Reserved.

dbAreaBoundaryHasValidRoutingLayers

```
dbAreaBoundaryHasValidRoutingLayers(
     d_areaBoundaryId
     )
     => t / nil
```

Description

This function returns a boolean value indicating that the `validRoutingLayers` constraint is set on the specified area boundary.

Arguments

d_areaBoundaryId	The ID of the area boundary.

Value Returned

`t`	Returns `t` if the `validRoutingLayers` constraint is set on the specified area boundary.
`nil`	Returned if the `validRoutingLayers` constraint is not set on the specified area boundary.

Example

```
dbAreaBoundaryHasValidRoutingLayers(areaBndyId)
=> t
```


© 2020 All Rights Reserved.

dbGetAreaBoundaryLayerPriorityOrder

```
dbGetAreaBoundaryLayerPriorityOrder(
     d_areaBoundaryId
     )
     => t / nil
```

Description

This function returns the value of the `layerPriorityOrder` constraint that is set on the specified area boundary.

Arguments

`d_areaBoundaryId`	The ID of the area boundary.

Value Returned

`t`	Returns `t` if the `layerPriorityOrder` constraint is set on the specified area boundary.
`nil`	Returned if the `layerPriorityOrder` constraint is not set on the specified area boundary.

Example

```
dbGetAreaBoundaryLayerPriorityOrder(areaBndyId)
=> t
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideKeepoutSpacing

```
dbGetPinGroupGuideKeepoutSpacing(
     d_pinGroupGuide
     tx_layer
     )
     => n_value / nil
```

Description

This function returns the value of the `keepoutSpacing` constraint set on the specified pin group guide.

Arguments

d_pinGroupGuide	The ID fo the pin group guide name.
tx_layer	The layer name or layer number.

Value Returned

n_value	Returns the value of the `keepoutSpacing` constraint if it is set on the specified pin group guide.
`nil`	Returned if the `keepoutSpacing` constraint is not set on the specified pin group guide.

Example

```
dbGetPinGroupGuideKeepoutSpacing(epg1 0)
=> 6
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideLayerPriorityOrder

```
dbGetPinGroupGuideLayerPriorityOrder(
     d_areaPinGroupGuide
     )
     => t / nil
```

Description

This function returns the value of the `layerPriorityOrder` constraint set on the specified area pin group guide.

Arguments

`d_areaPinGroupGuide`	The ID of the area pin group guide.

Value Returned

`t`	Returns `t` if the `layerPriorityOrder` constraint is set on the specified area pin group guide.
`nil`	Returned if the `layerPriorityOrder` constraint is not set on the specified area pin group guide.

Example

```
dbGetPinGroupGuideLayerPriorityOrder(apgId) => t
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideMinPinSpacing

```
dbGetPinGroupGuideMinPinSpacing(
     d_pinGroupGuide
     tx_layer
     )
     => n_value / nil
```

Description

This function returns the value of the `minSpacing` constraint set on the specified pin group guide.

Arguments

d_pinGroupGuide	Specifies the ID of the pin group guide from which the `minSpacing` constraint value is to be retrieved.
tx_layer	The layer name or number in the pin group from which the value of the `minSpacing` constraint is to be retrieved.

Value Returned

n_value	Returns the value of the `minSpacing` constraint if it is set on the specified pin group guide.
`nil`	Returned if the `minSpacing` constraint is not set on the specified pin group guide.

Example

```
dbGetPinGroupGuideMinPinSpacing(epg 0) => 2
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideMinPinWidth

```
dbGetPinGroupGuideMinPinWidth(
     d_pinGroupGuide
     tx_layer
     )
     => n_value / nil
```

Description

This function returns the value of the `minWidth` constraint set on the specified pin group guide.

Arguments

d_pinGroupGuide	Specifies the ID of the pin group guide from which the value of the `minWidth` constraint is to be retrieved.
tx_layer	The layer name or number in the pin group from which the value of the `minWidth` constraint is to be retrieved.

Value Returned

n_value	Returns the value of the `minWidth` constraint if it is set on the specified pin group guide.
`nil`	Returned if the `minWidth` constraint is not set on the specified pin group guide.

Example

```
dbGetPinGroupGuideMinPinWidth(epg 0) => 3
```


© 2020 All Rights Reserved.

dbGetPinGroupGuideValidRoutingLayers

```
dbGetPinGroupGuideValidRoutingLayers(
     d_areaPinGroupGuide
     )
     => l_layers / nil
```

Description

This function returns the value of the `validRoutingLayers` constraint set on the specified area pin group guide.

Arguments

d_areaPinGroupGuide	The ID of the area pin group guide.

Value Returned

l_layers	Returns the value of the `validRoutingLayers` constraint if it is set on the specified area pin group guide.
`nil`	Returned if the `validRoutingLayers` constraint is not set on the specified area pin group guide.

Example

```
dbGetPinGroupGuideValidRoutingLayers(apgId) => ("M1" "M2" "VIA2" "VIA3" "M6")
```


© 2020 All Rights Reserved.

dbGetPinMinPinSpacing

```
dbGetPinMinPinSpacing(
     d_pinId
     tx_layer
     )
     => n_value / nil
```

Description

This function returns the value of the `minSpacing` constraint set on the specified pin.

Arguments

d_pinId	Specifies the ID of the pin.
tx_layer	The layer name or number in the pin group from which the value of the `minSpacing` constraint is to be retrieved.

Value Returned

n_value	Returns the value of the `minSpacing` constraint if it is set on the specified pin.
`nil`	Returned if the `minSpacing` constraint is not set on the specified pin.

Example

```
dbGetPinMinPinSpacing(pinId 0) => 6
```


© 2020 All Rights Reserved.

dbGetPinMinPinWidth

```
dbGetPinMinPinWidth(
     d_pinId
     tx_layer
     )
     => n_value / nil
```

Description

This function returns the value of the `minWidth` constraint set on the specified pin.

Arguments

d_pinId	Specifies the ID of the pin.
tx_layer	The layer name or number in the pin group from which the value of the `minWidth` constraint is to be retrieved.

Value Returned

n_value	Returns the value of the `minWidth` constraint if it is set on the specified pin.
`nil`	Returned if the `minWidth` constraint is not set on the specified pin.

Example

```
dbGetPinMinPinWidth(pinId 0) => 8
```


© 2020 All Rights Reserved.

dbPinGroupGuideHasKeepoutSpacing

```
dbPinGroupGuideHasKeepoutSpacing(
     d_pinGroupGuide
     tx_layer
     )
     => t / nil
```

Description

This function returns a boolean value indicating that the `keepoutSpacing` constraint is set on the specified pin group guide.

Arguments

d_pinGroupGuide	Specifies the ID of the pin group guide.
tx_layer	The layer name or number in the pin group from which the value of the `keepoutSpacing` constraint is to be retrieved.

Value Returned

`t`	Returns `t` if the `keepoutSpacing` constraint is set on the specified pin group guide.
`nil`	Returned if the `keepoutSpacing` constraint is not set on the specified pin group guide.

Example

```
dbPinGroupGuideHasKeepoutSpacing(epg1 0) => t
```


© 2020 All Rights Reserved.

dbPinGroupGuideHasLayerPriorityOrder

```
dbPinGroupGuideHasLayerPriorityOrder(
     d_areaPinGroupGuide
     )
     => t / nil
```

Description

This function returns a boolean value indicating that the `layerPriorityOrder` constraint is set on the specified area pin group guide.

Arguments

`d_areaPinGroupGuide`	The ID of the area pin group guide.

Value Returned

`t`	Returns `t` if the `layerPriorityOrder` constraint is set on the specified area pin group guide.
`nil`	Returned if the `layerPriorityOrder` constraint is not set on the specified area pin group guide.

Example

```
dbPinGroupGuideHasLayerPriorityOrder(apgId) => t
```


© 2020 All Rights Reserved.

dbPinGroupGuideHasMinPinSpacing

```
dbPinGroupGuideHasMinPinSpacing(
     d_PinGroupGuide
     tx_layer
     )
     => t / nil
```

Description

This function returns a boolean value indicating whether the `minSpacing` constraint is set on the specified pin group guide.

Arguments

d_PinGroupGuide	The ID of the pin group guide.
tx_layer	The layer name or number in the pin group from which the value of the `minSpacing` constraint is to be retrieved.

Value Returned

`t`	Returns `t` if the `minSpacing` constraint is set on the specified pin group guide.
`nil`	Returned if the `minSpacing` constraint is not set on the specified pin group guide.

Example

```
dbPinGroupGuideHasMinPinSpacing(epg1 0) => t
```


© 2020 All Rights Reserved.

dbPinGroupGuideHasMinPinWidth

```
dbPinGroupGuideHasMinPinWidth(
     d_PinGroupGuide
     tx_layer
     )
     => t / nil
```

Description

This function returns a boolean value indicating whether the `minWidth` constraint is set on the specified pin group guide.

Arguments

d_PinGroupGuide	The ID of the pin group guide.
tx_layer	The layer name or number in the pin group from which the value of the `minWidth` constraint is to be retrieved.

Value Returned

`t`	Returns `t` if the `minWidth` constraint is set on the specified pin group guide.
`nil`	Returned if the `minWidth` constraint is not set on the specified pin group guide.

Example

```
dbPinGroupGuideHasMinPinWidth(epg1 0) => t
```


© 2020 All Rights Reserved.

dbPinGroupGuideHasValidRoutingLayers

```
dbPinGroupGuideHasValidRoutingLayers(
     d_areaPinGroupGuide
     )
     => t / nil
```

Description

This function returns a boolean value indicating whether the `validRoutingLayers` constraint is set on the specified area pin group guide.

Arguments

`d_areaPinGroupGuide`	The ID of the area pin group guide.

Value Returned

`t`	Returns `t` if the `validRoutingLayers` constraint is set on the specified area pin group guide.
`nil`	Returned if the `validRoutingLayers` constraint is not set on the specified area pin group guide.

Example

```
dbPinGroupGuideHasValidRoutingLayers(apgId) => t
```


© 2020 All Rights Reserved.

dbPinHasMinPinSpacing

```
dbPinHasMinPinSpacing(
     d_PinId
     tx_layer
     )
     => t / nil
```

Description

This function returns a boolean value indicating whether the `minSpacing` constraint is set on the specified pin.

Arguments

d_PinId	The ID of the pin.
tx_layer	The layer name or number in the pin group from which the value of the `minSpacing` constraint is to be retrieved.

Value Returned

`t`	Returns `t` if the `minSpacing` constraint is set on the specified pin.
`nil`	Returned if the `minSpacing` constraint is not set on the specified pin.

Example

```
dbPinHasMinPinSpacing(pinId 0) => t
```


© 2020 All Rights Reserved.

dbPinHasMinPinWidth

```
dbPinHasMinPinWidth(
     d_pinId
     tx_layer
     )
     => t / nil
```

Description

This function returns a boolean value indicating whether the minWidth constraint has been applied to the specified pin.

Arguments

d_pinId	The database ID of the pin.
tx_layer	The layer name or the layer number of the specified pin.

Value Returned

`t`	Returned when the `minWidth` contraint has been applied to the specified pin.
`nil`	Returned if the `minWidth` constraint has not been applied to the specified pin or if the pin ID is invalid.

Example

```
dbPinHasMinPinWidth(pinId 0) => t
```


© 2020 All Rights Reserved.

dbSetAreaBoundaryLayerPriorityOrder

```
dbSetAreaBoundaryLayerPriorityOrder(
     d_areaBoundaryId
     g_value
     )
     => t / nil
```

Description

This function sets the value of the `layerPriorityOrder` constraint on the given area boundary.

Arguments

d_areaBoundaryId	The database ID of the area boundary.
g_value	The value of the `layerPriorityOrder` constraint that you want to set.

Value Returned

`t`	Returned if the value of the `layerPriorityOrder` constraint is successfully set.
`nil`	Returned if the value of the `layerPriorityOrder` constraint cannot be set successfully.

Example

```
dbSetAreaBoundaryLayerPriorityOrder(areaBndyId t) => t
```


© 2020 All Rights Reserved.

dbSetAreaBoundaryValidRoutingLayers

```
dbSetAreaBoundaryValidRoutingLayers(
     d_areaBoundaryId
     l_layers
     )
     => t / nil
```

Description

This function sets the value of the validRoutingLayers constraint on the given area boundary.

Arguments

d_areaBoundaryId	The ID of the area boundary.
l_layers	The list of valid routing layers.

Value Returned

`t`	Returns `t` if the value of the validRoutingLayers constraint is successfully set.
`nil`	Returns `nil` if the value of the validRoutingLayers constraint cannot be set successfully.

Example

```
dbSetAreaBoundaryValidRoutingLayers(areaBndyId list("M1" "M2" "VIA2" "VIA3" "M6"))
=> t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideKeepoutSpacing

```
dbSetPinGroupGuideKeepoutSpacing(
     d_pinGroupGuide
     tx_layer
     n_value
     )
     => t / nil
```

Description

This function sets the value of the keepoutSpacing constraint on the given pin group guide.

Arguments

d_pinGroupGuide	The ID of the pin group guide.
tx_layer	The layer number or layer name.
n_value	The value that you want to set for the constraint.

Value Returned

`t`	Returns `t` if the value of the keepoutSpacing constraint is successfully set.
`nil`	Returns `nil` if the value of the keepoutSpacing constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideKeepoutSpacing(epg1 0 1)
=> t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideLayerPriorityOrder

```
dbSetPinGroupGuideLayerPriorityOrder(
     d_areaPinGroupGuide
     g_value
     )
     => t / nil
```

Description

This function sets the value of the `layerPriorityOrder` constraint on the given area pin group guide.

Arguments

`d_areaPinGroupGuide`	The ID of the area pin group guide.
`g_value`	The value that you want to set for the constraint.

Value Returned

`t`	Returns `t` if the value of the `layerPriorityOrder` constraint is successfully set.
`nil`	Returns `nil` if the value of the `layerPriorityOrder` constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideLayerPriorityOrder(apgId t)
=> t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideMinPinSpacing

```
dbSetPinGroupGuideMinPinSpacing(
     d_pinGroupGuide
     tx_layer
     n_value
     )
     => t / nil
```

Description

This function sets the value of the `minSpacing` constraint on the given pin group guide.

Arguments

d_PinGroupGuide	The ID of the pin group guide.
tx_layer	The layer name or layer number.
n_value	The integer value that you want to set for the minSpacing constraint.

Value Returned

`t`	Returns `t` if the value of the `minSpacing` constraint is successfully set.
`nil`	Returns `nil` if the value of the `minSpacing` constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideMinPinSpacing(epg 0 2)
=> t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideMinPinWidth

```
dbSetPinGroupGuideMinPinSpacing(
     d_pinGroupGuide
     tx_layer
     n_value
     )
     => t / nil
```

Description

This function sets the value of the `minWidth` constraint on the given pin group guide.

Arguments

d_PinGroupGuide	The ID of the pin group guide.
tx_layer	The layer name or layer number.
n_value	The integer value that you want to set for the minWidth constraint.

Value Returned

`t`	Returns `t` if the value of the `minWidth` constraint is successfully set.
`nil`	Returns `nil` if the value of the `minWidth` constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideMinPinWidth(epg 0 3)
=> t
```


© 2020 All Rights Reserved.

dbSetPinGroupGuideValidRoutingLayers

```
dbSetPinGroupGuideValidRoutingLayers(
     d_areapinGroupGuide
     l_layers
     )
     => t / nil
```

Description

This function sets the value of the `validRoutingLayers` constraint on the given area pin group guide.

Arguments

d_areaPinGroupGuide	The ID of the area pin group guide.
l_layers	A list of valid routing layers.

Value Returned

`t`	Returns `t` if the value of the `validRoutingLayers` constraint is successfully set.
`nil`	Returns `nil` if the value of the `validRoutingLayers` constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideValidRoutingLayers(apgId list("M1" 2 3 "VIA3" "M6"))
=> t
```


© 2020 All Rights Reserved.

dbSetPinMinPinSpacing

```
dbSetPinMinPinSpacing(
     d_pinID
     tx_layer
     n_value
     )
     => t / nil
```

Description

This function sets the value of the `minSpacing` constraint on the given pin.

Arguments

d_pinID	The ID of the pin.
tx_layer	The layer name or the layer number.
n_value	The integer value of the constraint that you want to set.

Value Returned

`t`	Returns `t` if the value of the `minSpacing` constraint is successfully set.
`nil`	Returns `nil` if the value of the `minSpacing` constraint cannot be set successfully.

Example

```
dbSetPinMinPinSpacing(pinId 0 5)
=> t
```


© 2020 All Rights Reserved.

dbSetPinMinPinWidth

```
dbSetPinMinPinWidth(
     d_pinID
     tx_layer
     n_value
     )
     => t / nil
```

Description

This function sets the value of the `minWidth` constraint on the given pin.

Arguments

d_pinID	The ID of the pin.
tx_layer	The layer name or the layer number.
n_value	The integer value of the constraint that you want to set.

Value Returned

`t`	Returns `t` if the value of the `minWidth` constraint is successfully set.
`nil`	Returns `nil` if the value of the `minWidth` constraint cannot be set successfully.

Example

```
dbSetPinMinPinWidth(pinId 0 4)
=> t
```


© 2020 All Rights Reserved.

dbUnsetAreaBoundaryLayerPriorityOrder

```
dbUnsetAreaBoundaryLayerPriorityOrder(
     d_areaBoundaryId
     )
     => t / nil
```

Description

This function deletes the `layerPriorityOrder` constraint on the given area boundary.

Arguments

`d_areaBoundaryId`	The ID of the area boundary.

Value Returned

`t`	Returns `t` if the `layerPriorityOrder` constraint is successfully deleted.
`nil`	Returns `nil` if the `layerPriorityOrder` constraint is not deleted successfully.

Example

```
dbUnsetAreaBoundaryLayerPriorityOrder(areaBndyId)
=> t
```


© 2020 All Rights Reserved.

dbUnsetAreaBoundaryValidRoutingLayers

```
dbUnsetAreaBoundaryValidRoutingLayers(
     d_areaBoundaryId
     )
     => t / nil
```

Description

This function deletes the `validRoutingLayers` constraint on the given area boundary.

Arguments

d_areaBoundaryId	The ID of the area boundary.

Value Returned

`t`	Returns `t` if the `validRoutingLayers` constraint is deleted successfully.
`nil`	Returns `nil` if the `validRoutingLayers` constraint cannot be deleted successfully.

Example

```
dbUnsetAreaBoundaryValidRoutingLayers(areaBndyId)
=> t
```


© 2020 All Rights Reserved.

dbUnsetPinGroupGuideKeepoutSpacing

```
dbUnsetPinGroupGuideKeepoutSpacing(
     d_pinGroupGuide
     tx_layer
     )
     => t / nil
```

Description

This function deletes the `keepoutSpacing` constraint on the given pin group guide.

Arguments

d_pinGroupGuide	The ID of the pin group guide.
tx_layer	The layer number or layer name.

Value Returned

`t`	Returns `t` if `keepoutSpacing` constraint is successfully deleted.
`nil`	Returns `nil` if the `keepoutSpacing` constraint cannot be deleted.

Example

```
dbSetPinGroupGuideKeepoutSpacing(epg1 0 1)
=> t
```


© 2020 All Rights Reserved.

dbUnsetPinGroupGuideLayerPriorityOrder

```
dbUnsetPinGroupGuideLayerPriorityOrder(
     d_areaPinGroupGuide
     )
     => t / nil
```

Description

This function deletes `layerPriorityOrder` constraint on the given area pin group guide.

Arguments

`d_areaPinGroupGuide`	The ID of the area pin group guide.

Value Returned

`t`	Returns `t` if the `layerPriorityOrder` constraint is successfully deleted.
`nil`	Returns `nil` if the `layerPriorityOrder` constraint cannot be deleted.

Example

```
dbUnsetPinGroupGuideLayerPriorityOrder(apgId)
=> t
```


© 2020 All Rights Reserved.

dbUnsetPinGroupGuideMinPinSpacing

```
dbUnsetPinGroupGuideMinPinSpacing(
     d_pinGroupGuide
     tx_layer
     )
     => t / nil
```

Description

This function deletes the `minSpacing` constraint on the given pin group guide.

Arguments

d_PinGroupGuide	The ID of the pin group guide.
tx_layer	The layer name or layer number.

Value Returned

`t`	Returns `t` if the `minSpacing` constraint is successfully deleted.
`nil`	Returns `nil` if the `minSpacing` constraint cannot be deleted successfully.

Example

```
dbUnsetPinGroupGuideMinPinSpacing(epg 0)
=> t
```


© 2020 All Rights Reserved.

dbUnsetPinGroupGuideMinPinWidth

```
dbUnsetPinGroupGuideMinPinSpacing(
     d_pinGroupGuide
     tx_layer
     )
     => t / nil
```

Description

This function deletes the `minWidth` constraint on the given pin group guide.

Arguments

d_PinGroupGuide	The ID of the pin group guide.
tx_layer	The layer name or layer number.

Value Returned

`t`	Returns `t` if the `minWidth` constraint is successfully deleted.
`nil`	Returns `nil` if the `minWidth` constraint cannot be deleted successfully.

Example

```
dbUnsetPinGroupGuideMinPinWidth(epg 0)
=> t
```


© 2020 All Rights Reserved.

dbUnsetPinGroupGuideValidRoutingLayers

```
dbUnsetPinGroupGuideValidRoutingLayers(
     d_areapinGroupGuide
     )
     => t / nil
```

Description

This function deletes the `validRoutingLayers` constraint on the given area pin group guide.

Arguments

`d_areaPinGroupGuide`	The ID of the area pin group guide.

Value Returned

`t`	Returns `t` if the `validRoutingLayers` constraint is successfully deleted.
`nil`	Returns `nil` if the `validRoutingLayers` constraint cannot be deleted successfully.

Example

```
dbUnsetPinGroupGuideValidRoutingLayers(apgId)
=> t
```


© 2020 All Rights Reserved.

dbUnsetPinMinPinSpacing

```
dbUnsetPinMinPinSpacing(
     d_pinID
     tx_layer
     )
     => t / nil
```

Description

This function deletes the `minSpacing` constraint on the given pin.

Arguments

d_pinID	The ID of the pin.
tx_layer	The layer name or the layer number.

Value Returned

`t`	Returns `t` if the `minSpacing` constraint is successfully deleted.
`nil`	Returns `nil` if the `minSpacing` constraint cannot be deleted successfully.

Example

```
dbUnsetPinMinPinSpacing(pinId 0)
=> t
```


© 2020 All Rights Reserved.

dbUnsetPinMinPinWidth

```
dbUnsetPinMinPinWidth(
     d_pinID
     tx_layer
     )
     => t / nil
```

Description

This function deletes the `minWidth` constraint on the given pin.

Arguments

d_pinID	The ID of the pin.
tx_layer	The layer name or the layer number.

Value Returned

`t`	Returns `t` if the `minWidth` constraint is successfully deleted.
`nil`	Returns `nil` if the `minWidth` constraint cannot be deleted successfully.

Example

```
dbUnsetPinMinPinWidth(pinId 0)
=> t
```


© 2020 All Rights Reserved.

7

Placement Database Access Functions

This chapter describes the following placement database SKILL functions:

- Background Definition Functions
  - dbCreateBackgroundDef
  - dbCreateBackgroundDefByAttr
  - dbGetBackgroundDefAttr
  - dbGetCellViewBackgroundDef
- Component Type Set Definition Functions
  - dbCreateCompTypeSetDef
  - dbCreateCompTypeSetDefByAttr
  - dbGetCellViewCompTypeSetDef
  - dbGetCompTypeSetDefAttr
  - dbGetCompTypeSetDefCompFilters
  - dbSetCompTypeSetDefCompFilters
  - dbUnsetCompTypeSetDefCompFilters
- Rail Definition Functions
  - dbCreateRailDef
  - dbCreateRailDefByAttr
  - dbGetCellViewRailDef
  - dbGetRailDefAttr
  - dbGetRailDefFigType
  - dbSetRailDefFigType

© 2020 All Rights Reserved.

- Placement Area Definition Functions
  - dbAddPlaceAreaBackgroundDef
  - dbAddPlaceAreaCompTypeSetDef
  - dbAddPlaceAreaRailDef
  - dbCreatePlaceArea
  - dbGetCellViewPlaceArea
  - dbGetPlaceAreaBackgroundDef
  - dbGetPlaceAreaCompTypeSetDef
  - dbGetPlaceAreaOrient
  - dbGetPlaceAreaRailDef
  - dbGetPlaceAreaUtilization
  - dbGetPlacementGrid
  - dbRemovePlaceAreaBackgroundDef
  - dbRemovePlaceAreaCompTypeSetDef
  - dbRemovePlaceAreaRailDef
  - dbSetPlaceAreaOrient
  - dbSetPlaceAreaUtilization
  - dbSetPlacementGrid
- Row-based Functions
  - dbAddRowBackgroundDef
  - dbAddRowCompTypeSetDef
  - dbGetRowBackgroundDef
  - dbGetRowCompTypeSetDef
  - dbAddRowRailDef
  - dbGetRowRailDef
  - dbRemoveRowCompTypeSetDef
  - dbRemoveRowBackgroundDef

© 2020 All Rights Reserved.

- dbRemoveRowRailDef

- Row Region Specification Functions (ICADVM20.1 Only)
  - dbCreateRowRegionSpec
  - dbFindRowRegionSpec
  - dbGetCellViewRowRegionSpecs
- Row Region Functions (ICADVM20.1 Only)
  - dbAttachRowRegionToPRBoundary
  - dbCreateRowRegion
  - dbDetachPRBoundaryFromRowRegion
  - dbFindRowRegion
  - dbFlattenRowRegion
  - dbGetCellViewRowRegions
  - dbGetRowRegionAttachedToPRBoundary
  - dbGetRowRegionPlaceRows
  - dbGetRowRegionPoints
  - dbGetRowRegionRowRegionSpec
  - dbGetRowRegionSpecRowRegions
  - dbGetRowRegionUsesPartialRowSpec
  - dbIsRowRegion
  - dbIsRowRegionAttachedToPRBoundary
  - dbRebuildRowRegion
  - dbRowRegionMatchesSpec
  - dbSetRowRegionPoints
  - dbSetRowRegionRowRegionSpec
  - dbSetRowRegionUsesPartialRowSpec
- Miscellaneous Functions (ICADVM20.1 Only)
  - dbGetPlaceRowVersion

© 2020 All Rights Reserved.

- dbGetRowRegionVersion
- dbIsImplicit
- dbIsInRowRegion

© 2020 All Rights Reserved.

Background Definition Functions

dbCreateBackgroundDef

```
dbCreateBackgroundDef(
     d_cvId
     t_name
     l_lpp
     t_netName
     g_leftEnc
     g_rightEnc
     g_topEnc
     g_bottomEnc
     )
     => d_backgroundDefId / nil
```

Description

Creates a backgroundDef based on the specified attributes. The layer-purpose pair must exist to create a backgroundDef.

Arguments

d_cvId	Database ID of a cellview.
t_name	Name of a backgroundDef.
l_lpp	Layer-purpose pair.
t_netName	Name of a net.
g_leftEnc	Left enclosure of the background shape.
g_rightEnc	Right enclosure of the background shape.
g_topEnc	Top enclosure of the background shape.
g_bottomEnc	Bottom enclosure of the background shape.

Value Returned

d_bgDefId	Database ID of a backgroundDef.
nil	Returned in case of failure.

© 2020 All Rights Reserved.

Example

```
bgDef1 = dbCreateBackgroundDef(cv "bgDef1" list("FinFet" "region1") "bgNet" 10 20
30 40)
```

A background definition `bgDef1` is created in the cellview `cv` with the following attributes:

- Layer-purpose pair: `FinFet region1`
- Net: `bgNet`
- Left enclosure: `10`
- Right enclosure: `20`
- Top enclosure: `30`
- Bottom enclosure: `40`

© 2020 All Rights Reserved.

dbCreateBackgroundDefByAttr

```
dbCreateBackgroundDefByAttr(
     d_cvId
     t_name
     g_dpl
     )
     => d_bgDefId / nil
```

Description

Creates a backgroundDef based on the SKILL DPL specification of the object. Contents of the SKILL DPL specification are checked before the backgroundDef is created.

Arguments

d_cvId	Database ID of a cellview.
t_name	Name of a backgroundDef.
g_dpl	SKILL DPL specification of the backgroundDef. DPL is a list of following required and optional attributes:

- Required Attributes

```
bckAttr = list(nil)
bckAttr->lpp = '("Metal1" "drawing")
```

- Optional Attributes

```
bckAttr2->netName  = "VSS!"
bckAttr2->leftEnc  = 0.01
bckAttr2->rightEnc = 0.01
bckAttr2->topEnc   = 0.01
bckAttr2->botEnc   = 0.01
```

Value Returned

d_bgDefId	Database ID of the backgroundDef that is created.
nil	Returned in case of failure.

Example

```
; creating the SKILL DPL
```


© 2020 All Rights Reserved.

```
        bgDefAttr = list(nil)
        bgDefAttr->lpp = list("bckLayer" "drawing")
        bgDefAttr->netName = "bgNet"
        bgDef2Attr->leftEnc = 20
        bgDef2Attr->rightEnc = 30
        bgDef2Attr->topEnc = 40
        bgDef2Attr->botEnc = 50
;; creating bgDef based on the SKILL DPL
        bgDef = dbCreateBackgroundDefByAttr(cv "bgDef" bgDefAttr)
```


© 2020 All Rights Reserved.

dbGetBackgroundDefAttr

```
dbGetBackgroundDefAttr(
     d_bgDefId
     )
     => l_attr / nil
```

Description

Returns the attributes of the specified backgroundDef.

Arguments

d_bgDefId	Database ID of the backgroundDef.

Value Returned

l_attr	List of backgroundDef attributes.
nil	Returned in case of failure.

Example

```
;; Create backgroundDef
       bgDef = dbCreateBackgroundDef(cv "bgDef1" list("FinFet" "region1") "bgNet"
10 20 30 40)

         ;; Get BackgroundDef Attributes
            dbGetBackgroundDefAttr(bgDef)
            (nil lpp ("FinFet" "region1") netName "bgNet"
             leftEnc 10.0 rightEnc 20.0 topEnc
             30.0 botEnc 40.0)
```


© 2020 All Rights Reserved.

dbGetCellViewBackgroundDef

```
dbGetCellViewBackgroundDef(
     d_cellviewId
     )
     => l_bgDefId / nil
```

Description

Returns list of backgroundDefs in the given cellview.

Arguments

d_cellviewId	Database ID of a cellview.

Value Returned

l_bgDefId	List of backgroundDefs in the given cellview.
nil	Returned in case of failure.

Example

```
dbGetcellviewBackgroundDef(cv)
```

Returns the list of background definitions in the cellview `cv`.

© 2020 All Rights Reserved.

Component Type Set Definition Functions

dbCreateCompTypeSetDef

```
dbCreateCompTypeSetDef(
     d_cellViewId
     t_compTypeSetName
     t_rowReference
     t_componentReference
     l_allowedOrientations
     l_componentTypes
     [ g_componentOffset ]
     [ g_pitch ]
     )
     => d_compTypeSetDef / nil
```

Description

This function creates a container of component types that use the same placement attributes.

Arguments

d_cellViewId	Database ID of a cellview.
t_compTypeSetName	The name of the compTypeSetDef.
t_rowReference	Specifies row reference. Valid Values: `bottomAlignment`, `centerAlignment`, `topAlignment`.
t_componentReference	Specifies component reference. Valid Values: `bottomAlignment`, `centerAlignment`, `topAlignment`.
l_allowedOrientations	Allowed orientations that placer can use for cells in the component types.
l_componentTypes	List of component types that share placement attributes.
g_componentOffset	Optionally the offset from the reference points computed from row reference can be specified in user units.
g_pitch	Optionally the pitch or grid to use between placement locations can be specified.

© 2020 All Rights Reserved.

Value Returned

d_compTypeSetDef	Returns Database ID of a compTypeSetDef.
nil	An error occurred during execution.

Example

```
;; create compTypeSetDef
def = dbCreateCompTypeSetDef(cellViewId
     "myCompTypeSetDef" ; name of the component type set
     "bottomAlignment" ; row reference
     "bottomAlignment" ; component reference
     list("R0" "MX") ; allowed orientations
     list("Comptype1" "CompType2")  ; component types
     offsetFromRowReference ; offset from reference points
     componnentPitch) ; pitch or grid to use between placement locations
```


© 2020 All Rights Reserved.

dbCreateCompTypeSetDefByAttr

```
dbCreateCompTypeSetDefByAttr(
     d_cvId
     t_name
     l_attr_dpl
     )
     => d_compTypeSetDefId/nil
```

Description

Creates a compTypeSetDef in the given cellview based on the specified attributes list.

Arguments

d_cvId	Database ID of a cellview.
t_name	Name of a compTypeSetDef.
l_attr_dpl	SKILL DPL for the compType attributes. DPL is a list of following required and optional attributes:

- Required attributes

```
compAttr = list(nil)
compAttr->refOnArea    = "centerAlignment"
compAttr->refOnComp    = "centerAlignment"
compAttr->orients      = list("R0" "R90")
```

- At least one of the following attributes:

```
compAttr->allowedTypes = list("compType1"
"compType2")
compAttr->compFilters  = list(list("cdn20FF" "n*"
"*"))
```

- Optional Attributes

```
compAttr->offset
compAttr->grid
```


© 2020 All Rights Reserved.

Value Returned

d_compTypeSetDef	Database ID of the compTypeSetDef that is created.
nil	Returned in case of failure.

Example

```
;Create compTypeSetDef
         compAttr = list(nil)
         compAttr->refOnArea  = "centerAlignment"
         compAttr->refOnComp  = "anyAlignment"
         compAttr->allowedTypes = list("compType1" "compType2")
         compAttr->orients = list("R0")
        compAttr->compFilters = list(list("cellLib" "nf*" "layout") list("cellLib"
"ng*" "layout"))

         cdef = dbCreateCompTypeSetDefByAttr(cv "compTypeSetDef" compAttr)
```


© 2020 All Rights Reserved.

dbGetCellViewCompTypeSetDef

```
dbGetCellViewCompTypeSetDef(
     d_cellViewId
     )
     => d_compTypeSetDef / nil
```

Description

This function returns a list of compTypeSetDef objects defined in the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

d_compTypeSetDef	Returns a list of compTypeSetDef objects.
nil	If a compTypeSetDef objects does not exist, nil is returned.

Example

```
cellViewId = geGetEditCellView()

compTypeSetDefs = dbGetCellViewCompTypeSetDef(cellViewId)
```


© 2020 All Rights Reserved.

dbGetCompTypeSetDefAttr

```
dbGetCompTypeSetDefAttr(
     d_compDefId
     )
     => l_attr / nil
```

Description

Returns the attributes of the specified dbCompTypeSetDef.

Arguments

d_compDefId	Database ID of compTypeSetDef.

Value Returned

l_attr	Lists the attributes of the specified compTypeSetDef.
nil	Returned in case of failure.

Example

```
;; Create compTypeSetDef.
         compAttr = list(nil)
         compAttr->refOnArea  = "centerAlignment"
         compAttr->refOnComp  = "anyAlignment"
         compAttr->allowedTypes = list("compType1" "compType2")
         compAttr->orients = list("R0")
        compAttr->compFilters = list(list("cellLib" "nf*" "layout") list("cellLib"
"ng*" "layout"))
         cdef = dbCreateCompTypeSetDefByAttr(cv "compTypeSetDef" compAttr)

         ;; Query attribute list form the given compTypeSetDef
         dbGetCompTypeSetDefAttr(cdef)
```


© 2020 All Rights Reserved.

dbGetCompTypeSetDefCompFilters

```
dbGetCompTypeSetDefCompFilters(
     d_compDefId
     )
     => l_compFilters / nil
```

Description

Returns the component filters in the specified compTypeSetDef.

Arguments

d_compDefID	Database ID of a compTypeSetDef.

Value Returned

l_compFilters	List of component filters.
nil	Returns an error in case of failure.

Example

```
; Create compTypeSetDef
         compAttr = list(nil)
         compAttr->refOnArea  = "centerAlignment"
         compAttr->refOnComp  = "anyAlignment"
         compAttr->allowedTypes = list("compType1" "compType2")
         compAttr->orients = list("R0")
        compAttr->compFilters = list(list("cellLib" "nf*" "layout") list("cellLib"
"ng*" "layout"))

         cdef = dbCreateCompTypeSetDefByAttr(cv "compTypeSetDef" compAttr);

         ;; Get compFilters
         dbGetCompTypeSetDefCompFilters(cdef)
```


© 2020 All Rights Reserved.

dbSetCompTypeSetDefCompFilters

```
dbSetCompTypeSetDefCompFilters(
     d_compDefId
     l_compFilters
     )
     => t/nil
```

Description

Sets the component filters in the specified compTypeSetDef.

Arguments

d_compDefId	Database ID of compTypeSetDef.
l_compFilters	List of component filters.

Value Returned

`t`	Updated component filters on compTypeSetDef.
`nil`	Returned in case of failure.

Example

```
;Create compTypeSetDef
         cdef = dbCreateCompTypeSetDef(cv "compTypeSetDef" "anyAlignment"
"anyAlignment" list("R0" "MX") list("CT1" "CT2") 0.2 0.3)
; Set compFilters
dbSetCompTypeSetDefCompFilters(cdef  list(list("cellLib" "nf*" "layout")
list("cellLib" "ngxx*" "layout")))
```


© 2020 All Rights Reserved.

dbUnsetCompTypeSetDefCompFilters

```
dbUnsetCompTypeSetDefCompFilters(
     d_compDefId
     )
     => t / nil
```

Description

Removes the component filters from the specified compTypeSetDef.

Arguments

d_compDefId	Database ID of a compTypeSetDef.

Value Returned

t	Removed component filters from the specified compTypeSetDef.
nil	Returned in case of failure.

Example

```
; Create compTypeSetDef with compFilters
         compAttr = list(nil)
         compAttr->refOnArea  = "centerAlignment"
         compAttr->refOnComp  = "anyAlignment"
         compAttr->allowedTypes = list("compType1" "compType2")
         compAttr->orients = list("R0")
       compAttr->compFilters = list(list("cellLib" "nf*" "layout") list("cellLib"
"ng*" "layout"))
         cdef = dbCreateCompTypeSetDefByAttr(cv "compTypeSetDef" compAttr)
;; Set the compFilters.
dbSetCompTypeSetDefCompFilters(cdef list(list("cellLib" "nf*" "layout")
list("cellLib" "ngxx*" "layout")))

         ;; Unset the compFilters.
         dbUnsetCompTypeSetDefCompFilters(cdef)
```


© 2020 All Rights Reserved.

Rail Definition Functions

dbCreateRailDef

```
dbCreateRailDef(
     d_cellViewId
     t_raildDefName
     t_netName
     l_lpp
     t_rowReference
     t_railReference
     g_width
     [ g_offset ]
     [ g_beginOffset ]
     [ g_endOffset ]
     )
     => d_railDef / nil
```

Description

This function creates a container of attributes to define a row rail. The definition can be referenced in more than one row or placeArea. If a physical attribute is modifed in the object, it is instantly reflected in all row/placeArea that reference the railDef.

Arguments

d_cellViewId	Database ID of a cellview.
t_railDefName	Name of the railDef.
t_netName	Name of the net.
l_lpp	Layer and purpose of the rail geometry.
t_rowReference	Specifies row reference. Valid Values: `bottomAlignment`, `centerAlignment`, `topAlignment`.
t_railReference	Specifies rail reference. Valid Values: `bottomAlignment`, `centerAlignment`, `topAlignment`.
g_width	Width of rail geometry.

© 2020 All Rights Reserved.

g_offset	Optional. If this value is specified, then the rail is placed at the specified distance (in user units) from the two reference points (rail and row). A negative value indicates offsetting the rail below the top/bottom edge of the row and a positive value indicates offsetting the rail above the top/bottom edge.
g_beginOffset	Optional. This value offsets or extends the rail by the specified user units from the left edge of the row. A negative value indicates offsetting the rail towards the left side of the left edge of the row and a positive value indicates offsetting the rail towards the right side of the left edge of the row.
g_endOffset	Optional. This value offsets or extends the rail by the specified user units from the right edge of the row. A negative value indicates offsetting the rail towards the left side of the right edge of the row and a positive value indicates offsetting the rail towards the right side of the right edge of the row.

Value Returned

d_railDef	Returns the database ID of the railDef.
nil	An error occurred during execution.

Example 1

```
;; create railDef
def = dbCreateRailDef(cellViewId
     "myRailDef"
      "netName"
      list("layer" "purpose")
      "bottomAlignment"
      "bottomAlignment"
      railWidth
     offsetFromRowReference)
```

Example 2

The following example shows two rails aligned to a row. The offset value for the first rail is 0. So, in the figure shown below the top of the first rail is aligned to the top edge of the row. The

© 2020 All Rights Reserved.

offset value for the second rail is also 0. Therefore, the bottom of the second rail is aligned to the bottom edge of the row.

Setting the offset value for the first rail to 0.4 and the value for the second rail to 0.3 will place the two rails in the row as shown in the figure below:

Example 3

The following table specifies, in addition to the offset values, the begin and end offset values for the two rails.

Rail	Offset	Begin offset	End offset
Top rail	0.4	-0.8	+1.2
Bottom rail	0.3	+1.5	-1.8

© 2020 All Rights Reserved.

Note: A negative begin/end offset value sets the rail relative to the left side of the row and positive begin/end offset value stes the rail relative to the right side of the row. This is shown in the figure below:

© 2020
All Rights Reserved.

dbCreateRailDefByAttr

```
dbCreateRailDefByAttr(
     d_cvId
     t_name
     l_attr_dpl
     )
     => d_railDefId / nil
```

Description

Creates railDef based on the SKILL DPL of the railDef object specification. Contents of the SKILL DPL specification are checked before railDef is created.

Arguments

d_cvId	Database ID of a cellview.
t_name	Name of a railDef.
l_attr_dpl	SKILL DPL of the railDef specification. DPL is a list of following required and optional attributes:

- Required

```
pwrRail = list(nil)
pwrRail->lpp       = '("Metal1" "drawing")
pwrRail->netName   = "VDD!"
pwrRail->width     = 0.064
pwrRail->refOnArea = "bottomAlignment"
pwrRail->refOnRail = "bottomAlignment"
```

- Optional

```
pwrRail->railFigType = "pathSeg"
pwrRail->offset
pwrRail->beginOffset
pwrRail->endOffset
```


© 2020 All Rights Reserved.

Value Returned

d_railDeff	Database ID of railDef that is created.
nil	Returned in case of failure.

Example

```
;Creating SKILL DPL for railDef specification
         gndRail = list(nil)
         gndRail->lpp = list("gndLayer" "drawing")
         gndRail->width = 0.02
         gndRail->refOnArea = "topAlignment"
         gndRail->refOnRail = "centerAlignment"
         gndRail->netName = "gndNet"
         gndRail->railFigType = "pathSeg"
;; Creating railDef
         rdef = dbCreateRailDefByAttr(cv "railDef" gndRail)
```


© 2020 All Rights Reserved.

dbGetCellViewRailDef

```
dbGetCellViewRailDef(
     d_cellViewId
     )
     => l_railDefObjects / nil
```

Description

Returns a list of railDef objects defined in the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

l_railDefObjects	Returns a list of railDef objects.
nil	If no railDef objects exist, `nil` is returned.

Example

```
cellViewId = geGetEditCellView()

defs = dbGetCellViewRailDef(cellViewId)
```


© 2020 All Rights Reserved.

dbGetRailDefAttr

```
dbGetRailDefAttr(
     d_railDefId
     )
     => l_attr / nil
```

Description

Returns the rail attributes of the specified railDef.

Arguments

d_railDefId	Database ID of the railDef database.

Value Returned

l_railAttrs	List of rail attributes of the specified railDef.
nil	Returned in case of failure.

Example

```
;; Create Rail attrs
         gndRail = list(nil)
         gndRail->lpp = list("gndLayer" "drawing")
         gndRail->width = 0.02
         gndRail->refOnArea = "topAlignment"
         gndRail->refOnRail = "centerAlignment"
         gndRail->netName = "gndNet"
         gndRail->railFigType = "pathSeg"
;; Create RailDef and query rail attributes
         rdef = dbCreateRailDefByAttr(cv "railDef" gndRail)
         dbGetRailDefAttr(rdef)
```


© 2020 All Rights Reserved.

dbGetRailDefFigType

```
dbGetRailDefFigType(
     d_railDefId
     )
     => t_figType / nil
```

Description

Returns the figType specified in the given railDef.

Arguments

d_railDefId	Database ID of railDef.

Value Returned

t_figType	Returns figType of the specified railDef. Valid values are `rect` and `pathSeg`.
`nil`	Returned in case of failure.

Example

```
; Create Rail attrs
         gndRail = list(nil)
         gndRail->lpp = list("gndLayer" "drawing")
         gndRail->width = 0.02
         gndRail->refOnArea = "topAlignment"
         gndRail->refOnRail = "centerAlignment"
         gndRail->netName = "gndNet"
         gndRail->railFigType = "pathSeg"

         ;; Create RailDef and query railDef figType
         rdef = dbCreateRailDefByAttr(cv "railDef" gndRail)

         dbGetRailDefFigType(rdef)
```


© 2020 All Rights Reserved.

dbSetRailDefFigType

```
dbSetRailDefFigType(
     d_railDefId
     t_railFigType
     )
     => t / nil
```

Description

Sets the railFigType for the specified railDef.

Arguments

d_railDef	Database ID of a railDef.
t_railFigType	Value of figType. Valid values are `default`, `rect` and `pathSeg`. Default value is `rect`.

Value Returned

`t`	Specifed figType is set on the given railDef.
`nil`	Returned in case of failure.

Example

```
dbSetRailDefFigType(d_rowRegionId "pathSeg")
```

Sets figType `pathseg` on rail definition `d_rowRegionId`.

© 2020 All Rights Reserved.

Placement Area Definition Functions

dbAddPlaceAreaBackgroundDef

```
dbAddPlaceAreaBackgroundDef(
     d_plcAreaId
     d_bgDefId
     )
     => t / nil
```

Description

Adds a reference to the specified placeArea object in the given backgroundDef. A warning message is displayed if the objects are not of the correct type.

Arguments

d_plcAreaId	Database ID of placeArea.
d_bgDefId	Database ID of backgroundDef.

Value Returned

`t`	A reference to the specified placeArea object is added to the specified backgroundDef.
`nil`	Returned in case of failure.

Example

```
dbAddPlaceAreaBackgroundDef(plcAreaId bgDefId)
```

Adds a reference to the placement area object `plcAreaId` in the given background definition `bgDefId`.

© 2020 All Rights Reserved.

dbAddPlaceAreaCompTypeSetDef

```
dbAddPlaceAreaCompTypeSetDef(
     d_placeAreaId
     d_compTypeSetDefId
     )
     => t / nil
```

Description

Adds a reference in the specified placeArea to the specified compTypeSetDefs. A warning message is displayed if the objects are not of the correct type.

Arguments

d_placeAreaId	Database ID of a placeArea.
d_compTypeSetDefId	Database ID of a compTypeSetDefs.

Value Returned

`t`	The reference was added successfully.
`nil`	An error occurred during execution.

Example

```
;; create placement area
     placeArea = dbCreatePlaceArea(cellViewid
     name
     width
     height
     origin
     orient
     utilization)

;; create compTypeSetDef
def = dbCreateCompTypeSetDef(cellViewId
     "myCompTypeSetDef"
     "bottomAlignment"
     "bottomAlignment"
     list("R0" "MX")
     list("Comptype1" "CompType2")
     offsetFromRowReference
     componnentPitch)

;; Add reference to compTypeSetDef in placeArea
dbAddPlaceAreaCompTypeSetDef(placeArea def)
```


© 2020 All Rights Reserved.

dbAddPlaceAreaRailDef

```
dbAddPlaceAreaRailDef(
     d_placeAreaId
     d_railDefId
     )
     => t / nil
```

Description

Adds a reference in the specified placeArea to the specified railDef. A warning message is displayed if the objects specified are not of the correct type.

Arguments

d_placeAreaId	Database ID of a placeArea.
d_railDefId	Database ID of a railDef.

Value Returned

`t`	The reference was added successfully.
`nil`	An error occurred during execution.

Example

```
;; create placement area
placeArea = dbCreatePlaceArea(cellViewid
     name
     width
     height
     origin
     orient
     utilization)

;; create railDef
def = dbCreateRailDef(cellViewId
     "myRailDef"
     "netName"
     list("layer" "purpose")
     "bottomAlignment"
     "bottomAlignment"
     railWidth
     offsetFromRowReference)
```


© 2020 All Rights Reserved.

```
;; Add reference to railDef in placeArea
dbAddPlaceAreaRailDef(placeArea def)
```


© 2020 All Rights Reserved.

dbCreatePlaceArea

```
dbCreatePlaceArea(
     d_cellViewId
     t_name
     g_width
     g_height
     l_origin
     t_orientation
     [ f_utilization ]
     )
     => d_IdOfCustomPlacementArea / nil
```

Description

Creates a custom placeArea in the specified cellview. A custom placeArea is a row structure used for placing device-level instances.

Arguments

d_cellViewId	Database ID of a cellview.
t_name	Name of a placeArea.
g_width	Width of the placeArea in user units.
g_height	Height of the placeArea in user units.
l_origin	Origin of the placeArea.
t_orientation	Orientation of the placeArea.
f_utilization	Specifies the utilization of the placeArea. The value is should be limited to [0.0,1.0], where 1.0 == 100% utilization.

Value Returned

d_IdOfCustomPlacementArea	Database ID of the placeArea is returned.
nil	An error occurred during execution.

Example

```
placeArea = dbCreatePlaceArea(geGetEditCellView() "myPlaceArea" 100.0 10.0 "R0")
```


© 2020 All Rights Reserved.

dbGetCellViewPlaceArea

```
dbGetCellViewPlaceArea(
     d_cellviewId
     )
     => l_customPlacementAreas / nil
```

Description

Returns a list of custom placeAreas in the specified cellview.

Arguments

d_cellviewIdy	Database ID of the cellview.

Value Returned

l_customPlacementAreas	A list of custom placeAreas is returned.
nil	The cellview ID is invalid or no custom placeAreas exist.

Example

```
myCustomPlaceAreas = dbGetCellViewPlaceArea(geGetEditCellView())
```


© 2020 All Rights Reserved.

dbGetPlaceAreaBackgroundDef

```
dbGetPlaceAreaBackgroundDef(
     d_plcAreaId
     )
     => l_bgDefId / nil
```

Description

Returns a list of database IDs of the backgroundDefs in the specified placeArea.

Arguments

d_placeAreaId	Database ID of a placeArea.

Value Returned

l_bgDefId	List of database IDs of backgroundDefs.
nil	Returned in case of failure.

Example

```
dbGetPlaceAreaBackgroundDef(placeAreaId)
```

Returns a list of database IDs of the background definitions in the placement area `placeAreaId.`

© 2020 All Rights Reserved.

dbGetPlaceAreaCompTypeSetDef

```
dbGetPlaceAreaCompTypeSetDef(
     d_customPlacementArea
     )
     => l_dbCompTypeSetDefs / nil
```

Description

Returns a list of dbCompTypeSetDefs referenced in the specified placeArea.

Arguments

d_customPlacementArea

Database ID of the custom placeArea.

Value Returned

l_dbCompTypeSetDefs

A list of dbCompTypeSetDefs is returned.

`nil` The custom placeArea ID is invalid or no dbCompTypeSetDefs exist.

Example

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView()))
myCompTypeSetDefs = dbGetPlaceAreaCompTypeSetDef(placeArea)
```


© 2020 All Rights Reserved.

dbGetPlaceAreaOrient

```
dbGetPlaceAreaOrient(
     d_customPlacementArea
     )
     => s_orientation / nil
```

Description

Returns the orientation of the specified custom placeArea.

Arguments

l_dbCompTypeSetDefs

Database ID of the custom placeArea.

Value Returned

s_orientation	The orientation of the custom placeAreas is returned.
nil	The custom placeArea ID is invalid

Example

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView())
dbGetPlaceAreaOrient(placeArea)
```


© 2020 All Rights Reserved.

dbGetPlaceAreaRailDef

```
dbGetPlaceAreaRailDef(
     d_customPlacementArea
     )
     => l_dbRailDefs / nil
```

Description

Returns a list of dbRailDefs referenced in the specified custom placeArea.

Arguments

d_customPlacementArea

Database ID of a custom placeArea.

Value Returned

l_dbRailDefs	A list of dbRailDefs is returned.
nil	The custom placeArea ID is invalid

Example

```
row = car(dbGetCellViewPlaceArea(geGetEditCellView()))
myRailDefs = dbGetPlaceAreaRailDef(row)
```


© 2020 All Rights Reserved.

dbGetPlaceAreaUtilization

```
dbGetPlaceAreaUtilization(
     d_customPlacementArea
     )
     => f_utilizationValue / nil
```

Description

Returns the assigned placement utilization for the specified custom placeArea.

Arguments

d_customPlacementArea

Database ID of a custom placeArea.

Value Returned

f_utilizationValue	The assigned utilization value is returned.
nil	The custom placeArea ID is invalid

Example

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView())
assignedUtilization = dbGetPlaceAreaUtilization(placeArea)
```


© 2020 All Rights Reserved.

dbGetPlacementGrid

```
dbGetPlacementGrid(
     d_cellviewId
     )
     => l_placementGridInfo / nil
```

Description

Returns the placement grid information for the specified cellview.

Arguments

d_cellviewId	Database ID of the cellview.

Value Returned

l_placementGridInfo	The origin of the placement grid, and the pitch X and Y values are returned.
nil	The cellview ID is invalid

Example

The following returns the X origin, Y origin, X pitch, and Y pitch

```
dbGetPlacementGrid(geGetEditCellView())
```

```
((0.0 0.0) 10.0 10.0)
```


© 2020 All Rights Reserved.

dbRemovePlaceAreaBackgroundDef

```
dbRemovePlaceAreaBackgroundDef(
     d_plcAreaId
     d_bgDefId
     )
     => t / nil
```

Description

Removes the association between the specified placeArea and backgroundDef. The background shape on the placeArea is also removed.

Arguments

d_plcAreaId	Database ID of a placeArea.
d_bgDefId	Database ID of a backgroundDef.

Value Returned

`t`	Association between the given placeArea and backgroundDef is successfully removed.
`nil`	Returned in case of failure.

Example

```
; Create placeArea
         placeArea = dbCreatePlaceArea(cv "placeArea" 100 150 list(0 0) "R0")

         ;; Create BackgroundDef and add association with the placeArea.
        bgDef = dbCreateBackgroundDef(cv "bgDef" list("FinFet" "region1") "bgNet"
10 20 30 40)
         dbAddPlaceAreaBackgroundDef(placeArea bgDef)

         ;; Remove the association with the placeArea.
         dbRemovePlaceAreaBackgroundDef(placeArea bgDef)
```


© 2020 All Rights Reserved.

dbRemovePlaceAreaCompTypeSetDef

```
dbRemovePlaceAreaCompTypeSetDef(
     d_placeAreaId
     d_compTypeSetDefId
     )
     => t / nil
```

Description

Removes a reference to the specified compTypeSetDef in the specified placeArea. A warning message is displayed if the objects are not of the correct type.

Arguments

d_placeAreaId	Database ID of a placeArea.
d_compTypeSetDefId	Database ID of a compTypeSetDefs.

Value Returned

`t`	The reference was removed successfully.
`nil`	An error occurred during execution.

Example

```
;; create placement area
placeArea = dbCreatePlaceArea(cellViewid
     name
     width
     height
     origin
     orient
     utilization)

;; create compTypeSetDef
def = dbCreateCompTypeSetDef(cellViewId
     "myCompTypeSetDef"
     "bottomAlignment"
     "bottomAlignment"
     list("R0" "MX")
     list("Comptype1" "CompType2")
     offsetFromRowReference
     componnentPitch)
```


© 2020 All Rights Reserved.

```
;; Add reference to compTypeSetDef in placeArea
dbAddPlaceAreaCompTypeSetDef(placeArea def)

;; removes reference
dbRemovePlaceAreaCompTypeSetDef(placeArea def)
```


© 2020 All Rights Reserved.

dbRemovePlaceAreaRailDef

```
dbRemovePlaceAreaRailDef(
     d_placeAreaId
     d_railDefId
     )
     => t / nil
```

Description

Removes a reference to the specified railDef in the specified placeArea. The rail geometry is also deleted. A warning message is displayed if the objects are not of the correct type.

Arguments

d_placeAreaId	Database ID of a placeArea.
d_railDefId	Database ID of a railDef.

Value Returned

`t`	The reference was removed successfully.
`nil`	An error occurred during execution.

Example

```
;; create placeArea
placeArea = dbCreatePlaceArea(cellViewid
     name
     width
     height
     origin
     orient
     utilization)

;; create railDef
def = dbCreateRailDef(cellViewId
     "myRailDef"
     "netName"
     list("layer" "purpose")
     "bottomAlignment"
     "bottomAlignment"
     railWidth
     offsetFromRowReference)
```


© 2020 All Rights Reserved.

```
;; Add reference to railDef in placeArea
dbAddPlaceAreaRailDef(placeArea def)

;; removes reference and rail geometry
dbRemovePlaceAreaRailDef(placeArea def)
```


© 2020 All Rights Reserved.

dbSetPlaceAreaOrient

```
dbSetPlaceAreaOrient(
     d_customPlacementArea
     s_orientation
     )
     => t / nil
```

Description

Sets the orientation of the specified custom placeArea.

Arguments

d_customPlacementArea	Database ID of a custom placeArea.
s_orientation	The orientation of the custom placeAreas.

Value Returned

`t`	The orientation is set successfully.
`nil`	The custom placeArea ID is invalid

Example

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView())
dbSetPlaceAreaOrient(placeArea "R90")
```


© 2020 All Rights Reserved.

dbSetPlaceAreaUtilization

```
dbSetPlaceAreaUtilization(
     d_customPlacementArea
     f_utilizationValue
     )
     => t / nil
```

Description

Sets the assigned placement utilization for the specified custom placeArea.

Arguments

d_customPlacementArea	Database ID of the custom placeArea.
f_utilizationValue	The value is should be limited to [0.0,1.0], where 1.0 == 100% utilization.

Value Returned

`t`	The utilization value is set successfully.
`nil`	The custom placeArea ID is invalid

Example

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView())
dbSetPlaceAreaUtilization(placeArea .8)
```


© 2020 All Rights Reserved.

dbSetPlacementGrid

```
dbSetPlacementGrid(
     d_cellviewId
     l_origin
     f_pitchX
     f_pitchY
     )
     => t / nil
```

Description

Sets the placement grid information. The origin of the placement grid, and the X pitch and Y values are set for the specified cellview.

Note: To unset the placement grid, call `dbSetPlacementGrid` with either *`f_pitchX`* or *`f_pitchY`* or both set to zero.

Arguments

`d_cellviewId`	Database ID of the cellview.
`l_origin`	The origin of the placement grid.
`f_pitchX`	The X pitch of the placement grid.
`f_pitchY`	The Y pitch of the placement grid.

Value Returned

`t`	The placement grid values are set successfully.
`nil`	The placement grid values are invalid

Example

```
dbSetPlacementGrid(geGetEditCellView() 0:0 10.0 10.0)
```


© 2020 All Rights Reserved.

Row-based Functions

dbAddRowBackgroundDef

```
dbAddRowBackgroundDef(
     d_rowId
     d_bgDefId
     )
     => t / nil
```

Description

Adds a reference to the specified row object in the given backgroundDef. A warning message is displayed if the objects are not of the correct type.

Arguments

d_rowId	Database ID of the row.
d_bgDefId	Database ID of the backgroundDef.

Value Returned

`t`	A reference to the specified row object is added to the specified backgroundDef.
`nil`	Returned in case of failure.

Example

```
dbAddRowBackgroundDef(rowId bgDefId)
```

Adds a reference to the placement area object `rowId` in the given background definition `bgDefId`.

© 2020 All Rights Reserved.

dbAddRowCompTypeSetDef

```
dbAddRowCompTypeSetDef(
     d_rowId
     d_compTypeSetDefId
     )
     => t / nil
```

Description

Adds a reference in the specified compTypeSetDef to the specified row object. A warning message is displayed if the objects are not of the correct type.

Arguments

d_row_Id	Database ID of a row.
d_compTypeSetDefId	Database ID of a compTypeSetDef.

Value Returned

t	The reference was added successfully.
nil	An error occurred during execution.

Example

```
;; create row
row = dbCreateRow(cellViewid
     siteDef
     name
     origin
     numSites)

;; create compTypeSetDef
def = dbCreateCompTypeSetDef(cellViewId
     "myCompTypeSetDef"
     "bottomAlignment"
     "bottomAlignment"
     list("R0" "MX")
     list("Comptype1" "CompType2")
     offsetFromRowReference
     componnentPitch)

;; Add reference to compTypeSetDef in row
dbAddRowCompTypeSetDef(row def)
```


© 2020 All Rights Reserved.

dbGetRowBackgroundDef

```
dbGetRowBackgroundDef(
     d_rowId
     )
     => l_bgDefId / nil
```

Description

Returns a list of backgroundDefs in the specified custom row.

Arguments

d_rowId	Database ID of a custom row.

Value Returned

l_bgDefID	List of backgroundDefs in the specified custom row.
nil	Returned in case of failure.

Example

```
; Create custom row.
            row = dbCreateRow(cv siteDef "row" list(3 3) 10 "R0")

         ;; Add backgroundDef to custom row.
            dbAddRowBackgroundDef(row bgDef1)
            dbAddRowBackgroundDef(row bgDef2)

         ;; Get the list of backgroundDefs on the custom row.
            dbGetRowBackgroundDef(row)
```


© 2020 All Rights Reserved.

dbGetRowCompTypeSetDef

```
dbGetRowCompTypeSetDef(
     d_rowId
     )
     => l_compTypeSetDefs / nil
```

Description

Returns a list of dbCompTypeSetDefs referenced in the specified row.

Arguments

d_rowId	Database ID of the row.

Value Returned

l_dbCompTypeSetDefs	A list of dbCompTypeSetDefs is returned.
nil	The row ID is invalid

Example

```
row = car(geGetEditCellView()~>rows)

myCompTypeSetDefs = dbGetRowCompTypeSetDef(row)
```


© 2020 All Rights Reserved.

dbAddRowRailDef

```
dbAddRowRailDef(
     d_rowId
     d_railDefId
     )
     => t / nil
```

Description

Adds a reference in the specified railDef to the specified row object. A warning message is displayed if the objects are not of the correct type.

Arguments

d_rowId	Database ID of a row.
d_railDefId	Database ID of a railDef.

Value Returned

`t`	The reference was added successfully.
`nil`	An error occurred during execution.

Example

```
;; create row
row = dbCreateRow(cellViewid
     siteDef
     name
     origin
     numSites)

;; create railDef
def = dbCreateRailDef(cellViewId
     "myRailDef"
     "netName"
     list("layer" "purpose")
     "bottomAlignment"
     "bottomAlignment"
     railWidth
     offsetFromRowReference)


;; Add reference to railDef in row
dbAddRowRailDef(row def)
```


© 2020 All Rights Reserved.

dbGetRowRailDef

```
dbGetRowRailDef(
     d_rowId
     )
     => l_dbRailDefs / nil
```

Description

Returns a list of dbRailDefs referenced in the specified row.

Arguments

d_rowId	Database ID of the row.

Value Returned

l_dbRailDefs	A list of dbRailDefs is returned.
nil	The row ID is invalid

Example

```
row = car(geGetEditCellView()~>rows)
myRailDefs = dbGetRowRailDef(row)
```


© 2020 All Rights Reserved.

dbRemoveRowCompTypeSetDef

```
dbRemoveRowCompTypeSetDef(
     d_rowId
     d_compTypeSetDefId
     )
     => t / nil
```

Description

Removes a reference to the specified compTypeSetDef in the specified row. A warning message is displayed if the objects are not of the correct type.

Arguments

d_rowId	Database ID of a row.
d_compTypeSetDefId	Database ID of a compTypeSetDef.

Value Returned

t	The reference was removed successfully.
nil	An error occurred during execution.

Example

```
;; create row
row = dbCreateRow(cellViewid
     siteDef
     name
     origin
     numSites)
;; create compTypeSetDef
def = dbCreateCompTypeSetDef(cellViewId
     "myCompTypeSetDef"
     "bottomAlignment"
     "bottomAlignment"
     list("R0" "MX")
     list("Comptype1" "CompType2")
     offsetFromRowReference
     componnentPitch)
;; Add reference to compTypeSetDef in row
dbAddRowCompTypeSetDef(row def)
;; removes reference
dbRemoveRowCompTypeSetDef(row def)
```


© 2020 All Rights Reserved.

dbRemoveRowBackgroundDef

```
dbRemoveRowBackgroundDef(
     d_rowId
     d_bgDefId
     ) => t / nil
```

Description:

Removes the given backgroundDef from the specified custom row. Removing a backgroundDef also removes the background shape associated with the backgroundDef from the custom row.

Arguments

d_rowId	Database ID of the custom row.
d_bgDefID	Database ID of the backgroundDef.

Value Returned

`t`	Returned when backgroundDef is successfully removed from the custom row.
`nil`	Returned in case of failure.

Example

```
; create custom row
         row = dbCreateRow(cv siteDef "row" list(3 3) 10 "R0")

         ;; Adding backgroundDef to the row.
         dbAddRowBackgroundDef(row bgDef)

         ;; removing backgroundDef from the row.
         dbRemoveRowBackgroundDef(row bgDef)
```


© 2020 All Rights Reserved.

dbRemoveRowRailDef

```
dbRemoveRowRailDef(
     d_rowId
     d_railDefId
     )
     => t / nil
```

Description

Removes a reference to the specified railDef in the specified row. The rail geometry is also deleted. A warning message is displayed if the objects are not of the correct type.

Arguments

d_rowId	Database ID of a row.
d_railDefId	Database ID of a railDef area.

Value Returned

`t`	The reference was removed successfully.
`nil`	An error occurred during execution.

Example

```
;; create row
row = dbCreateRow(cellViewid
     siteDef
     name
     origin
     numSites)
;; create railDef
> def = dbCreateRailDef(cellViewId
     "myRailDef"
     "netName"
     list("layer" "purpose")
     "bottomAlignment"
     "bottomAlignment"
     railWidth
     offsetFromRowReference)
;; Add reference to RailDef in row
dbAddRowRailDef(row def)
;; removes reference and rail geometry
dbRemoveRowRailDef(row def)
```


© 2020 All Rights Reserved.

Row Region Specification Functions (ICADVM20.1 Only)

A rowRegionSpec is a template, which is stored in a cellview, and is used for building a rowRegion. This template contains all the attributes to fully describe all rows in a rowRegion.

dbCreateRowRegionSpec

```
dbCreateRowRegionSpec(
     d_cvId
     t_name
     { n_height | d_siteDef }
     l_rowAttrs
     [ l_refGridNames ]
     [ l_refGridOffsets ]
     [ t_relSnapPatternName ]
     [ t_type ]
     )
     => d_specId / nil
```

Description

(ICADVM20.1 Only) Creates rowRegionSpec in the specified cellview.

You can create specifications to create two types of rows:

- dbRow - Needs siteDef from techDB
- dbPlaceArea - Does not need siteDef from techDB

Arguments

d_cvId	Database ID of a cellview.
t_name	Name of rowRegionSpec.

© 2020 All Rights Reserved.

n_height | d_siteDef

Specify *n_height* to create dbPlaceArea or specify *d_siteDef* to create dbRow, where,

- *n_height*

  Height of one complete instance of the region specification used to create a dbPlaceArea.

- *d_siteDef*

  Site definition to create a row.

l_rowAttrs

List of placement row specifications. Row attributes include the following:

- Required Attributes

```
row1 = list(nil)
row1->name = "n"
row1->offset = 0.02
row1->orient = "R0"  ; or "MX"
```

- Attribute to create dbPlaceArea

```
row1->height = 0.768
```

- Attribute to create a dbRow

```
row1->siteDef = "PP_90M2_12"
```

- Optional Attributes

```
row1->railAttrs = list(pwrRail1 …)
row1->backgroundAttrs = list(bckAttr1 …)
row1->compAttrs = list(compAttr1 …)
```

l_refGridNames

List of names of reference grids in the X and Y direction

l_refGridOffset

List of the offsets from the reference grids.

t_relSnapPatternName

Name of relatedSnapPattern.

t_type

A description string specified by the user. This string can be used in the rowRegion fill flow to group rowRegionSpecs.

© 2020 All Rights Reserved.

Value Returned

d_specID	Database ID of the rowRegionSpec that is created.
nil	Returned in case of failure.

Examples

There are two ways in which you can specify this function. These are illustrated in the examples below.

Example 1

The following example uses external DPLs, where, the DPLs are defined first and then passed to the DB function.

```
; Create Rail Attributes
    gndRail = list(nil)
    gndRail->lpp = list("gndLayer" "drawing")
    gndRail->width = 0.02
    gndRail->refOnArea = "topAlignment"
    gndRail->refOnRail = "centerAlignment"
    gndRail->netName = "gndNet"
    gndRail->railFigType = "pathSeg"

;; Create compType attributes
    compAttr = list(nil)
    compAttr->refOnArea  = "centerAlignment"
    compAttr->refOnComp  = "anyAlignment"
    compAttr->allowedTypes = list("compType1" "compType2")
    compAttr->orients = list("R0")
    compAttr->compFilters = list(list("cellLib" "nf*" "layout")
    list("cellLib" "ng*" "layout"))

;; Create Background Attributes
    bckAttr = list(nil)
    bckAttr->lpp = list("bckLayer" "drawing")
    bckAttr->netName = "bkNet"
    bckAttr->leftEnc = 1
    bckAttr->rightEnc = 0.8
    bckAttr->topEnc = 0.9
    bckAttr->botEnc = 1.1
```


© 2020 All Rights Reserved.

```
;; Create Row Attributes
    row = list(nil)
    row->name = "placeRow"
    row->offset = 3.0
    row->orient = "MX"
    row->siteDef = "CORE"
    row->railAttrs = list(gndRail)
    row->backgroundAttrs = list(bckAttr)
    row->compAttrs = list(compAttr)

;; Create rowRegionSpec
    dbCreateRowRegionSpec(cv "rowRegionSpec" 2.1 list(row))
```

Example 2

This example uses inline DPLs, where the DPLs are specified within the DB function call.

```
dbCreateRowRegionSpec(cv
    "rowRegionSpec2"
    2.1
    '(
    (nil
    name "placeRow"
    offset 3.0
    orient "MX"
    siteDef "CORE"
    railAttrs (
        (nil
        lpp         ("gndLayer" "drawing")
        width       0.02
        refOnArea   "topAlignment"
        refOnRail   "centerAlignment"
        netName     "gndNet"
        railFigType "pathSeg"
        )
        )

    backgroundAttrs (
        (nil
        lpp      ("bckLayer" "drawing")
        netName  "bkNet"
        leftEnc  1
```


© 2020 All Rights Reserved.

```
        rightEnc 0.8
        topEnc   0.9
        botEnc   1.1
        )
        )
compAttrs (
        (nil
        refOnArea    "centerAlignment"
        refOnComp    "anyAlignment"
        allowedTypes ("compType1" "compType2")
        orients      ("R0")
        compFilters  (("tn010" "nf*" "layout") ("tn010" "ng*" "layout"))
        )
        )
)
)
)
```


© 2020 All Rights Reserved.

dbFindRowRegionSpec

```
dbFindRowRegionSpec(
     d_cvId
     t_name
     )
     => d_specId / nil
```

Description

(ICADVM20.1 Only) Searches for a rowRegionSpec with the specified name in the given cellview.

Arguments

d_cellviewId	Database ID of a cellview.
t_name	Name of a rowRegionSpec.

Value Returned

d_rowRegionSpecId	Database ID of the searched rowRegionSpec, if found.
nil	Returned in case of failure.

Example

```
dbFindRowRegionSpec(cv "rowRegion")
```

Searches for the rowRegionSpec `rowRegion` in the cellview `cv`.

© 2020 All Rights Reserved.

dbGetCellViewRowRegionSpecs

```
dbGetCellViewRowRegionSpecs(
     d_cellviewId
     )
     => l_rowRegionSpecs / nil
```

Description

Returns a list of rowRegionSpecs in the specified cellview.

Arguments

d_cellviewId	Database ID of a cellview.

Value Returned

l_rowRegionSpecs	List of database IDs of the rowRegionSpecs in the given cellview.
nil	Returned in case of failure.

Example

```
dbGetCellViewRowRegionSpecs(cv)
```

Returns a list of RowRegionSpecs in the cellview `cv`.

© 2020 All Rights Reserved.

Row Region Functions (ICADVM20.1 Only)

A rowRegion is a figGroup that is created and edited on the canvas. It references a rowRegionSpec.

A rowRegion should match its corresponding rowRegionSpec. However, if the rowRegionSpec is modified, all the rowRegions that reference it will no longer match their specification. In this case, you need to use the dbRebuildRowRegion function to rebuild the rowRegion to match its specification

dbAttachRowRegionToPRBoundary

```
dbAttachRowRegionToPRBoundary(
     d_rowRegionId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Associates the given rowRegion with the PRBoundary in the same block. If the PRBoundary is assigned to another rowRegion, it is first detached from that region before being assigned to the specified rowRegion.

Arguments

d_rowRegionId	Database ID of a rowRegion.
d_bgDefId	Database ID of a backgroundDef.

Value Returned

`t`	The given rowRegion is attached to the specified PRBoundary.
`nil`	Returned in case of failure.

Example

```
;Create PRBoundary
dbCreatePRBoundary(cv list(list(0 0) list(1.1 0) list(1.1 1.1) list(0 1.1)))
;Create rowRegion
dbCreateRowRegion(cvId "rowRegion" "rowRegionSpec" list(list(0 0) list(2000 0)
list(2000 5000) list(0 5000)))
```


© 2020 All Rights Reserved.

```
;Attach rowRegion to PRBoundary
dbAttachRowRegionToPRBoundary(rowRegion)
```


© 2020 All Rights Reserved.

dbCreateRowRegion

```
dbCreateRowRegion(
     d_cvId
     t_name
     t_specName
     l_points
     )
     => d_rowRegionId/nil
```

Description

(ICADVM20.1 Only) Creates a rowRegion in the specified cellview. The contents of the rowRegion are created based on the template defined in the rowRegionSpec.

Note: This function supports only non-concave rectilinear polygons.

Arguments

d_cellviewID	Database ID of a cellview.
t_name	Name of a rowRegion.
t_specName	Name of a rowRegionSpec.
l_points	List of the points that define the areaBoundary enclosing the rowRegion.

Value Returned

d_rowRegionId	Database ID of rowRegion that is created.
nil	Returned in case of failure.

Example

```
dbCreateRowRegion(cvId "rowRegion" "rowRegionSpec" list(list(0 0) list(2000 0)
list(2000 5000) list(0 5000)))
```

Creates a row region named `rowRegion` in the cellview `cvId` based on the information in `rowRegionSpec`.

© 2020 All Rights Reserved.

dbDetachPRBoundaryFromRowRegion

```
dbDetachPRBoundaryFromRowRegion(
     d_PRBoundaryId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Detaches the PRBoundary from any associated rowRegion.

Arguments

d_PRBoundaryId	Datbase ID of PRBoundary.

Value Returned

t	PRBoundary is detached from the rowRegion.
nil	Returned in case of failure.

Example

```
; Create RowRegion and PRBoundary.
         rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point)
       prb = dbCreatePRBoundary(cv list(list(0 0) list(100 0) list(100 100) list(0
100)))

         ;; Attach RowRegion to PRBoundary and detach PRBoundary from RowRegion.
         dbAttachRowRegionToPRBoundary(rowRegion)
         dbDetachPRBoundaryFromRowRegion(prb)
```


© 2020 All Rights Reserved.

dbFindRowRegion

```
dbFindRowRegion(
     d_cvId
     t_name
     )
     => d_rowRegionId / nil
```

Description

(ICADVM20.1 Only) Searches for a rowRegion by the specified name in the given cellview.

Arguments

d_cvId	Database ID of a cellview.
t_name	Name of a rowRegion to search.

Value Returned

d_rowRegionId	Database ID of the searched rowRegion, if found.
nil	Returned in case of failure.

Example

```
;Create rowRegion
         rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point)

         ;; Find rowRegion by name.
         dbFindRowRegion(cv "rowRegion")
```


© 2020 All Rights Reserved.

dbFlattenRowRegion

```
dbFlattenRowRegion(
     d_rowRegionId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Flattens the rowRegion by removing information related to the associated figGroup and areaBoundary, and retaining information related to rows and other shapes.

Arguments

d_rowRegionId	Database ID of the rowRegion to be flattened.

Value Returned

`t`	The rowRegion is successfully flattened.
`nil`	Returned in case of failure.

Example

```
dbFlattenRowRegion(rowRegion)
=> t
```

The specified row region `rowRegion` is successfuly flattened.

© 2020 All Rights Reserved.

dbGetCellViewRowRegions

```
dbGetCellViewRowRegions(
     d_cellviewId
     )
     => l_rowRegionSpecs / nil
```

Description

(ICADVM20.1 Only) Returns a list of rowRegions in the specified cellview.

Arguments

d_cellviewId	Database ID of a cellview.

Value Returned

l_rowRegionSpecs	List of database IDs of rowRegion in the given cellview.
nil	Returned in case of failure.

Example

```
dbGetcellviewRowRegion(cv)
```

Returns a list of rowRegions in the cellview `cv`.

© 2020 All Rights Reserved.

dbGetRowRegionAttachedToPRBoundary

```
dbGetRowRegionAttachedToPRBoundary(
     d_prBoundaryId
     )
     => d_rowRegionId / nil
```

Description

(ICADVM20.1 Only) Returns the database ID of the rowRegion attached to the given PRBoundary.

Arguments

d_prBoundaryId	Database ID of a PRBoundary.

Value Returned

d_rowRegionId	Database ID of the rowRegion attached to the specified PRBoundary.
nil	Returned when no rowRegion is attached to the PRBoundary.

Example

```
;; Create RowRegion and PRBoundary.
         rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point)
       prb = dbCreatePRBoundary(cv list(list(0 0) list(100 0) list(100 100) list(0
100)))

        ;; Attach RowRegion to PRBoundary and query the rowRegion ID attached to
the PRBoundary
         dbAttachRowRegionToPRBoundary(rowRegion)
         dbGetRowRegionAttachedToPRBoundary(prb)
```


© 2020 All Rights Reserved.

dbGetRowRegionPlaceRows

```
dbGetRowRegionPlaceRows(
     d_rowRegionId)
     => l_placeRowId / nil
```

Description

(ICADVM20.1 Only) Returns a list of all placement rows scoped to the given rowRegion. Placement rows can be a standard row or a custom placeArea object.

Arguments

d_rowRegionId	Database ID of a rowRegion.

Value Returned

placeRowId	List of database IDs for standard rows or custom placeArea objects.
nil	Returned in case of failure.

Example

```
dbGetRowRegionPlaceRows(rowRegionId)
```


© 2020 All Rights Reserved.

dbGetRowRegionPoints

```
dbGetRowRegionPoints(
     d_rowRegionId
     )
     => l_points / nil
```

Description

(ICADVM20.1 Only) Returns a list of rectilinear points that define the rowRegion boundary.

Arguments

d_rowRegionId	Database ID of a rowRegion.

Value Returned

l_points	List of the rectilinear points.
nil	Returned in case of failure.

Example

```
; Create a rowRegion and query the points.
       rowRegion = dbCreateRowRegion(cvId "rowRegion" "rowRegionSpec" list(list(0
0) list(2000
                                        0) list(2000 5000) list(0 5000)))
         dbGetRowRegionPoints(rowRegion)
```


© 2020 All Rights Reserved.

dbGetRowRegionRowRegionSpec

```
dbGetRowRegionRowRegionSpec(
     d_rowRegionId
     )
     => d_rowRegionSpecId / nil
```

Description

(ICADVM20.1 Only) Returns the rowRegionSpec from which the specified rowRegion was generated.

Arguments

d_rowRegionId	Database ID of a rowRegion.

Value Returned

d_rowRegionSpecId	Database ID of the rowRegionSpec from which the specified rowRegion was generated.
nil	Returned in the case of failure.

Example

```
dbGetRowRegionRowRegionSpec(rowRegionId)
```

Returns the row region specification from which the row region `rowRegionId` was generated.

© 2020 All Rights Reserved.

dbGetRowRegionSpecRowRegions

```
dbGetRowRegionSpecRowRegions(
     d_rowRegionSpecId
     )
     => l_rowRegions / nil
```

Description

(ICADVM20.1 Only) Returns a list of rowRegions built on the basis of the given rowRegionSpec.

Arguments

d_rowRegionSpec	Database ID of a rowRegionSpec.

Value Returned

l_rowRegionIds	List of database IDs of rowRegions.
nil	Returned in case of failure.

Example

```
dbGetRowRegionSpecRowRegions(rowRegionSpecId)
```

Returns a list of rowRegions built on the basis of the rowRegionSpec `rowRegionSpecId`.

© 2020 All Rights Reserved.

dbGetRowRegionUsesPartialRowSpec

```
dbGetRowRegionUsesPartialRowSpec(
     d_rowRegionID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Confirms whether the fill space of rowRegion should be populated with partial specialization of rowRegionSpec.

Arguments

d_rowRegionId	Database ID of a rowRegion.

Value Returned

t	Returned if the rowRegion uses partial specialization of rowRegionSpec.
nil	Returned if the rowRegion does not use partial specialization of rowRegionSpec.

Example

```
; Create rowRegion.
rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point t "R90")
; Query if the rowRegion uses partial rowRegionSpec.
         dbGetRowRegionUsesPartialRowSpec(rowRegion)
```


© 2020 All Rights Reserved.

dbIsRowRegion

```
dbIsRowRegion(
     d_Id
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Checks if the specified database object is a rowRegion.

Arguments

d_Id	Database ID of an object.

Value Returned

`t`	Returned if the object is a rowRegion.
`nil`	Returned if the object is not a rowRegion.

Example

```
rowRegion=dbCreateRowRegion(...)
dbIsRowRegion(rowRegion)
```

Returns `t` because `rowRegion` is a row region.

© 2020 All Rights Reserved.

dbIsRowRegionAttachedToPRBoundary

```
dbIsRowRegionAttachedToPRBoundary(
     d_rowRegionId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Checks whether the specified rowRegion is associated with the PRBoundary of the block that contains the rowRegion.

Arguments

d_rowRegionId	Database ID of a rowRegion database.

Value Returned

`t`	Returned if rowRegion is associated with the PRBoundary.
`nil`	Returned if rowRegion is not associated with the PRBoundary.

Example

```
;Create PRBoundary
dbCreatePRBoundary(cv list(list(0 0) list(1.1 0) list(1.1 1.1) list(0 1.1)))
;Create rowRegion
dbCreateRowRegion(cvId "rowRegion" "rowRegionSpec" list(list(0 0) list(2000 0)
list(2000 5000) list(0 5000)))
;Attach rowRegion to PRBoundary
dbAttachRowRegionToPRBoundary(rowRegion)
; Checks if d_rowRegionId is associated with the PRBoundary
dbIsRowRegionAttachedToPRBoundary(d_rowRegionId)
```

Returns `t` because the row region is attached to the PRBoundary.

© 2020 All Rights Reserved.

dbRowRegionPointsCutOut

```
dbRowRegionPointsCutOut(
     d_rowRegionId
     l_box
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Changes the rectilinear specification of the specified rowRegion by cutting out the portion of it defined by the specified box. The box cannot fully cover the current rowRegion and it cannot split it into two or more fully independent rowRegions.

Arguments

d_rowRegionId	Database ID of a rowRegion.
l_box	The box to cut out from the current rowRegion.

Value Returned

`t`	The rowRegion specification was modified successfully.
`nil`	An error occurred during execution.

Example

The following example, first defines a rectilinear box and then sets the rowRegion points. Finally, the middle of the rowRegion is cut out.

```
;Define a set of rectilinear points.
             points = list(list(0 0) list(3000 0) list(3000 5000) list(0 5000))
; Set the rowRegion points
             dbSetRowRegionPoints(rowRegion point)
; Cut out the middle of the rowRegion:
             box = list(1000:2000 2000:3000)
             dbRowRegionPointsCutOut(rowRegion box)
```


© 2020 All Rights Reserved.

dbRebuildRowRegion

```
dbRebuildRowRegion(
     d_rowRegionId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Removes contents of the specified rowRegion and then regenerates it based on the rowRegionSpec associated with it.

Arguments

d_rowRegionId	Database ID of a rowRegion.

Value Returned

`t`	Returned in case of success.
`nil`	Returned in case of failure.

Example

```
dbCreateRowRegion(cvId "rowRegion" "rowRegionSpec" list(list(0 0) list(2000 0)
list(2000 5000) list(0 5000)))
;Remove and then regenerate d_rowRegionId
dbRebuildRowRegion(rowRegion)
;; Rebuilds the rowRegion from scratch based on its rowRegionSpec.
```

For more information, see dbRowRegionMatchesSpec. When a rowRegion does not match its rowRegionSpec, then `dbRebuildRowRegion` is needed to rebuild the rowRegion based on the latest attribtues of its corresponding rowRegionSpec.

© 2020 All Rights Reserved.

dbRowRegionMatchesSpec

```
dbRowRegionMatchesSpec(
     d_rowRegionId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Checks whether the rowRegion matches its rowRegionSpec.

A rowRegion does not matches its rowRegionSpec when one of the following is true:

- The rowRegionSpec is modified after the rowRegion is created. In this case, the rowRegion should be rebuilt to match it with the new rowRegionSpec.
- The row region is modified out of context. For example, one of its rows is modified in an older version of Virtuoso.

Arguments

d_rowRegionId	Database ID of a rowRegion.

Value Returned

t	Specified rowRegion matches its rowRegionSpec.
nil	Specified rowRegion does not match its rowRegionSpec.

Example

```
;; Creating rowRegion along with the corresponding rowRegionSpec
         spec = dbCreateRowRegionSpec(cv "RRS" 1100 list(newRow newRow))
         rowRegion = dbCreateRowRegion(cv "rowRegion" "RRS" point t "MXR90")

         ;; Check if the rowRegion matches spec.
         dbRowRegionMatchesSpec(rowRegion)
         \t t

        ;; rowRegion matches the rowRegionSpec.
        dbGetRowRegionRowRegionSpec(rowRegion)~>height = 2200
        dbRowRegionMatchesSpec(rowRegion)
        \t nil
```


© 2020 All Rights Reserved.

```
;; rowRegion does not matches the spec because rowRegionSpec has been
modified.
         ;; Rebuild rowRegion
dbRebuildRowRegion(rowRegion)

;; Check if the rowRegion matches spec.
dbRowRegionMatchesSpec(rowRegion)
t t
;; rowRegion matches the spec again after it is rebuilt
```


© 2020 All Rights Reserved.

dbSetRowRegionPoints

```
dbSetRowRegionPoints(
     d_rowRegionId
     l_points
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Changes the rectilinear specification of the given rowRegion.

Arguments

d_rowRegionId	Database ID of a rowRegion.
l_points	List of rectilinear points specifying the area covered by the rowRegion.

Value Returned

`t`	Modified rowRegion points.
`nil`	Returned in case of failure.

Example

```
; Define a set of rectilinear points.
             point = list(list(0 0) list(3000 0) list(3000 5000) list(0 5000))
;; Set rowRegion points
             dbSetRowRegionPoints(rowRegion point)
```


© 2020 All Rights Reserved.

dbSetRowRegionRowRegionSpec

```
dbSetRowRegionRowRegionSpec(
     d_rowRegionId
     d_rowRegionSpecId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the given rowRegionSpec as the template for the specified rowRegion.

Arguments

d_rowRegionId	Database ID of a rowRegion.
d_rowRegionSpecId	Database ID of a rowRegionSpec.

Value Returned

`t`	Sets the given rowRegionSpec as the row template for the specified rowRegion.
`nil`	Returned in case of failure.

Example

```
; Create RowRegionSpecs
         gndRail = list(nil)
         gndRail->lpp = list("gndLayer" "drawing")
         gndRail->width = 0.02
         gndRail->refOnArea = "topAlignment"
         gndRail->refOnRail = "centerAlignment"
         gndRail->netName = "gndNet"
         gndRail->railFigType = "pathSeg"

         compAttr = list(nil)
         compAttr->refOnArea  = "centerAlignment"
         compAttr->refOnComp  = "anyAlignment"
         compAttr->allowedTypes = list("compType1" "compType2")
         compAttr->orients = list("R0")
```


© 2020 All Rights Reserved.

```
        compAttr->compFilters = list(list("cellLib" "nf*" "layout") list("cellLib"
"ng*" "layout"))

          bckAttr = list(nil)
          bckAttr->lpp = list("bckLayer" "drawing")
          bckAttr->netName = "bkNet"
          bckAttr->leftEnc = 1
          bckAttr->rightEnc = 0.8
          bckAttr->topEnc = 0.9
          bckAttr->botEnc = 1.1

          row = list(nil)
          row->name = "placeRow"
          row->offset = 3.0
          row->orient = "MX"
          row->siteDef = "CORE"
          row->railAttrs = list(gndRail)
          row->backgroundAttrs = list(bckAttr)
          row->compAttrs = list(compAttr)

          rrspec1 = dbCreateRowRegionSpec(cv "rowRegionSpec" 2.1 list(row))
          rrspec2 = dbCreateRowRegionSpec(cv "rowRegionSpec2" 4.2 list(row))


          ;; Create RowRegion and set rowRegionSpec.
        rowRegion = dbCreateRowRegion(cvId "rowRegion" "rowRegionSpec" list(list(0
0)
                                      list(2000 0) list(2000 5000) list(0 5000)))

          dbSetRowRegionRowRegionSpec(rowRegion rrspec2)
```


© 2020 All Rights Reserved.

dbSetRowRegionUsesPartialRowSpec

```
dbSetRowRegionUsesPartialRowSpec(
     d_rowRegionId
     g_value
     )
     => t / nil
```

Description

(ICADVM20.1 Only) A Boolean indicating whether the population of rowRegion is set to fill the space with partial specialization of rowRegionSpec.

Arguments

d_rowRegionId	Database ID of a rowRegion.
g_value	A Boolean that determines whether rowRegion should be set to use partial specialization of rowRegionSpec.

Value Returned

`t`	Set or unset partial specialization of rowRegionSpec.
`nil`	Returned in case of failure.

Example

```
; Create rowRegion
         rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point)

         ;; Set RowRegion to use partial rowSpec.
         dbSetRowRegionUsesPartialRowSpec(rowRegion t)
```


© 2020 All Rights Reserved.

Miscellaneous Functions (ICADVM20.1 Only)

dbGetPlaceRowVersion

```
dbGetPlaceRowVersion(
     [d_cvId]
     )
     => n_version / nil
```

Description

(ICADVM20.1 Only) Returns the version of the placement row infrastructure in the given design, or returns the software version if the cellview is not specified.

Arguments

d_cellviewId	Database ID of a cellview.

Value Returned

n_version	Version of the placement infrastructure used in the cellview or version of the software, when no cellview is specified.
nil	Returned in case of failure.

Example

```
dbGetPlaceRowVersion(cv)
```

Returns the version of the placement row infrastructure in the cellview `cv`.

```
dbGetPlaceRowVersion()
```

Returns the version of the software version.

© 2020 All Rights Reserved.

dbGetRowRegionVersion

```
dbGetRowRegionVersion(
     [d_cvId]
     )
     => n_version / nil
```

Description

(ICADVM20.1 Only) Returns the version of the rowRegion infrastructure used in the given design, or returns the software version if the cellview is not specified.

Arguments

d_cellviewId	Database ID of a cellview.

Value Returned

n_version	Version of the rowRegion infrastructure used in the cellview or version of the software, when no cellview is provided.
nil	Returned in case of failure.

Example

```
dbGetRowRegionVersion()
```

Returns the version of the software version.

```
dbGetRowRegionVersion(cvId)
```

Returns the version of the rowRegion infrastructure in the cellview `cv`.

© 2020 All Rights Reserved.

dbIsImplicit

```
dbIsImplicit(
     d_objectId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Verifies whether the object represented by the specified database ID is implicit. Supported objects: dbRailDef, dbBackgroundDef and dbCompTypeSetDef.

An implicit object is owned by a rowRegion. Implicit objects cannot be modified directly. However, they can be modified indirectly by changing their corresponding attributes in the rowRegionSpec. Otherwise, the rowRegion can be flattened and the objects owned by the rowRegion are no longer implicit.

Arguments

d_objectId	Database ID of the object to be queried.

Value Returned

`t`	The object is implicit.
`nil`	The object is not implict.

Example

```
dbIsImplicit(backgroundDefId)
=> t
```

The object `backgroundDefId` is implicit.

© 2020 All Rights Reserved.

dbIsInRowRegion

```
dbIsInRowRegion(
     d_FigId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Checks if a figure with the given figure database ID belongs to a rowRegion.

Arguments

d_figId	Database ID of a figure.

Value Returned

`t`	Figure exists in or belongs to a rowRegion.
`nil`	Figure does not belong to a rowRegion.

Example

```
; Create a rowRegion and obtain all the placeRows in the rowRegion.
         RR = dbCreateRowRegion(cv2 "RR" "RRS" point)
         placerow = car(dbGetRowRegionPlaceRows(RR))

         ;; Query that the placeRow belongs to a rowRegion.
         dbIsInRowRegion(placeRow)
```


© 2020 All Rights Reserved.

8

Topology Pattern Database Access Functions

This section describes the functions that support channel-based routing. These functions can be classified into different topology patterns and are defined to create, process, or return an object specified by a topology pattern ID.

The definitions of the common topology patterns are as given below.

`Topology Pattern` The set of rules for a channel-based router so that it automatically performs trunk-based routing. The different topology patterns are specified as follows:

- Twig
- Trunk
- Trunk Connection
- Strap

`Trunk` An interconnect connection between instTerms that runs parallel to a channel.

© 2020 All Rights Reserved.

`Twig` An interconnect connection between instTerms that runs perpendicular to a channel and connects trunks to a different instTerm.

© 2020 All Rights Reserved.

`Trunk Connection`

The connection between more than one trunks, also called a dogleg.

A vertical and a horizontal trunk are connected together using a trunk connection.

© 2020
All Rights Reserved.

`Strap` The connection between a number of instTerms. A strap is connected to a trunk using a twig.

A strap can have multiple geometric segments.

This chapter describes the following topology pattern database SKILL functions:

- dbCreateTopology
- dbDeleteTopology
- dbGetCellViewTopologies
- dbGetNetTopologies
- dbGetTopologyName
- dbGetTopologyNet
- dbGetTopologyObjects

© 2020 All Rights Reserved.

- dbGetTopologyPatternType
- dbSetTopologyName
- dbSetTopologyNet
- dbCreateTrunk
- dbDeleteTrunk
- dbGetTrunkAnchor
- dbGetTrunkDirection
- dbGetTrunkLongOffset1
- dbGetTrunkLongOffset2
- dbGetTrunkLongOffsetSource
- dbGetTrunkLongOffsetSource1
- dbGetTrunkLongOffsetSource2
- dbGetTrunkName
- dbGetTrunkOrthoOffset
- dbGetTrunkSide
- dbGetTrunkTopology
- dbGetTrunkTwigs
- dbSetTrunkAnchor
- dbSetTrunkLongOffset1
- dbSetTrunkLongOffset2
- dbSetTrunkLongOffsetSource
- dbSetTrunkLongOffsetSource1
- dbSetTrunkLongOffsetSource2
- dbSetTrunkName
- dbSetTrunkDirection
- dbSetTrunkOrthoOffset
- dbSetTrunkSide

© 2020 All Rights Reserved.

- dbCreateTwig
- dbDeleteTwig
- dbGetTwigName
- dbGetTwigObject
- dbGetTwigPinName
- dbGetShapeTwigs
- dbGetInstTermTwigs
- dbGetViaTwigs
- dbGetTwigTrunk
- dbSetTwigName
- dbSetTwigObject
- dbSetTwigPinName
- dbCreateTrunkConnect
- dbDeleteTrunkConnect
- dbGetTrunkConnectName
- dbGetTrunkConnectTopology
- dbGetTrunkConnectTrunks
- dbSetTrunkConnectName
- dbSetTrunkConnectTrunks
- dbCreateStrap
- dbDeleteStrap
- dbGetStrapName
- dbGetStrapObjects
- dbGetStrapTopology
- dbGetStrapTwig
- dbGetInstTermStraps
- dbGetViaStraps

© 2020 All Rights Reserved.

- dbGetShapeStraps
- dbAddStrapObject
- dbRemoveStrapObject
- dbSetStrapName
- dbSetStrapTwig
- dbIsTopologyPatternId
- dbNetHasTopology

© 2020 All Rights Reserved.

dbCreateTopology

```
dbCreateTopology(
     t_name
     d_netId
     )
     => d_topologyId / nil
```

Description

Creates a topology on the specified net with the given name. The net must be a single-bit net.

Arguments

t_name	The name of the new topology.
d_netId	The single-bit net ID to associate with the new topology.

Value Returned

d_topologyId	The ID of the new topology.
nil	The topology creation failed.

Example

```
top = dbCreateTopology("top" netA)
```


© 2020 All Rights Reserved.

dbDeleteTopology

```
dbDeleteTopology(
     d_topologyId
     )
     => t / nil
```

Description

Deletes the specified topology, which includes removing the association it has with the net, and deleting the associated trunk, trunk connect pattern, and strap pattern.

Arguments

d_topologyId	The topology ID to be deleted.

Value Returned

`t`	The topology was deleted.
`nil`	The topology deletion failed.

Example

```
dbDeleteTopology(top)
```


© 2020 All Rights Reserved.

dbGetCellViewTopologies

```
dbGetCellViewTopologies(
     d_cellViewId
     )
     => l_topologyIds / nil
```

Description

Returns a list of topologies in the specified cellview.

Arguments

d_cellViewId	The cellview ID whose list of topologies must be returned.

Value Returned

l_topologyIds	The list of topologies associated with the specified cellview.
nil	The list of topologies could not be returned.

Example

```
dbGetCellViewTopologies(cv)
```


© 2020 All Rights Reserved.

dbGetNetTopologies

```
dbGetNetTopologies(
     d_netId
     )
     => l_topologyIds / nil
```

Description

Returns a list of topologies associated with the specified net.

Arguments

d_netId	The net ID whose list of topologies must be returned.

Value Returned

l_topologyIds	The list of topologies associated with the specified net.
nil	The net does not have any associated topology pattern.

Example

```
dbGetNetTopologies(netA)
```


© 2020 All Rights Reserved.

dbGetTopologyName

```
dbGetTopologyName(
     d_topologyId
     )
     => t_name / nil
```

Description

Returns the name of the topology associated with the specified topology ID.

Arguments

d_topologyId	The topology ID whose name must be returned.

Value Returned

t_name	The name of the specified topology ID.
nil	The specified topology ID is incorrect.

Example

```
dbGetTopologyName(top)
```


© 2020 All Rights Reserved.

dbGetTopologyNet

```
dbGetTopologyNet(
     d_topologyId
     )
     => t_name / nil
```

Description

Returns the net associated with the specified topology ID.

Arguments

d_topologyId	The topology ID whose net must be returned.

Value Returned

t_name	The name of the net associated with the specified topology ID.
nil	The specified topology ID is incorrect.

Example

```
dbGetTopologyNet(top)
```


© 2020 All Rights Reserved.

dbGetTopologyObjects

```
dbGetTopologyObjects(
     d_topologyId
     )
     => l_patterns / nil
```

Description

Returns the list of trunks, trunk connects, and straps owned by the specified topology.

Arguments

d_topologyId	The topology ID whose list of trunks, trunk connects, and straps must be returned.

Value Returned

l_patterns	The list of trunks, trunk connects, and straps associated with the specified topology ID.
nil	The specified topology ID is incorrect.

Example

```
dbGetTopologyObjects(top)
```


© 2020 All Rights Reserved.

dbGetTopologyPatternType

```
dbGetTopologyPatternType(
     d_patternId
     )
     => t_tpGroupType / nil
```

Description

Returns the type of the specified topology pattern ID. The valid types are topology, trunkConnect, trunk, twig, strap, and generic.

Arguments

d_patternId	The topology pattern ID whose group type must be returned.

Value Returned

t_tpGroupType	The type of the topology pattern associated with the specified topology pattern ID.
nil	The specified topology pattern ID is incorrect.

Example

```
dbGetTopologyPatternType(top)
"topology"
dbGetTopologyPatternType(trunkConn)
"trunkConnect"
dbGetTopologyPatternType(trunk)
"trunk"
dbGetTopologyPatternType(twig)
"twig"
dbGetTopologyPatternType(strap)
"strap"
```


© 2020 All Rights Reserved.

dbSetTopologyName

```
dbSetTopologyName(
     d_topologyId
     t_name
     )
     => t / nil
```

Description

Sets the name of the specified topology ID.

Arguments

d_topologyId	The topology ID whose name must be changed.
t_name	The new name of the specified topology ID.

Value Returned

`t`	The specified new name was set for the specified topology ID.
`nil`	The specified new name was not set for the specified topology ID.

Example

```
dbSetTopologyName(top, "newTopName")
```


© 2020 All Rights Reserved.

dbSetTopologyNet

```
dbSetTopologyNet(
     d_topologyId
     d_netId
     )
     => t / nil
```

Description

Changes the net associated with the specified topology. The net must be a single-bit net.

Arguments

d_topologyId	The topology ID whose net association must be changed.
d_netId	The net ID to be associated with the specified topology ID.

Value Returned

`t`	The net association of the topology was changed with the specified net ID.
`nil`	The net association of the topology was not changed with the specified net ID.

Example

```
dbSetTopologyNet(top, netB)
```


© 2020 All Rights Reserved.

dbCreateTrunk

```
dbCreateTrunk(
     t_name
     d_topologyId
     t_direction
     )
     => d_trunkId / nil
```

Description

Creates a trunk pattern on the specified topology with the given name and direction.

Arguments

t_name	The name of the new trunk pattern.
d_topologyId	The topology ID on which the trunk pattern is created.
t_direction	The direction of the new trunk pattern. Valid direction values are `horizontal` and `vertical`.

Value Returned

d_trunkId	The ID of the new trunk pattern.
`nil`	The trunk pattern creation failed.

Examples

```
trunk1 = dbCreateTrunk("trunk1" top1 "horizontal")
trunk2 = dbCreateTrunk("trunk2" top1 "vertical")
```


© 2020 All Rights Reserved.

dbDeleteTrunk

```
dbDeleteTrunk(
     d_trunkId
     )
     => t / nil
```

Description

Deletes the specified trunk pattern and objects associated with that pattern, such as trunk connects and twigs.

Arguments

d_trunkId	The trunk pattern ID to be deleted.

Value Returned

`t`	The trunk pattern was deleted.
`nil`	The trunk pattern deletion failed.

Example

```
dbDeleteTrunk(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkAnchor

```
dbGetTrunkAnchor(
     d_trunkId
     )
     => d_anchorId / nil
```

Description

Returns the anchor of the specified trunk pattern. The anchor may be an instance or another trunk.

Arguments

d_trunkId	The trunk pattern ID whose anchor must be returned.

Value Returned

d_anchorId	The group type associated with the specified topology pattern ID.
nil	The anchor is not set for the specified topology pattern ID.

Example

```
dbGetTrunkAnchor(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkDirection

```
dbGetTrunkDirection(
     d_trunkId
     )
     => t_direction / nil
```

Description

Returns the direction of the specified trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose direction must be returned.

Value Returned

t_direction	The direction of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkDirection(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkLongOffset1

```
dbGetTrunkLongOffset1(
     d_trunkId
     )
     => f_longitudinalOffset1 / nil
```

Description

Returns the longitudinal offset1 value of the specified trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose longitudinal offset1 value must be returned.

Value Returned

f_longitudinalOffset1	The longitudinal offset1 value of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkLongOffset1(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkLongOffset2

```
dbGetTrunkLongOffset2(
     d_trunkId
     )
     => f_longitudinalOffset2 / nil
```

Description

Returns the longitudinal offset2 value of the specified trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose longitudinal offset2 value must be returned.

Value Returned

f_longitudinalOffset2	The longitudinal offset2 value of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkLongOffset2(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkLongOffsetSource

```
dbGetTrunkLongOffsetSource(
     d_trunkId
     )
     => t_source / nil
```

Description

Returns the longitudinal offset source value of the specified trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose longitudinal offset source must be returned.

Value Returned

t_source	The longitudinal offset source of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkLongOffsetSource(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkLongOffsetSource1

```
dbGetTrunkLongOffsetSource1(
     d_trunkId
     )
     => t_source / nil
```

Description

Returns the first longitudinal offset source value of the specified trunk pattern ID. Valid source values are `unknown`, `twigExtents`, `figGroupExtents`, `instanceExtents`, `guardRing`, and `connExtents`.

Arguments

d_trunkId	The trunk pattern ID of which the first longitudinal offset source must be returned.

Value Returned

t_source	The first longitudinal offset source of the specified trunk pattern ID.
`nil`	The specified trunk pattern ID is incorrect.

Example

```
dbGetTrunkLongOffsetSource1(trunk1)
=> "twigExtents"
```


© 2020 All Rights Reserved.

dbGetTrunkLongOffsetSource2

```
dbGetTrunkLongOffsetSource2(
     d_trunkId
     )
     => t_source / nil
```

Description

Returns the second longitudinal offset source value of the specified trunk pattern ID. Valid source values are `unknown`, `twigExtents`, `figGroupExtents`, `instanceExtents`, `guardRing`, and `connExtents`.

Arguments

d_trunkId	The trunk pattern ID of which the second longitudinal offset source must be returned.

Value Returned

t_source	The second longitudinal offset source of the specified trunk pattern ID.
`nil`	The specified trunk pattern ID is incorrect.

Example

```
dbGetTrunkLongOffsetSource2(trunk1)
=> "instanceExtents"
```


© 2020 All Rights Reserved.

dbGetTrunkName

```
dbGetTrunkName(
     d_trunkId
     )
     => t_name / nil
```

Description

Returns the name of the trunk specified by the trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose name must be returned.

Value Returned

t_name	The name of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkName(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkOrthoOffset

```
dbGetTrunkOrthoOffset(
     d_trunkId
     )
     => n_offset / nil
```

Description

Returns the orthogonal offset of the specified trunk pattern ID. The value returned is in userUnits. An orthogonal offset is meaningful only when there is an anchor set for the trunk pattern.

Arguments

d_trunkId	The trunk pattern ID whose orthogonal offset name must be returned.

Value Returned

n_offset	The orthogonal offset of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkOrthoOffset(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkSide

```
dbGetTrunkSide(
     d_trunkId
     )
     => t_side / nil
```

Description

Returns the side of the trunk specified by the trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose side must be returned.

Value Returned

t_side	The side of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkSide(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkTopology

```
dbGetTrunkTopology(
     d_trunkId
     )
     => d_topologyId / nil
```

Description

Returns the topology pattern that owns the specified trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose owner topology pattern must be returned.

Value Returned

d_topologyId	The topology pattern that owns the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkTopology(trunk1)
```


© 2020 All Rights Reserved.

dbGetTrunkTwigs

```
dbGetTrunkTwigs(
     d_trunkId
     )
     => l_twigs / nil
```

Description

Returns the list of twigs associated with the specified trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose list of twigs must be returned.

Value Returned

l_twigs	The list of twigs of the specified trunk pattern ID.
nil	The trunk pattern ID is incorrect.

Example

```
dbGetTrunkTwigs(trunk1)
```


© 2020 All Rights Reserved.

dbSetTrunkAnchor

```
dbSetTrunkAnchor(
     d_trunkId
     d_anchorId
     )
     => t / nil
```

Description

Sets the anchor for the specified trunk pattern. The anchor may be an instance or another trunk.

Arguments

d_trunkId	The trunk pattern ID whose anchor must be set.
d_anchorId	The anchor ID to be associated with the specified trunk pattern ID.

Value Returned

`t`	The anchor was set for the specified trunk pattern ID.
`nil`	The anchor was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkAnchor(trunk1 trunk2)
```


© 2020 All Rights Reserved.

dbSetTrunkLongOffset1

```
dbSetTrunkLongOffset1(
     d_trunkId
     f_longitudinalOffset1
     )
     => t / nil
```

Description

Sets the longitudinal offset1 value for the specified trunk pattern.

Arguments

d_trunkId	The trunk pattern ID whose longitudinal offset1 value must be set.
f_longitudinalOffset1	The longitudinal offset1 value that must be set for the specified trunk pattern ID.

Value Returned

`t`	The longitudinal offset1 value was set for the specified trunk pattern ID.
`nil`	The longitudinal offset1 value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkLongOffset1(trunk1 4.0)
```


© 2020 All Rights Reserved.

dbSetTrunkLongOffset2

```
dbSetTrunkLongOffset2(
     d_trunkId
     f_longitudinalOffset2
     )
     => t / nil
```

Description

Sets the longitudinal offset2 value for the specified trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose longitudinal offset2 value must be set.
f_longitudinalOffset2	The longitudinal offset2 value that must be set for the specified trunk pattern ID.

Value Returned

`t`	The longitudinal offset2 value was set for the specified trunk pattern ID.
`nil`	The longitudinal offset2 value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkLongOffset2(trunk1 8.1)
```


© 2020 All Rights Reserved.

dbSetTrunkLongOffsetSource

```
dbSetTrunkLongOffsetSource(
     d_trunkId
     t_source
     )
     => t / nil
```

Description

Sets the longitudinal offset source value of the specified trunk pattern ID. Valid source values are `twigExtents`, `figGroupExtents`, `twigExtentsTermOnly`, `guardRing`.

Arguments

d_trunkId	The trunk pattern ID whose longitudinal offset source value must be set.
t_source	The longitudinal offset source value that must be set for the specified trunk pattern ID.

Value Returned

`t`	The longitudinal offset source value was set for the specified trunk pattern ID.
`nil`	The longitudinal offset source value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkLongOffsetSource(trunk1 "twigExtents")
```


© 2020 All Rights Reserved.

dbSetTrunkLongOffsetSource1

```
dbSetTrunkLongOffsetSource1(
     d_trunkId
     t_source
     )
     => t / nil
```

Description

Sets the first longitudinal offset source value of the specified trunk pattern ID. Valid source values are `twigExtents`, `figGroupExtents`, `instanceExtents`, `guardRing`, and `connExtents`.

Arguments

d_trunkId	The trunk pattern ID for which the first longitudinal offset source value must be set.
t_source	The first longitudinal offset source value that must be set for the specified trunk pattern ID.

Value Returned

`t`	The first longitudinal offset source value was set for the specified trunk pattern ID.
`nil`	The first longitudinal offset source value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkLongOffsetSource1(trunk "twigExtents")
=> t
```


© 2020 All Rights Reserved.

dbSetTrunkLongOffsetSource2

```
dbSetTrunkLongOffsetSource2(
     d_trunkId
     t_source
     )
     => t / nil
```

Description

Sets the second longitudinal offset source value of the specified trunk pattern ID. Valid source values are `twigExtents`, `figGroupExtents`, `instanceExtents`, `guardRing`, and `connExtents`.

Arguments

d_trunkId	The trunk pattern ID for which the second longitudinal offset source value must be set.
t_source	The second longitudinal offset source value that must be set for the specified trunk pattern ID.

Value Returned

`t`	The second longitudinal offset source value was set for the specified trunk pattern ID.
`nil`	The second longitudinal offset source value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkLongOffsetSource2(trunk "twigExtents")
=> t
```


© 2020 All Rights Reserved.

dbSetTrunkName

```
dbSetTrunkName(
     d_trunkId
     t_name
     )
     => t / nil
```

Description

Sets the name of the trunk specified by the trunk pattern ID.

Arguments

d_trunkId	The trunk pattern ID whose name must be set.
t_name	The new name of the trunk pattern ID.

Value Returned

`t`	The new name of the trunk pattern ID was set.
`nil`	The new name of the trunk pattern ID was not set.

Example

```
dbSetTrunkName(trunk1 "tk1")
```


© 2020 All Rights Reserved.

dbSetTrunkDirection

```
dbSetTrunkName(
     d_trunkId
     t_direction
     )
     => t / nil
```

Description

Specifies whether the given trunk is running in the horizontal or vertical direction.

Arguments

d_trunkId	The trunk pattern ID whose direction must be set.
t_direction	The horizontal or vertical direction that must be set.

Value Returned

`t`	The direction of the trunk ID was set.
`nil`	The direction of the trunk ID was not set.

Example

```
dbSetTrunkDirection(trunkId "horizontal")
dbSetTrunkDirection(trunkId "vertical")
```


© 2020 All Rights Reserved.

dbSetTrunkOrthoOffset

```
dbSetTrunkOrthoOffset(
     d_trunkId
     n_offset
     )
     => t / nil
```

Description

Sets the orthogonal offset value for the specified trunk pattern ID. The specified value must be in userUnits. An orthogonal offset is only meaningful when there is an anchor set for the trunk pattern.

Arguments

d_trunkId	The trunk pattern ID whose orthogonal offset value must be set.
n_offset	The orthogonal offset value that must be set for the specified trunk pattern ID.

Value Returned

`t`	The orthogonal offset value was set for the specified trunk pattern ID.
`nil`	The orthogonal offset value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkOrthoOffset(trunk1 2)
```


© 2020 All Rights Reserved.

dbSetTrunkSide

```
dbSetTrunkSide(
     d_trunkId
     t_side
     )
     => t / nil
```

Description

Sets the side value for the specified trunk pattern ID. The valid side values are `bottom`, `top`, `left`, and `right`.

Arguments

d_trunkId	The trunk pattern ID whose side value must be set.
t_side	The side value that must be set for the specified trunk pattern ID.

Value Returned

`t`	The side value was set for the specified trunk pattern ID.
`nil`	The side value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkSide(trunk1 "top")
```


© 2020 All Rights Reserved.

dbCreateTwig

```
dbCreateTwig(
     t_name
     d_trunkId
     )
     => d_twigId / nil
```

Description

Creates a twig pattern on the specified trunk.

Arguments

t_name	The name of the new twig pattern.
d_trunkId	The trunk ID on which the twig pattern is created.

Value Returned

d_twigId	The ID of the new twig pattern.
nil	The twig pattern creation failed.

Example

```
twig = dbCreateTwig("twig" trunk)
```


© 2020 All Rights Reserved.

dbDeleteTwig

```
dbDeleteTwig(
     d_twigId
     )
     => t / nil
```

Description

Deletes the twig pattern.

Arguments

d_twigId	The twig pattern ID to be deleted.

Value Returned

`t`	The twig pattern was deleted.
`nil`	The twig pattern deletion failed.

Example

```
dbDeleteTwig(twig)
```


© 2020 All Rights Reserved.

dbGetTwigName

```
dbGetTwigName(
     d_twigId
     )
     => t_name / nil
```

Description

Returns the name of the specified twig pattern ID.

Arguments

d_twigId	The twig pattern ID whose name must be returned.

Value Returned

t_name	The name of the specified twig ID.
nil	The twig ID is incorrect.

Example

```
dbGetTwigName(twig)
```


© 2020 All Rights Reserved.

dbGetTwigObject

```
dbGetTwigObject(
     d_twigId
     )
     => d_dbId / nil
```

Description

Returns the database object set for the specified twig ID. A twig can have only one object and that is one of the following: InstTerm, Via, MPPShape, or a Strap.

Arguments

d_twigId	The twig pattern ID whose database object set must be returned.

Value Returned

d_dbId	The database object set of the specified twig ID.
nil	The twig ID is incorrect.

Example

```
dbGetTwigObject(twig)
```


© 2020 All Rights Reserved.

dbGetTwigPinName

```
dbGetTwigPinName(
     d_twigId
     )
     => t_pinName / nil
```

Description

Returns the pin name set on the specified twig pattern ID. If there is no pin name for the twig pattern ID, an empty string is returned.

Arguments

d_twigId	The twig pattern ID whose pin name set must be returned.

Value Returned

t_pinName	The pin name set of the specified twig pattern ID.
nil	The twig ID is incorrect.

Example

```
dbGetTwigPinName(twig)
```


© 2020 All Rights Reserved.

dbGetShapeTwigs

```
dbGetShapeTwigs(
     d_shapeId
     )
     => l_twigs / nil
```

Description

Returns a list of twigs associated with the given shape ID.

Arguments

d_shapeId	The ID of the shape to query.

Value Returned

l_twigs	The list of twigs associated with the given shape.
nil	The shape is not associated with any twigs.

Example

```
dbGetShapeTwigs(shapeId)
```


© 2020 All Rights Reserved.

dbGetInstTermTwigs

```
dbGetInstTermTwigs(
     d_instTermId
     )
     => l_twigs / nil
```

Description

Returns a list of twigs associated with the given instTerm ID.

Arguments

d_instTermId	The ID of the instTerm to query.

Value Returned

l_twigs	The list of twigs associated with the given instTerm.
nil	The instTerm is not associated with any twigs.

Example

```
dbGetInstTermTwigs(instTermId)
```


© 2020 All Rights Reserved.

dbGetViaTwigs

```
dbGetViaTwigs(
     d_viaId
     )
     => l_twigs / nil
```

Description

Returns a list of twigs associated with the given via.

Arguments

d_viaId	The ID of the via to query.

Value Returned

l_twigs	The list of twigs associated with the given via.
nil	The via is not associated with any twigs.

Example

```
dbGetViaTwigs(viaId)
```


© 2020 All Rights Reserved.

dbGetTwigTrunk

```
dbGetTwigTrunk(
     d_twigId
     )
     => d_trunkId / nil
```

Description

Returns the trunk that owns the specified twig pattern ID.

Arguments

d_twigId	The twig pattern ID whose trunk must be returned.

Value Returned

d_trunkId	The trunk ID of the specified twig pattern ID.
`nil`	The twig ID is incorrect.

Example

```
dbGetTwigTrunk(twig)
```


© 2020 All Rights Reserved.

dbSetTwigName

```
dbSetTwigName(
     d_twigId
     t_name
     )
     => t / nil
```

Description

Sets the name of the specified twig pattern ID.

Arguments

d_twigId	The twig pattern ID whose name must be set.
t_name	The new name of the specified twig pattern ID.

Value Returned

`t`	The new name was set for the specified twig pattern ID.
`nil`	The new name was not set for the specified twig pattern ID.

Example

```
dbSetTwigName(twig "newTwig")
```


© 2020 All Rights Reserved.

dbSetTwigObject

```
dbSetTwigObject(
     d_twigId
     d_dbId
     )
     => t / nil
```

Description

Sets the database object for the specified twig ID. A twig can have only one object and that is one of the following: InstTerm, Via, MPPShape, or a Strap.

Arguments

d_twigId	The twig ID whose database object must be set.
d_dbId	The database object ID that must be set for the specified twig ID.

Value Returned

`t`	The database object was set for the specified twig ID.
`nil`	The database object was not set for the specified twig ID or the specified database ID is not an InstTerm, Via, MPP Shape, or a Strap.

Example

```
dbSetTwigObject(twig via)
dbSetTwigObject(twig instTerm)
```


© 2020 All Rights Reserved.

dbSetTwigPinName

```
dbSetTwigPinName(
     d_twigId
     t_pinName
     )
     => t / nil
```

Description

Sets the pin name set for the specified twig.

Arguments

d_twigId	The twig ID whose pin name must be set.
t_pinName	The pin name that must be set for the specified twig ID.

Value Returned

`t`	The pin name was set for the specified twig ID.
`nil`	The pin name was not set for the specified twig ID.

Example

```
dbSetTwigPinName(twig "pinN")
```


© 2020 All Rights Reserved.

dbCreateTrunkConnect

```
dbCreateTrunkConnect(
     t_name
     d_topologyId
     d_startTrunkId
     d_endTrunkId
     )
     => d_trunkConnectId / nil
```

Description

Creates a trunk connect pattern on the specified topology with the specified start and end trunks.

Arguments

t_name	The name of the trunk connect pattern.
d_topologyId	The topology ID on which the trunk connect pattern is created.
d_startTrunkId	The start trunk ID.
d_endTrunkId	The end trunk ID.

Value Returned

d_trunkConnectId	The ID of the new trunk connect pattern.
nil	The trunk connect pattern creation failed.

Example

```
trunkConnect = dbCreateTrunkConnect("trunkConnect" topology startTrunk endTrunk)
```


© 2020 All Rights Reserved.

dbDeleteTrunkConnect

```
dbDeleteTrunkConnect(
     d_trunkConnectId
     )
     => t / nil
```

Description

Deletes the specified trunk connect pattern, which includes removing the association the pattern has with the topology, and deleting associated trunks.

Arguments

d_trunkConnectId	The trunk connect pattern ID to be deleted.

Value Returned

`t`	The trunk connect pattern was deleted.
`nil`	The trunk connect pattern deletion failed.

Example

```
dbDeleteTrunkConnect(trunkConnect)
```


© 2020 All Rights Reserved.

dbGetTrunkConnectName

```
dbGetTrunkConnectName(
     d_trunkConnectId
     )
     => t_name / nil
```

Description

Returns the name of the specified trunk connect pattern ID.

Arguments

d_trunkConnectId	The trunk connect pattern ID whose name must be returned.

Value Returned

t_name	The name of the specified trunk connect pattern ID.
nil	The trunk connect pattern ID is incorrect.

Example

```
dbGetTrunkConnectName(trunkConnect)
```


© 2020 All Rights Reserved.

dbGetTrunkConnectTopology

```
dbGetTrunkConnectTopology(
     d_trunkConnectId
     )
     => d_topologyId / nil
```

Description

Returns the owner topology of the specified trunk connect pattern ID.

Arguments

d_trunkConnectId	The trunk connect pattern ID whose owner topology must be returned.

Value Returned

d_topologyId	The owner topology of the specified trunk connect pattern ID.
nil	The trunk connect pattern ID is incorrect.

Example

```
dbGetTrunkConnectTopology(trunkConnect)
```


© 2020 All Rights Reserved.

dbGetTrunkConnectTrunks

```
dbGetTrunkConnectTrunks(
     d_trunkConnectId
     )
     => l_trunkIds / nil
```

Description

Returns the list of trunks associated with the specified trunk connect ID. The returned list includes the start and end trunks.

Arguments

d_trunkConnectId	The trunk connect pattern ID whose list of associated trunks must be returned.

Value Returned

l_trunkIds	The list of start and end trunks of the specified trunk connect pattern ID.
nil	The trunk connect pattern ID is incorrect.

Example

```
dbGetTrunkConnectTrunks(tc)
```


© 2020 All Rights Reserved.

dbSetTrunkConnectName

```
dbSetTrunkConnectName(
     d_trunkConnectId
     t_name
     )
     => t / nil
```

Description

Sets the name of the specified trunk connect pattern.

Arguments

d_trunkConnectId	The trunk connect pattern ID whose name must be set.
t_name	The new name of the specified trunk connect pattern ID.

Value Returned

`t`	The new name of the trunk connect pattern ID was set.
`nil`	The new name of the trunk connect pattern ID was not set.

Example

```
dbSetTrunkConnectName(trunkConnect "newTrunkConnect")
```


© 2020 All Rights Reserved.

dbSetTrunkConnectTrunks

```
dbSetTrunkConnectTrunks(
     d_trunkConnectId
     d_startTrunkId
     d_endTrunkId
     )
     => t / nil
```

Description

Sets the start and end trunks for the specified trunk connect ID.

Arguments

d_trunkConnectId	The trunk connect ID whose start and end trunks must be set.
d_startTrunkId	The start trunk for the specified trunk connect ID.
d_endTrunkId	The end trunk for the specified trunk connect ID.

Value Returned

`t`	The start and end trunks were set for the specified trunk connect ID.
`nil`	The start and end trunks were not set for the specified trunk connect ID.

Example

```
dbSetTrunkConnectTrunks(trunkConnect endTrunk startTrunk)
```


© 2020 All Rights Reserved.

dbCreateStrap

```
dbCreateStrap(
     t_name
     d_topologyId
     )
     => d_strapId / nil
```

Description

Creates a strap pattern with the given name on the specified topology.

Arguments

t_name	The name of the new strap pattern.
d_topologyId	The topology ID to which the strap pattern must belong.

Value Returned

d_strapId	The ID of the new strap pattern.
nil	The strap pattern creation failed.

Example

```
dbCreateStrap("strap1" top)
```


© 2020 All Rights Reserved.

dbDeleteStrap

```
dbDeleteStrap(
     d_strapId
     )
     => t / nil
```

Description

Deletes the specified strap pattern.

Arguments

d_strapId	The strap ID to be deleted.

Value Returned

`t`	The strap was deleted.
`nil`	The strap deletion failed.

Example

```
dbDeleteStrap(strap)
```


© 2020 All Rights Reserved.

dbGetStrapName

```
dbGetStrapName(
     d_strapId
     )
     => t_name / nil
```

Description

Returns the name of the specified strap pattern ID.

Arguments

d_strapId	The strap pattern ID whose name must be returned.

Value Returned

t_name	The name associated with the strap pattern ID.
nil	The specified strap pattern ID is incorrect.

Example

```
dbGetStrapName(strap)
```


© 2020 All Rights Reserved.

dbGetStrapObjects

```
dbGetStrapObjects(
     d_strapId
     )
     => l_objects / nil
```

Description

Returns a list of database objects associated with the specified strap pattern ID.

Arguments

d_strapId	The strap pattern ID whose list of associated database objects must be returned.

Value Returned

l_objects	The list of database objects associated with the specified strap pattern ID.
nil	The specified strap pattern ID is incorrect.

Example

```
dbGetStrapObjects(strap)
```


© 2020 All Rights Reserved.

dbGetStrapTopology

```
dbGetStrapTopology(
     d_strapId
     )
     => d_topologyId / nil
```

Description

Returns the topology pattern ID that owns the specified strap pattern.

Arguments

d_strapId	The strap pattern ID whose topology pattern ID must be returned.

Value Returned

d_topologyId	The topology pattern ID of the specified strap pattern ID.
nil	The specified strap pattern ID is incorrect.

Example

```
dbGetStrapTopology(strap)
```


© 2020 All Rights Reserved.

dbGetStrapTwig

```
dbGetStrapTwig(
     d_strapId
     )
     => t_name / nil
```

Description

Returns the twig associated with the specified strap pattern ID.

Arguments

d_strapId	The strap pattern ID whose twig name must be returned.

Value Returned

t_name	The name of the twig associated with the specified strap pattern ID.
nil	The strap pattern ID does not have an associated twig.

Example

```
dbGetStrapTwig(strap)
```


© 2020 All Rights Reserved.

dbGetInstTermStraps

```
dbGetInstTermStraps(
     d_instTermId
     )
     => l_straps / nil
```

Description

Returns a list of straps associated with the given instTerm ID.

Arguments

d_instTermId	The ID of the instTerm to query.

Value Returned

l_straps	The list of straps associated with the given instTerm.
nil	The instTerm is not associated with any straps.

Example

```
dbGetInstTermStraps(instTermId)
```


© 2020 All Rights Reserved.

dbGetViaStraps

```
dbGetViaStraps(
     d_viaId
     )
     => l_straps / nil
```

Description

Returns a list of straps associated with the given via ID.

Arguments

d_viaId	The ID of the via to query.

Value Returned

l_straps	The list of straps associated with the given via.
nil	The via is not associated with any straps.

Example

```
dbGetViaStraps(viaId)
```


© 2020 All Rights Reserved.

dbGetShapeStraps

```
dbGetShapeStraps(
     d_shapeId
     )
     => l_straps / nil
```

Description

Returns a list of straps associated with the given shape ID.

Arguments

d_shapeId	The ID of the shape to query.

Value Returned

l_straps	The list of straps associated with the given shape.
nil	The given shape is not associated with any straps.

Example

```
dbGetShapeStraps(shapeId)
```


© 2020 All Rights Reserved.

dbAddStrapObject

```
dbAddStrapObject(
     d_strapId
     d_dbId
     )
     => t / nil
```

Description

Adds a database object to a specified strap pattern. The object must be of the type InstTerm, Via, or Shape. The object must be in the same database as the strap.

Arguments

d_strapId	The strap pattern ID to which the database object will be added.
d_dbId	The database object ID to be added to the strap pattern.

Value Returned

`t`	The database object was added.
`nil`	The database object addition failed.

Examples

```
dbAddStrapObject(strap instTerm)
dbAddStrapObject(strap rect1)
dbAddStrapObject(strap via)
```


© 2020 All Rights Reserved.

dbRemoveStrapObject

```
dbRemoveStrapObject(
     d_strapId
     d_dbId
     )
     => t / nil
```

Description

Removes the database object from the specified strap pattern.

Arguments

d_strapId	The strap pattern ID from where the specified database object must be removed.
d_dbId	The database object ID that must be removed from the specified strap pattern ID.

Value Returned

`t`	The database object ID was removed from the specified strap pattern ID.
`nil`	The database object ID was not removed from the specified strap pattern ID.

Example

```
dbRemoveStrapObject(strap rect1)
```


© 2020 All Rights Reserved.

dbSetStrapName

```
dbSetStrapName(
     d_strapId
     t_name
     )
     => t / nil
```

Description

Sets the name of the specified strap pattern ID.

Arguments

d_strapId	The strap pattern ID whose name must be set.
t_name	The new name of the specified strap pattern ID.

Value Returned

`t`	The name is set for the specified strap pattern ID.
`nil`	The name was not set for the specified strap pattern ID.

Example

```
dbSetStrapName(strap "strap2")
```


© 2020 All Rights Reserved.

dbSetStrapTwig

```
dbSetStrapTwig(
     d_strapId
     t_name
     )
     => t / nil
```

Description

Changes the twig associated with the specified strap pattern.

Arguments

d_strapId	The strap pattern ID whose twig association must be changed.
t_name	The name of the twig.

Value Returned

`t`	The twig association was changed with the specified strap pattern ID.
`nil`	The twig association was not changed with the specified strap pattern ID.

Example

```
dbSetStrapTwig(strap twig1)
```


© 2020 All Rights Reserved.

dbIsTopologyPatternId

```
dbIsTopologyPatternId(
     d_objectId
     )
     => t / nil
```

Description

Returns a Boolean indicating whether the specified object ID is a `dbTopologyPattern`.

Arguments

d_objectId	The object ID whose `dbTopologyPattern` status must be returned.

Value Returned

`t`	The specified object ID is a `dbTopologyPattern`.
`nil`	The specified object ID is not a `dbTopologyPattern`.

Example

```
dbIsTopologyPatternId(top)
t
dbIsTopologyPatternId(trunk)
t
dbIsTopologyPatternId(twig)
t
dbIsTopologyPatternId(net)
nil
```


© 2020 All Rights Reserved.

dbNetHasTopology

```
dbNetHasTopology(
     d_netId
     )
     => t / nil
```

Description

Returns a Boolean indicating whether the specified net has a topology.

Arguments

d_netId	The net ID whose topology association status must be returned.

Value Returned

`t`	The specified net ID has a topology.
`nil`	The specified net ID does not have a topology.

Example

```
dbNetHasTopology(netA)
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

9

Generator Database Access Functions

This chapter describes the following generator database SKILL functions:

- dbStartGenAnyInstToInstTerm
- dbStartGenInstHeader
- dbStartGenInstHeaderToAnyInst
- dbStartGenLPPHeaderToShape
- dbStartGenLPToShape
- dbStartGenMarker
- dbStartGenNet
- dbStartGenNetToInstTerm
- dbStartGenNetToRoute
- dbStartGenRoute
- dbStartGenShape
- dbStartGenTerm
- dbStartGenViaHeader
- dbStartGenViaHeaderToVia
- dbStopGen

© 2020 All Rights Reserved.

dbStartGenAnyInstToInstTerm

```
dbStartGenAnyInstToInstTerm(
     d_instId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all instance terminals for the specified instance.

Arguments

d_instId	Database ID of an instance.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenAnyInstToInstTerm(inst1)
while(instTermId = gen1~>next
;Use instTermId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenInstHeader

```
dbStartGenInstHeader(
     d_cellViewId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all instance headers of the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenInstHeader(cellview1)
while(instHeaderId = gen1~>next
;Use instHeaderId in the code.
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenInstHeaderToAnyInst

```
dbStartGenInstHeaderToAnyInst(
     d_instHeaderId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all instances of the specified instance header.

Arguments

d_instHeaderId	Database ID of an instance header.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenInstHeaderToAnyInst(instHeader1)
while(instId = gen1~>next
;Use instId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenLPPHeaderToShape

```
dbStartGenLPPHeaderToShape(
     d_LPPHeaderId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all shapes of the specified cellview and layer-purpose pair (LPP) header.

Arguments

d_LPPHeaderId	Database ID of an LPP header ID.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
lpp1 = car(cellview1~>lpps)
gen1 = dbStartGenLPPHeaderToShape(lpp1)
while(shapeId = gen1~>next
;Use shapeId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenLPToShape

```
dbStartGenLPToShape(
     d_cellViewId
     l_layerPurpose
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all shapes of the specified cellview and layer-purpose pair (LPP).

Arguments

d_cellViewId	Database ID of a cellview.
l_layerPurpose	Specifies the of layer-purpose pair.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
lpp1 = car(cellview~>shapes~>lpp)
gen1 = dbStartGenLPToShape(lpp1)
while(shapeId = gen1~>next
;Use shapeId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenMarker

```
dbStartGenMarker(
     d_cellViewId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all markers of the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenMarker(cellview1)
while(markerId = gen1~>next
;Use markerId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenNet

```
dbStartGenNet(
     d_cellViewId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all nets of the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenNet(cellview1)
while(netId = gen1~>next
;Use netId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenNetToInstTerm

```
dbStartGenNetToInstTerm(
     d_netId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all instance terminals of the specified net.

Arguments

d_netId	Database ID of a net.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenNetToInstTerm(net1)
while(instTermId = gen1~>next
;Use instTermId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenNetToRoute

```
dbStartGenNetToRoute(
     d_netId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all routes of the specified net.

Arguments

d_netId	Database ID of a net

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenNetToRoute(net1)
while(routeId = gen1~>next
;Use routeId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenRoute

```
dbStartGenRoute(
     d_cellViewId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all routes of the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenRoute(cellview1)
while(routeId = gen1~>next
;Use routeId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenShape

```
dbStartGenShape(
     d_cellViewId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all shapes of the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenShape(cellview1)
while(shapeId = gen1~>next
;Use shapeId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenTerm

```
dbStartGenTerm(
     d_cellViewId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all terms of the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenTerm(cellview1)
while(termId = gen1~>next
;Use termId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenViaHeader

```
dbStartGenViaHeader(
     d_cellViewId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all via headers of the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenViaHeader(cellview1)
while(viaHeaderId = gen1~>next
;Use viaHeaderId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStartGenViaHeaderToVia

```
dbStartGenViaHeaderToVia(
     d_viaHeaderId
     )
     => I_genId
```

Description

Returns a dbGenerator object that can be used to iterate over all via of the specified via header.

Arguments

d_viaHeaderId	Database ID of a via header.

Value Returned

I_genId	ID of a dbGenerator object.

Example

```
gen1 = dbStartGenViaHeaderToVia(viaHeader1)
while(viaId = gen1~>next
;Use viaId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

dbStopGen

```
dbStopGen(
     I_genId
     )
     => t / nil
```

Description

Forces a dbGenerator object reference to close and removes it from virtual memory.

Arguments

I_genId	Specifies ID of a dbGenerator object.

Value Returned

t	dbGenerator is closed.
nil	dbGenerator is not closed.

Example

```
gen1 = dbStartGenNet(cellview1)
while(netId = gen1~>next
;Use netId in the code
)

; Close dbGenerator object reference and remove it from virtual memory
dbStopGen(gen1)
```


© 2020 All Rights Reserved.

10

InPlace Cover Obstruction Database Access Functions

The InPlaceCoverObstruction SKILL functions are used to create or destroy InPlaceCoverObstruction objects and to retrieve or set InPlaceCoverObstruction attributes. This chapter describes the following in place cover obstruction database SKILL functions:

- dbCreateInPlaceCoverObstruction
- dbDestroyInPlaceCoverObstruction
- dbGetInPlaceCoverObstructionAbstractFileName
- dbGetInPlaceCoverObstructionAbstractFilePath
- dbGetInPlaceCoverObstructionAbstractViewName
- dbGetInPlaceCoverObstructionBloat
- dbGetInPlaceCoverObstructionBlockageAttributeValue
- dbGetInPlaceCoverObstructionBlockageModel
- dbGetInPlaceCoverObstructionDoughnutHalo
- dbGetInPlaceCoverObstructionLayers
- dbGetInPlaceCoverObstructionMaxMask
- dbGetInPlaceCoverObstructionMergeDistance
- dbGetInPlaceCoverObstructionMinMask
- dbGetInPlaceCoverObstructionNeedsRemodeling
- dbGetInPlaceCoverObstructionPinCutOut
- dbGetInPlaceCoverObstructionPinRemodeling
- dbGetInPlaceCoverObstructionSpacingModel

© 2020 All Rights Reserved.

- dbInPlaceCoverObstructionExists
- dbSetInPlaceCoverObstructionAbstractFileName
- dbSetInPlaceCoverObstructionAbstractFilePath
- dbSetInPlaceCoverObstructionAbstractViewName
- dbSetInPlaceCoverObstructionBloat
- dbSetInPlaceCoverObstructionBlockageAttributeValue
- dbSetInPlaceCoverObstructionBlockageModel
- dbSetInPlaceCoverObstructionDoughnutHalo
- dbSetInPlaceCoverObstructionMergeDistance
- dbSetInPlaceCoverObstructionNeedsRemodeling
- dbSetInPlaceCoverObstructionPinCutOut
- dbSetInPlaceCoverObstructionPinRemodeling
- dbSetInPlaceCoverObstructionSpacingModel

© 2020 All Rights Reserved.

dbCreateInPlaceCoverObstruction

```
dbCreateInPlaceCoverObstruction(
     d_InstHeaderId
     l_intRange
     )
     => t/nil
```

Description

Creates an InPlaceCoverObstruction object on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.
l_intRange	Integer mask range (`minMask`, `maxMask`) for the InPlaceCoverObstruction to be created on the instance header.

Value Returned

`t`	An InPlaceCoverObstruction object is created on the specified instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
```

The above function is passed an instance header ID along with a closed range of integers with minimum mask value of `1`, and maximum mask value of `4`, both values inclusive.

© 2020 All Rights Reserved.

dbDestroyInPlaceCoverObstruction

```
dbDestroyInPlaceCoverObstruction(
     d_InstHeaderId
     )
     => t / nil
```

Description

Destroys an InPlaceCoverObstruction object on the given instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

`t`	The InPlaceCoverObstruction object on the given instance header is destroyed.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     dbDestroyInPlaceCoverObstruction(instHeaderId)
```

Destroys the InPlaceCoverObstruction object on the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionAbstractFileName

```
dbGetInPlaceCoverObstructionAbstractFileName(
     d_InstHeaderId
     )
     => t_fileName / nil
```

Description

Returns value of the InPlaceCoverObstruction AbstractFileName attribute for the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

t_fileName	Valid value of the InPlaceCoverObstruction AbstractFileName attribute.
nil	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionAbstractFileName(instHeaderId) "")
         dbSetInPlaceCoverObstructionAbstractFileName(instHeaderId "newFileName")
         equal(dbGetInPlaceCoverObstructionAbstractFileName(instHeaderId)
"newFileName")
```

Returns `newFileName` as the InPlaceCoverObstruction AbstractFileName attribute value for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionAbstractFilePath

```
dbGetInPlaceCoverObstructionAbstractFilePath(
     d_InstHeaderId
     )
     => t_filePath / nil
```

Description

Returns the value of the InPlaceCoverObstruction AbstractFilePath attribute for the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

t_filePath	Value of the InPlaceCoverObstruction AbstractFilePath attribute for the specified instance header.
nil	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionAbstractFilePath(instHeaderId) "")
         dbSetInPlaceCoverObstructionAbstractFilePath(instHeaderId "newFilePath")
         equal(dbGetInPlaceCoverObstructionAbstractFilePath(instHeaderId)
"newFilePath")
```

Returns `newFilePath` as the InPlaceCoverObstruction AbstractFilePath attribute value for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionAbstractViewName

```
dbGetInPlaceCoverObstructionAbstractViewName(
     d_InstHeaderId
     )
     => t_viewName / nil
```

Description

Returns value of the InPlaceCoverObstruction AbstractViewName attribute for the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

t_viewName	Value of the AbstractViewName attribute.
nil	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionAbstractViewName(instHeaderId) "")
         dbSetInPlaceCoverObstructionAbstractViewName(instHeaderId "newViewName")
         equal(dbGetInPlaceCoverObstructionAbstractViewName(instHeaderId)
"newViewName")
```

Returns `newViewName` as the InPlaceCoverObstruction AbstractViewName attribute value for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionBloat

```
dbGetInPlaceCoverObstructionBloat(
     d_InstHeaderId
     t_layerName
     )
     => x_bloat
```

Description

Returns value of the InPlaceCoverObstruction Bloat attribute from the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.

Value Returned

x_bloat	Valid Bloat attribute values. If the operation fails, returns `0` and displays a warning message.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionBloat(instHeaderId "Layer1") 0))
     dbSetInPlaceCoverObstructionBloat(instHeaderId "Layer1" 10)
     equal(dbGetInPlaceCoverObstructionBloat(instHeaderId "Layer1") 10)
```

Returns `10` as the InPlaceCoverObstruction Bloat attribute value for the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionBlockageAttributeValue

```
dbGetInPlaceCoverObstructionBlockageAttributeValue(
     d_InstHeaderId
     t_layerName
     )
     => f_blockageAttributeValue
```

Description

Returns value of the InPlaceCoverObstruction BlockageAttributeValue attribute from the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.

Value Returned

f_blockageAttributeValue

Valid BlockageAttributeValue values.

If the operation fails, returns `0` and displays a warning message.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId
     "Layer1") 0.0))
     dbSetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId "Layer1"
     10.0)
     equal(dbGetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId
     "Layer1") 10.0)
```

Returns `10.0` as the InPlaceCoverObstruction BlockageAttributeValue on the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionBlockageModel

```
dbGetInPlaceCoverObstructionBlockageModel(
     d_InstHeaderId
     t_layerName
     )
     => t_blockageModelName/nil
```

Description

Returns value of the InPlaceCoverObstruction BlockageModel attribute from the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.

Value Returned

t_blockageModelName	Valid BlockageModel value set on the specified instance header on the given layer.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionBlockageModel(instHeaderId) "fullCover"))
     dbSetInPlaceCoverObstructionBlockageModel(instHeaderId "doughnut")
     equal(dbGetInPlaceCoverObstructionBlockageModel(instHeaderId) "doughnut")
```

Returns `"doughnut"` as the InPlaceCoverObstruction BlockageModel attribute value for the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionDoughnutHalo

```
dbGetInPlaceCoverObstructionDoughnutHalo(
     d_InstHeaderId
     t_layerName
     )
     => x_doughnutHalo
```

Description

Returns value of the InPlaceCoverObstruction DoughnutHalo attribute from the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.

Value Returned

x_doughnutHalo	Valid DoughnutHalo values. If the operation fails, returns `0` and displays a warning message.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
    instHeaderId = inst~>instHeader
    dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
    equal(dbGetInPlaceCoverObstructionDoughnutHalo(instHeaderId "Layer1") 0))
    dbSetInPlaceCoverObstructionDoughnutHalo(instHeaderId "Layer1" 10)
    equal(dbGetInPlaceCoverObstructionDoughnutHalo(instHeaderId "Layer1") 10)
```

Returns `10` as the InPlaceCoverObstruction DoughnutHalo attribute value for the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionLayers

```
dbGetInPlaceCoverObstructionLayers(
     d_InstHeaderId
     )
     => l_layerNames / nil
```

Description

Returns a list of physical layer names in the technology database, with mask value lying between the `minMask` and `maxMask` values.

Arguments

d_InstHeaderId	ID of the instance header on which InPlaceCoverObstruction object is present.

Value Returned

l_layerNames	List of valid physical layer names.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     dbGetInPlaceCoverObstructionLayers(instHeaderId)
```

Returns a list of physical layer names in the technology database for the instance header `instHeaderId`, with mask value lying between the minimum mask value of `1`, and maximum mask value of `4`, both values inclusive.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionMaxMask

```
dbGetInPlaceCoverObstructionMaxMask(
     d_InstHeaderId
     )
     => x_maxMask
```

Description

Returns value of the InPlaceCoverObstruction MaxMask attribute from the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

x_maxMask	Valid InPlaceCoverObstruction MaxMask values for the specified instance header. If the operation fails, returns `0` and displays a warning message.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     equal(dbGetInPlaceCoverObstructionMaxMask(instHeaderId) -1))
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionMaxMask(instHeaderId) 4))
```

Returns `4` as the InPlaceCoverObstruction MaxMask attribute value for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionMergeDistance

```
dbGetInPlaceCoverObstructionMergeDistance(
     d_InstHeaderId
     t_layerName
     )
     => f_mergeDistanceValue
```

Description

Returns value of the InPlaceCoverObstruction MergeDistance attribute for the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.

Value Returned

f_mergeDistanceValue

Valid MergeDistance value for the specified instance header.

If the operation fails, default value of MergeDistance is returned.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionMergeDistance(instHeaderId "Layer1")
0.0))
         dbSetInPlaceCoverObstructionMergeDistance(instHeaderId "Layer1" 10.0)
         equal(dbGetInPlaceCoverObstructionMergeDistance(instHeaderId "Layer1")
10.0)
```

Returns `10.0` as the InPlaceCoverObstruction MergeDistance attribute value for the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionMinMask

```
dbGetInPlaceCoverObstructionMinMask(
     d_InstHeaderId
     )
     => x_minMask
```

Description

Returns value of the InPlaceCoverObstruction MinMask attribute from the specifed instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

x_minMask	Valid InPlaceCoverObstruction MinMask values for the specified instance header. If the operation fails, returns `0` and displays a warning message.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     equal(dbGetInPlaceCoverObstructionMinMask(instHeaderId) -1))
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionMinMask(instHeaderId) 1))
```

Returns `1` as the InPlaceCoverObstruction MinMask attribute value for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionNeedsRemodeling

```
dbGetInPlaceCoverObstructionNeedsRemodeling(
     d_InstHeaderId
     )
     =>
     t / nil
```

Description

Returns value of the InPlaceCoverObstruction NeedsRemodeling attribute from the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

`t`	The NeedsRemodeling attribute value for the specified instance is set to `t`.
`nil`	The NeedsRemodeling attribute value for the specified instance is set to `nil`. This is also the default value of NeedsRemodeling attribute when the InPlaceCoverObstruction is created.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionNeedsRemodeling(instHeaderId) nil)
     dbSetInPlaceCoverObstructionNeedsRemodeling(instHeaderId t)
     equal(dbGetInPlaceCoverObstructionNeedsRemodeling(instHeaderId) t)
```

Returns `t` as the InPlaceCoverObstruction NeedsRemodeling attribute value for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionPinCutOut

```
dbGetInPlaceCoverObstructionPinCutOut(
     d_InstHeaderId
     t_layerName
     )
     => t / nil
```

Description

Returns value of the InPlaceCoverObstruction PinCutOut attribute for the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.

Value Returned

`t`	The PinCutOut attribute value for the specified instance is set to `t`.
`nil`	The PinCutOut attribute value for the specified instance is set to `nil`. This is also the default value of PinCutOut attribute when the InPlaceCoverObstruction is created.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionPinCutOut(instHeaderId "Layer1") nil)
         dbSetInPlaceCoverObstructionPinCutOut(instHeaderId "Layer1" t)
         equal(dbGetInPlaceCoverObstructionPinCutOut(instHeaderId "Layer1") t)
```

Returns `t` as the value to confirm that the PinCutOut attribute value for the instance header `instHeaderId` on the layer `Layer1` is set to `t`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionPinRemodeling

```
dbGetInPlaceCoverObstructionPinRemodeling(
     d_InstHeaderId
     )
     => t_remodelingName/nil
```

Description

Returns value of the InPlaceCoverObstruction PinRemodeling attribute from the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

t_remodeling	Valid PinRemodeling attribute values.
nil	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionPinRemodeling(instHeaderId) "none"))
     dbSetInPlaceCoverObstructionPinRemodeling(instHeaderId "perimeterOnly")
     equal(dbGetInPlaceCoverObstructionPinRemodeling(instHeaderId)
     "perimeterOnly")
```

Returns "`perimeterOnly`" as the InPlaceCoverObstruction PinRemodeling attribute value for the specified instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbGetInPlaceCoverObstructionSpacingModel

```
dbGetInPlaceCoverObstructionSpacingModel(
     d_InstHeaderId
     t_layerName
     )
     => t_spacingModelName / nil
```

Description

Returns value of the InPlaceCoverObstruction SpacingModel attribute from the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.

Value Returned

t_blockageModelName	Valid SpacingModel values.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
instHeaderId = inst~>instHeader
dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
      equal(dbGetInPlaceCoverObstructionSpacingModel(instHeaderId) "spacing"))
      dbSetInPlaceCoverObstructionSpacingModel(instHeaderId "asIs")

      equal(dbGetInPlaceCoverObstructionSpacingModel(instHeaderId) "asIs")
```

Returns "`asIs`" as the InPlaceCoverObstruction SpacingModel attribute value for the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbInPlaceCoverObstructionExists

```
dbInPlaceCoverObstructionExists(
     d_InstHeaderId
     )
     => t / nil
```

Description

Checks whether an InPlaceCoverObstruction object is present on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.

Value Returned

`t`	Returned in case the InPlaceCoverObstruction object is present on the specified instance header.
`nil`	Returned in case the InPlaceCoverObstruction object is not present on the specified instance header.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbInPlaceCoverObstructionExists(instHeaderId) t)
```

Returns `t` as the InPlaceCoverObstruction object is present on the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionAbstractFileName

```
dbSetInPlaceCoverObstructionAbstractFileName(
     d_InstHeaderId
     t_fileName
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction AbstractFileName attribute on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.
t_fileName	A valid AbstractFileName value.

Value Returned

`t`	The specified AbstractFileName attribute value is set for the specified instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionAbstractFileName(instHeaderId) "")
         dbSetInPlaceCoverObstructionAbstractFileName(instHeaderId "newFileName")
         equal(dbGetInPlaceCoverObstructionAbstractFileName(instHeaderId)
"newFileName")
```

Sets the InPlaceCoverObstruction AbstractFileName attribute value `newFileName` for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionAbstractFilePath

```
dbSetInPlaceCoverObstructionAbstractFilePath(
     d_InstHeaderId
     t_filePath
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction AbstractFilePath attribute on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.
t_filePath	A valid AbstractFilePath value.

Value Returned

`t`	The specified AbstractFilePath attribute value is set for the given instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionAbstractFilePath(instHeaderId) "")
         dbSetInPlaceCoverObstructionAbstractFilePath(instHeaderId "newFilePath")
         equal(dbGetInPlaceCoverObstructionAbstractFilePath(instHeaderId)
"newFilePath")
```

Sets the InPlaceCoverObstruction AbstractFilePath attribute value `newFilePath` for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionAbstractViewName

```
dbSetInPlaceCoverObstructionAbstractViewName(
     d_InstHeaderId
     t_viewName
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction AbstractViewName attribute on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.
t_viewName	A valid AbstractViewName value.

Value Returned

`t`	The specified AbstractViewName attribute value is set for the specified instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionAbstractViewName(instHeaderId) "")
         dbSetInPlaceCoverObstructionAbstractViewName(instHeaderId "newViewName")
         equal(dbGetInPlaceCoverObstructionAbstractViewName(instHeaderId)
"newViewName")
```

Sets the InPlaceCoverObstruction AbstractViewName attribute value `newViewName` for the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionBloat

```
dbSetInPlaceCoverObstructionBloat(
     d_InstHeaderId
     t_layerName
     n_value
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction Bloat attribute on the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.
n_value	A valid Bloat value.

Value Returned

`t`	The specified InPlaceCoverObstruction Bloat attribute value is set on the given layer.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionBloat(instHeaderId "Layer1") 0))
     dbSetInPlaceCoverObstructionBloat(instHeaderId "Layer1" 10)
     equal(dbGetInPlaceCoverObstructionBloat(instHeaderId "Layer1") 10)
```

Sets the InPlaceCoverObstruction Bloat attribute value `10` for the instance header `instHeaderId` on the layer `"Layer1"`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionBlockageAttributeValue

```
dbSetInPlaceCoverObstructionBlockageAttributeValue(
     d_InstHeaderId
     t_layerName
     n_value
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction BlockageAttributeValue attribute on the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.
n_value	Value of BlockageAttributeValue value that needs to be set.

Value Returned

`t`	The specified BlockageAttributeValue is set on the given instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId
"Layer1") 0.0))
     dbSetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId "Layer1"
10.0)
     equal(dbGetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId
"Layer1") 10.0)
```

Sets the InPlaceCoverObstruction BlockageAttributeValue attribute value `10.0` for the instance `instHeaderId` on the layer `"Layer1"`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionBlockageModel

```
dbSetInPlaceCoverObstructionBlockageModel(
     d_InstHeaderId
     t_layerName
     t_blockageModelName
     )
     => t/nil
```

Description

Sets value of the InPlaceCoverObstruction BlockageModel attribute on the specified instance header on the specified layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.
t_blockageModelName	Valid BlockageModel value.

Value Returned

`t`	Returned in case of success.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionBlockageModel(instHeaderId "Layer1")
     "fullCover"))
     dbSetInPlaceCoverObstructionBlockageModel(instHeaderId "Layer1" "doughnut")
     equal(dbGetInPlaceCoverObstructionBlockageModel(instHeaderId "Layer1")
     "doughnut")
```

Sets value of the InPlaceCoverObstruction BlockageModel attribute `"doughnut"` on the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionDoughnutHalo

```
dbSetInPlaceCoverObstructionDoughnutHalo(
     d_InstHeaderId
     t_layerName
     n_value
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction DoughnutHalo attribute on the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.
n_value	Valid DoughnutHalo value that needs to be set.

Value Returned

`t`	The specified InPlaceCoverObstruction DoughnutHalo attribute Value is set on the specified instance.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionDoughnutHalo(instHeaderId "Layer1") 0))
     dbSetInPlaceCoverObstructionDoughnutHalo(instHeaderId "Layer1" 10)
     equal(dbGetInPlaceCoverObstructionDoughnutHalo(instHeaderId "Layer1") 10)
```

Sets value of the InPlaceCoverObstruction DoughnutHalo attribute `10` on the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionMergeDistance

```
dbSetInPlaceCoverObstructionMergeDistance(
     d_InstHeaderId
     t_layerName
     n_value
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction MergeDistance attribute on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.
n_value	A valid MergeDistance value.

Value Returned

`t`	The specified MergeDistance attribute value is set for the specified instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionMergeDistance(instHeaderId "Layer1")
0.0))
         dbSetInPlaceCoverObstructionMergeDistance(instHeaderId "Layer1" 10.0)
         equal(dbGetInPlaceCoverObstructionMergeDistance(instHeaderId "Layer1")
10.0)
```

Sets the InPlaceCoverObstruction MergeDistance attribute value as `10.0` for the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionNeedsRemodeling

```
dbSetInPlaceCoverObstructionNeedsRemodeling(
     d_InstHeaderId
     g_value
     )
     => t/nil
```

Description

Sets value of the InPlaceCoverObstruction NeedsRemodeling attribute on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.
g_value	Valid NeedsRemodeling value.

Value Returned

`t`	The InPlaceCoverObstruction NeedsRemodeling attribute value is set on the specified instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     dbSetInPlaceCoverObstructionNeedsRemodeling(instHeaderId t)
     equal(dbGetInPlaceCoverObstructionNeedsRemodeling(instHeaderId) t)
```

Sets InPlaceCoverObstruction NeedsRemodeling attribute on the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionPinCutOut

```
dbSetInPlaceCoverObstructionPinCutOut(
     d_InstHeaderId
     t_layerName
     g_value
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction PinCutOut attribute on the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.
g_value	A valid PinCutOut value.

Value Returned

`t`	The specified PinCutOut attribute value is set for the specified instance header.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
         instHeaderId = inst~>instHeader
         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
         equal(dbGetInPlaceCoverObstructionPinCutOut(instHeaderId "Layer1") nil)
         dbSetInPlaceCoverObstructionPinCutOut(instHeaderId "Layer1" t)
         equal(dbGetInPlaceCoverObstructionPinCutOut(instHeaderId "Layer1") t)
```

Sets the InPlaceCoverObstruction PinCutOut attribute value `true` for the instance header `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionPinRemodeling

```
dbSetInPlaceCoverObstructionPinRemodeling(
     d_InstHeaderId
     t_remodelingName
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction PinRemodeling attribute on the specified instance header.

Arguments

d_InstHeaderId	ID of an instance header.
t_remodelingName	Valid PinRemodeling attribute value.

Value Returned

`t`	The specified InPlaceCoverObstruction PinRemodeling attribute Value is set on the specified instance.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)

         instHeaderId = inst~>instHeader

         dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))

         dbSetInPlaceCoverObstructionPinRemodeling(instHeaderId "perimeterOnly")

         equal(dbGetInPlaceCoverObstructionPinRemodeling(instHeaderId)
"perimeterOnly")
```

Sets value of the InPlaceCoverObstruction PinRemodeling attribute `"perimeterOnly"` on the instance header `instHeaderId`.

© 2020 All Rights Reserved.

dbSetInPlaceCoverObstructionSpacingModel

```
dbSetInPlaceCoverObstructionSpacingModel(
     d_InstHeaderId
     t_layerName
     t_spacingModelName
     )
     => t / nil
```

Description

Sets value of the InPlaceCoverObstruction SpacingModel attribute on the specified instance header on the given layer.

Arguments

d_InstHeaderId	ID of an instance header.
t_layerName	Name of a physical layer specified in the technology file.
t_spacingModelName	A valid SpacingModel value.

Value Returned

`t`	The InPlaceCoverObstruction SpacingModel attribute value is set on the specified instance header on the given layer.
`nil`	The operation failed.

Example

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)
     instHeaderId = inst~>instHeader
     dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
     equal(dbGetInPlaceCoverObstructionSpacingModel(instHeaderId "Layer1")
     "spacing"))
     dbSetInPlaceCoverObstructionSpacingModel(instHeaderId "Layer1" "asIs")
    equal(dbGetInPlaceCoverObstructionSpacingModel(instHeaderId "Layer1") "asIs")
```

Sets the InPlaceCoverObstruction SpacingModel attribute value `"asIs"` for the instance `instHeaderId` on the layer `Layer1`.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

11

Figure Database Access Functions

This chapter includes the following topics:

- FigGroup Creation and Modification Functions
- Figure Creation and Retrieval Functions

© 2020 All Rights Reserved.

FigGroup Creation and Modification Functions

The following functions create and modify figGroups.

- dbCreateFigGroup
- dbAddFigToFigGroup
- dbRemoveFigFromFigGroup
- dbGetFigGroupByName

© 2020 All Rights Reserved.

dbCreateFigGroup

```
dbCreateFigGroup(
     d_cellView
     t_name
     g_ordered
     l_origin
     t_orient
     )
     => d_figGroup / nil
```

Description

Creates a figGroup object in the specified cellview.

Arguments

d_cellView	The cellview ID.
t_name	The name of the figGroup.
g_ordered	Boolean specifying order.
l_origin	Origin of the new instance.
t_orient	Specifies the orientation of the figGroup.

Value Returned

d_figGroupId	Returns the database ID of the figGroup created.
`nil`	An error occurred during execution.

Example

The following creates an ordered figGroup by the name of `fg2` with an origin of 5:6 and with the orientation of R0.

```
fg2 = dbCreateFigGroup(cv "fg2" t 5:6 "R0")
```


© 2020 All Rights Reserved.

dbAddFigToFigGroup

```
dbAddFigToFigGroup(
     d_figGroupId
     d_figId
     )
     => t / nil
```

Description

Adds a figure to the specified figGroup.

Arguments

d_figGroup	The figGroup ID.
d_fig	The ID of the figure to be added.

Value Returned

`t`	The figure was added successfully.
`nil`	An error occurred during execution.

Example

The following creates a figGroup.

```
figGroup1 = dbCreateFigGroup(cv "figGroup1" t 0:0 "R0")
```

The following creates a figure.

```
figOfRect = dbCreateRect(cv "METAL1" list(0:0 6.3:8.7)
```

The following adds the figure to the figGroup.

```
dbAddFigToFigGroup(figGroup1 figOfRect)
```


© 2020 All Rights Reserved.

dbRemoveFigFromFigGroup

```
dbRemoveFigFromFigGroup(
     d_figGroupId
     d_figId
     )
     => t / nil
```

Description

Removes a figure from a figGroup object.

Arguments

d_figGroup	The figGroup ID.
d_fig	The ID of the figure to be removed.

Value Returned

`t`	The figure was removed from the figGroup successfully.
`nil`	An error occurred during execution.

Example

The following removes the figure `figOfRect` from the figGroup `figGroup1`.

```
dbRemoveFigFromFigGroup(figGroup1 figOfRect)
```


© 2020 All Rights Reserved.

dbGetFigGroupByName

```
dbGetFigGroupByName(
     d_cellView
     t_name
     )
     => d_figGroupId / nil
```

Description

Retrieves a figGroup by name.

Arguments

d_cellView	The cellview ID.
t_name	The name of the figGroup.

Value Returned

d_figGroupId	The figGroup ID is returned successfully.
nil	An error occurred during execution.

Example

The following returns the database ID of the specified figGroup.

```
dbGetFigGroupByName(cv "myFigGroup")
```


© 2020 All Rights Reserved.

Figure Creation and Retrieval Functions

The following functions create and retrieve figures (both shapes and instances, including mosaics) from the database.

Supported functions include:

- dbCopyFig
- dbMoveFig
- dbGetOverlaps
- dbGetTrueOverlaps
- dbGetNeighbor
- dbGetNeighborList
- dbGetMaxHierDepth
- dbGetPathSegStyle
- dbSetPathSegStyle
- dbProduceOverlap
- dbProduceOverlapInst

© 2020 All Rights Reserved.

dbCopyFig

```
dbCopyFig(
     d_fig
     d_cellView
     [ l_transform ]
     )
     => d_fig / nil
```

Description

Copies and transforms a figure, and places it in a cellview.

If *d_cellView* is `nil`, the figure is copied in the same cellview as the original figure.

Note: (ICADVM20.1 Only) By default, coloring information is not copied. To copy coloring information, set the `cdba copyMPAttributes` environment variable to `t` before using `dbCopyFig`.

When copying a `widthSpacingPattern` region:

- This function copies all Width Spacing Pattern (WSP) attributes except the committed pattern attributes. This is because the committed pattern attributes are recomputed by DRD based on the target location.
- This function skips the shapes in the WSP region for which the createType is `systemCreated`. This is because these shapes are maintained by the system and are not intended for interactive editing.

© 2020 All Rights Reserved.

Arguments

d_fig	Original *dbObject* of the instance to be copied.
d_cellView	The cellview to which the figure is copied; if `nil`, the figure is copied to the same cellview.
l_transform	Describes the transformation of the original figure during the copy process, such as offset, rotation, and magnification.

Value Returned

d_fig	A new *dbObject.*
`nil`	The figure is not copied or transformed.

Example

```
dstFig = dbCopyFig(srcFig dstCellView list(0:0 "R90" 2.0))
```


© 2020 All Rights Reserved.

dbMoveFig

```
dbMoveFig(
     d_fig
     d_cellView
     [ l_transform ]
     )
     => d_fig / nil
```

Description

Moves and transforms a figure, and places it in a cellview.

If *d_cellView* is `nil`, the figure is moved to the same cellview as the original figure and keeps the same ID. If *d_cellView* is not `nil`, the figure is copied to the specified cellview, the original figure is deleted, and the new figure has a different ID.

Note: (ICADVM20.1 Only) By default, coloring information is not copied. To copy coloring information, set the `cdba copyMPAttributes` environment variable to `t` before using `dbMoveFig`.

Arguments

d_fig	Original *dbObject* of the instance to be moved.
d_cellView	The cellview to which the figure is moved; if `nil`, the figure is moved within the same cellview.
l_transform	Describes the transformation of the figure during the move, such as offset, rotation, and magnification.

Value Returned

d_fig	The *dbObject* of the figure.
`nil`	The figure is not moved or transformed.

Example

```
dstFig = dbMoveFig(srcFig nil list(-1:2 "R180"))
```


© 2020 All Rights Reserved.

dbGetOverlaps

```
dbGetOverlaps
     dbGetOverlaps(
     d_cellView
     l_bBox
     [ tx_layer ]
     [ lx_level ]
     [ g_doRowCol ]
     )
     => ld_fig

     dbGetOverlaps(
     d_cellView
     l_bBox
     [ l_layerPurposePair ]
     [ lx_level ]
     [ g_doRowCol ]
     )
     => ld_fig
```

Description

Returns a list of all the objects (such as, shapes, instances, and mosaics) in a cellview that overlaps the area specified by *l_bBox*.
If the object comes from a lower level in the hierarchy, instead of from the specified cellview, the object is represented by a list (potentially a nested list, if it comes from more than one level down). Each level of the list has the ID of the instance in the corresponding level in the hierarchy.
For example, if there is a rectangle in a master cellview two levels down, for this rectangle, in the top cellview, `dbGetOverlaps` will return a nested list. The output of this function is same as dbGetTrueOverlaps.

Note: From OA 2.2 release onward, the shape query returns only those shapes whose geometry overlap the region. This means that the trueOverlap rule is applied to all shapes at all times. As a result no additional processing is required for performing trueOverlap.

Arguments

d_cellView	Database ID of a cellview.
l_bBox	The searching box.

© 2020 All Rights Reserved.

tx_layer	Specifies the layer name or number. If specified, the function produces instances, mosaics, or shapes (the default). If not specified, `dbGetOverlaps` produces only instances and mosaics.
tx_purpose	Specifies the layer purpose.
lx_level	Search level. It can be a list of two integers (start and stop level for the search), or a single integer (stop level, start level is zero). If no level argument is specified, the start and stop levels are both zero, so that only the top-level figures are returned.
l_layerPurposePair	The layer-purpose pair.
g_doRowCol	If set to `true`, displays row and column value, if the overlapped object is a mosaic instance.

Value Returned

ld_fig	A list identifying all the objects encountered in the search. If the object is in the given cellview, it is identified by its own ID. Otherwise, the object is identified by its hierarchical path, represented as a list of object IDs where the length of the list is one greater than the level of the object.
`nil`	An error occurs.

Examples

shapeList finds all the shapes on the device layer with drawing purpose on the top level.

```
shapeList = dbGetOverlaps(cv cv~>bBox "device")
```

shapeList finds all the shapes on the device layer (with any purpose) on the top level.

```
shapeList = dbGetOverlaps(cv cv~>bBox list("device"))
```

shapeList finds all the shapes on the device layer with pin purpose on the top level.

```
shapeList = dbGetOverlaps(cv cv~>bBox list("device" "pin"))
```

figList finds all the mosaics and instances on the top level.

```
figList = dbGetOverlaps(cv cv~>bBox nil)
```

figList finds all the shapes, mosaics and instances on the top level.

© 2020 All Rights Reserved.

```
figList = dbGetOverlaps(cv cv~>bBox)
```

figList finds all the shapes, mosaics and instances from the top level to the level 5.

```
figList = dbGetOverlaps(cv cv~>bBox t 0:5)
```

figList finds all the mosaics and instances on the top level.

```
figList = dbGetOverlaps(cv cv~>bBox nil)
```

figList finds all the mosaics and instances from the top level to the level 3.

```
figList = dbGetOverlaps(cv cv~>bBox nil 3)
```

The illustration below shows that *l_bBox* overlaps the perimeters of B, C, and D.

© 2020 All Rights Reserved.

dbGetTrueOverlaps

```
dbGetTrueOverlaps(
     d_cellView
     l_bBox
     [ l_layerPurposePair | tx_layer | g_value {t | nil} ]
     [ lx_level ]
     [ g_doRowCol ]
     )
     => ld_fig / nil
```

Description

Returns a list of all the objects (such as, shapes, instances, and mosaics) in a cellview that overlap the area specified by *l_bBox*.
If the object comes from a lower level in the hierarchy, instead of from the specified cellview, the object is represented by a list (potentially a nested list, if it comes from more than one level down). Each level of the list has the ID of the instance in the corresponding level in the hierarchy.
For example, if there is a rectangle in a master cellview two levels down, for this rectangle, in the top cellview, the function will return a nested list. The output of this function is same as dbGetOverlaps.
You can switch the output syntax of `dbGetTrueOverlaps` between the new and legacy format by setting the `noDetailedRow` .cdsenv environment variable.

Note: From OA 2.2 release onward, the shape query returns only those shapes whose geometry overlap the region. This means that the trueOverlap rule is applied to all shapes at all times. As a result no additional processing is required for performing trueOverlap.

Arguments

d_cellView	Database ID of a cellview
l_bBox	The searching box.
tx_layer	A layer name or number, followed by an optional list of start and stop levels (*lx_level*) and optional rowCol boolean (*g_doRowCol*). If specified, the function produces instances, mosaics, or shapes (the default). If not specified, `dbGetOverlaps` produces only instances and mosaics.

© 2020 All Rights Reserved.

l_layerPurposePair	The layer purpose pair, followed by an optional list of start and stop levels (*lx_level*) and optional rowCol boolean (*g_doRowCol*).
g_value	A Boolean, followed by an optional list of start and stop levels (*lx_level*) and optional rowCol boolean (*g_doRowCol*).
lx_level	Search level. It can be a list of two integers (start and stop level for the search), or a single integer (stop level, start level is zero). If no level argument is specified, the start and stop levels are both zero, so that only the top-level figures are returned.
g_doRowCol	If set to `true`, displays row and column value, if the overlapped object is a mosaic instance.

Value Returned

ld_fig	A list identifying all the objects encountered in the search. If the object is in the given cellview, it is identified by its own ID. Otherwise, the object is identified by its hierarchical path, represented as a list of object IDs where the length of the list is one greater than the level of the object.
`nil`	An error occurs.

Examples

shapeList finds all the shapes on the device layer with drawing purpose on the top level.

```
shapeList = dbGetTrueOverlaps(cv cv~>bBox "device")
```

shapeList finds all the shapes on the device layer (with any purpose) on the top level.

```
shapeList = dbGetTrueOverlaps(cv cv~>bBox list("device"))
```

shapeList finds all the shapes on the device layer with pin purpose on the top level.

```
shapeList = dbGetTrueOverlaps(cv cv~>bBox list("device" "pin"))
```

figList finds all the mosaics and instances on the top level.

```
figList = dbGetTrueOverlaps(cv cv~>bBox nil)
```

figList finds all the shapes, mosaics, and instances on the top level.

```
figList = dbGetTrueOverlaps(cv cv~>bBox)
```


© 2020 All Rights Reserved.

figList finds all the shapes, mosaics, and instances from the top level to the level 5.

```
figList = dbGetTrueOverlaps(cv cv~>bBox t 0:5)
```

The illustration below shows that *l_bBox* overlaps the perimeters of B, C, and D.

For more information, see the noDetailedRow environment variable.

© 2020 All Rights Reserved.

dbGetNeighbor

```
dbGetNeighbor(
     d_cellViewId
     l_bBox
     t_direction
     [ ( t_layer [ t_purpose ] ) ]
     [ x_depth ]
     )
     => ld_closestNeighbor / nil
```

Description

Returns a list containing information about the closest neighboring shape in the specified direction of the given layer-purpose pair.

Shapes that overlap the searching box are ignored.

Arguments

d_cellViewId	Cellview ID.
l_bBox	Specifies a bounding box that defines the search area.
t_direction	Direction to search for the neighboring shape. The following strings identify the directions that can be specified: `T`, `t`, or `top` Above the searching box. `B`, `b`, or `bottom` Below the searching box. `L`, `l`, or `left` To the left of the searching box. `R`, `r`, or `right` To the right of the searching box.
t_layer	Layer specification with drawing purpose. Only the bounding boxes of the shapes on this layer-drawing pair are checked against the searching box.
t_purpose	Layer-purpose pair specification. Only the bounding boxes of the shapes on this layer-purpose pair are checked against the searching box.
x_depth	The depth hierarchy up to which the search needs to be performed. If `x_depth` is 0, only the top-level shapes are returned.

Note: If you do not specify the values for the optional arguments, `dbGetNeighbor` uses the default values. For example,

© 2020 All Rights Reserved.

- If `t_purpose` is not specified, the `(t_layer, drawing)` layer purpose pair is used.
- If both `t_layer` and `t_purpose` are not specified, all layer purpose pairs are used.
- If `x_depth` is not specified, 0 depth is used.

Value Returned

`ld_closestNeighbor`

A list containing the following information about the closest neighboring shape:
The distance to the closest shape in the specified direction.
The closest object ID identified by its hierarchical path.
The co-ordinate pair representing the position of the closest shape.

`nil` The closest neighboring shape in the specified direction has not been found.

Example

```
closestID = dbGetNeighbor(cell3 ((10 5)(25 12)) "T")
```

In the example, `dbGetNeighbor` searches for the closest neighboring shape above the searching box. It returns the information about B because B is the closest shape above the specified searching box.

© 2020 All Rights Reserved.

Shapes D and E are ignored because they are fully or partially within the searching box. Shapes C and F are ignored because they do not satisfy the search constraints. Although shape A is above the box, it is ignored because it is not the closest shape in that direction.

© 2020 All Rights Reserved.

dbGetNeighborList

```
dbGetNeighborList(
     d_cellView
     l_area
     t_direction
     [ ( t_layer [ t_purpose ] ) ]
     [ lx_depth ]
     )
     => l_closestNeighbor
```

Description

Lists neighboring shapes in the specified direction of the given layer-purpose pair.

Shapes that overlap the searching box are ignored.

Arguments

d_cellView	Cellview.
l_area	Specifies a bounding box that defines the search area.
t_direction	Direction in which to search for neighboring shapes. The following strings identify the directions that can be specified: `T`, `t`, or `top` Above the searching box. `B`, `b`, or `bottom` Below the searching box. `L`, `l`, or `left` To the left of the searching box. `R`, `r`, or `right` To the right of the searching box.
t_layer	Specifies layer with the drawing purpose. Only the bounding boxes of the shapes on this layer-drawing pair are checked against the searching box.
t_purpose	Specifies layer-purpose pair. Only the bounding boxes of the shapes on this layer-purpose pair are checked against the searching box.
lx_depth	Search level, that is, the depth hierarchy up to which the search needs to be performed. If `lx_depth` is 0, only the top-level shapes are returned.

Note: If you do not specify the values for the optional arguments, `dbGetNeighbor` uses the default values. For example,

© 2020 All Rights Reserved.

- If `t_purpose` is not specified, the `(t_layer, drawing)` layer purpose pair is used.
- If both `t_layer` and `t_purpose` are not specified, all layer purpose pairs are used.
- If `lx_depth` is not specified, 0 depth is used.

Value Returned

`l_closestNeighbor`	A list of `ld_closestNeighbor` (in the search direction). The list is in no specific order, but you can put the list into order of increasing distance with a sorting function.

Example

```
closestIDs = dbGetNeighborList(cell3 ((10 5)(20 12)) "T")
```

In the example, `dbGetNeighborList` searches for the closest neighboring shapes that are above the searching box.

It returns the information about shapes A and B because both shapes are above the specified searching box. Shapes C and F are ignored because they do not satisfy the search constraints and shapes D and E are ignored because they overlap the searching box.

© 2020 All Rights Reserved.

dbGetMaxHierDepth

```
dbGetMaxHierDepth(
     )
     => x_maxHierDepth
```

Description

Returns the `dbcMaxHierDepth` constant in SKILL.

Arguments

None

Value Returned

x_maxHierDepth	The `dbcMaxHierDepth` constant.

© 2020 All Rights Reserved.

dbGetPathSegStyle

```
dbGetPathSegStyle(
     d_pathSeg
     )
     => l_pathSegStyle
```

Description

Returns the path segment end styles for the supplied pathSeg database object.

Arguments

d_pathSeg	Specifies the database identifier of the path segment object

Value Returned

l_pathSegStyle	List of lists containing the following path seg style parameters: (*n_width* *t_beginStyle* *t_endStyle* [*n_beginExt n_endExt* [*n_beginLeftDiagExt* *n_beginRightDiagExt n_beginRightHalfWidth* *n_endLeftDiagExt n_endRightDiagExt* *n_endRightHalfWidth*]])

Example

```
dbGetPathSegStyle(d_pathSeg)
(1.0 "custom" "custom"
    (0.5 0.5
    (0.71 0.71 0.5 0.71 0.71
        0.5
    )
     )
)
```


© 2020 All Rights Reserved.

dbSetPathSegStyle

```
dbSetPathSegStyle(
     d_pathSeg
     n_width t_beginStyle t_endStyle [ n_beginExt n_endExt
         [ n_beginLeftDiagExt n_beginRightDiagExt n_beginRightHalfWidth
         n_endLeftDiagExt n_endRightDiagExt n_endRightHalfWidth ]]
     )
     => t / nil
```

Description

Sets the path segment style of a given path seg database object.

Arguments

d_pathSeg	Specifies the database identifier of the pathSeg object
n_width t_beginStyle t_endStyle [*n_beginExt n_endExt* [*n_beginLeftDiagExt n_beginRightDiagExt n_beginRightHalfWidth n_endLeftDiagExt n_endRightDiagExt n_endRightHalfWidth*]]	Specifies the path seg style parameters

Value Returned

`t`	If the path seg style is successfully set
`nil`	If the path seg style is not set

Example

```
dbSetPathSegStyle(p2 1.0 "custom" "custom" list(0.5 0.5 list(0.71 0.71 0.5 0.71
0.71 0.5)))
```


© 2020 All Rights Reserved.

dbGetPathBoundary

```
dbGetPathBoundary(
     d_path
     )
     => l_points / nil
```

Description

Returns the polygon boundary for the specified path.

Arguments

d_path	Database ID of a path.

Value Returned

l_path	List of vertices for the polygon boundary.
nil	Returned in case of failure.

Example

```
dbGetPathBoundary(dbCreatePath(cv '("text" "drawing") list(0:0 10:0 10:10 0:0)
0.1))
> ((0.0 -0.05)
    (10.05 -0.05)
    (10.05 10.02)
    (9.98 10.05)
    (-0.035 0.035)
    (0.035 -0.035)
    (9.95 9.88)
    (9.95 0.05)
    (0.0 0.05)
)
```


© 2020 All Rights Reserved.

dbProduceOverlap

```
dbProduceOverlap(
     d_cellView
     l_bBox
     [ lx_level
     [ l_layerPurposePair ] ]
     )
     => ld_shape / nil
```

Description

Returns a list of all the shapes in a cellview whose bounding boxes overlap the area specified by *l_bBox*. This function interface is simpler than `dbGetOverlaps` if you are only interested in finding the overlapping shapes.

Arguments

d_cellView	The cellview.
l_bBox	Searching box.
lx_level	Search level. It can be two integers (start and stop level for the search), or a single integer (stop level, start level is zero). If no level argument is specified, the start and stop levels are both zero, so that only the top-level shapes are returned.
l_layerPurposePair	Layer purpose pair. Defaults to `all` if not specified.

Value Returned

ld_shape	A list identifying all the overlapping shapes.
`nil`	An error occurred.

Examples

shapeList finds all the shapes on the top level.

```
shapeList = dbProduceOverlap(cv cv~>bBox)
```

shapeList finds all the shapes from level 1 to level 5.

```
shapeList = dbProduceOverlap(cv cv~>bBox 1:5)
```


© 2020 All Rights Reserved.

shapeList finds all the shapes from the top level to level 4.

```
shapeList = dbProduceOverlap(cv cv~>bBox 4)
```

shapeList finds all the shapes on device layer with drawing purpose from the top level to level 4.

```
shapeList = dbProduceOverlap(cv cv~>bBox 4 "device")
```


© 2020 All Rights Reserved.

dbProduceOverlapInst

```
dbProduceOverlapInst(
     d_cellView
     l_bBox
     [ lx_level ]
     )
     => ld_inst / nil
```

Description

Returns a list of all the instances in a cellview that overlap the area specified by *l_bBox*. This function interface is simpler than `dbGetOverlaps` if you are only interested in finding instances.

Arguments

d_cellView	The cellview.
l_bBox	Searching box.
lx_level	Search level. Must be two integers (start and stop level for the search). If no level argument is specified, the start and stop levels are both zero, so that only the top-level instances are returned.

Value Returned

ld_inst	A list identifying all the overlapping instances.
`nil`	An error occurs.

Examples

instList finds all the instances on the top level.

```
skillinstList = dbProduceOverlapInst(cv cv~>bBox)
```

instList finds all the instances from level 1 to level 5.

```
instList = dbProduceOverlapInst(cv cv~>bBox 1:5)
```

instList finds all the instances from the top level to level 4.

```
instList = dbProduceOverlapInst(cv cv~>bBox 4:4)
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

12

Figure Set Database Access Functions

This chapter includes the following topics:

- FigSet Functions
- FigSetGroup Functions

© 2020 All Rights Reserved.

FigSet Functions

A FigSet is a high performance, ordered grouping of database geometric objects or other FigSets. A member of a FigSet is unique within that FigSet, but the same geometric object can be added to multiple FigSets. The name of a FigSet is unique within the cellview it is associated with.

FigSets offer high performance because they are neither database objects nor saved to the disk. Consequently, operations performed on a FigSet cannot be undone. If you delete an object that is a member of a FigSet, the object is removed from the FigSet, and when you undo delete, the object is restored, but it is not added back to the FigSet. To store the contents of a FigSet to disk, you need to create a corresponding FigSetGroup. For information about FigSetGroups, see FigSetGroup Functions.

FigSet SKILL objects are like CDB SKILL objects on which certain "~>" (squiggle-arrow) queries can be run:

- `figSet~>objType` — Returns the object type, `figSet` in this case
- `figSet~>name` — Returns the name of the FigSet
- `figSet~>cellView` — Returns the ID of the cellview associated with the FigSet
- `figSet~>length` — Returns the number of objects in the FigSet
- `figSet~>members` — Returns the list of objects in the FigSet

FigSet functions are used to add, find, move, copy, append, prepend, clone, clear, and delete FigSet objects; access FigSet parameters; and iterate over the objects stored in a cellview.

Supported functions include:

- txCreateFigSet
- txGetFigSetData
- txAppendObjectToFigSet
- txClearFigSet
- txCloneFigSet
- txConcatFigSet
- txCopyFigSet
- txFindFigSet
- txDeleteFigSet

© 2020 All Rights Reserved.

- txGetCellViewFigSets
- txGetFigSetBBox
- txIsFigSet
- txMakeFigSet
- txMoveFigSet
- txPrependObjectToFigSet
- txRemoveFigSetData
- txRemoveObjectFromFigSet

© 2020 All Rights Reserved.

txCreateFigSet

```
txCreateFigSet(
     d_cellView
     [ t_name ]
     )
     => T_figSet / nil
```

Description

Creates an empty FigSet with the specified name in the given cellview. If the name is not specified, it will be generated automatically. If a FigSet with the specified name already exists, an error message is issued.

Arguments

d_cellView	ID of the cellview.
t_name	Name of the FigSet.

Value Returned

T_figSet	ID of the created FigSet.
`nil`	Returned in case of failure.

Example

```
txCreateFigSet(cv "figset0")
> tx@02af68cf
```

Creates an empty FigSet with the name `figset0` in the cellview `cv`.

© 2020 All Rights Reserved.

txGetFigSetData

```
txGetFigSetData(
     T_figSet
     n_entry
     )
     => g_object / nil
```

Description

Returns the nth object from the specified FigSet. This object is either a dbObject (CDB figure) or txObject (a FigSet or a transient shape).

Arguments

T_figSet	ID of the FigSet.
n_entry	Index of the object to return.

Value Returned

g_object	The nth object in the specified FigSet.
nil	Returned in case of failure.

Example

```
id = txGetFigSetData(fs 0)
> tx@0x1fc106d0
id~>objType
> "FigSet"
id = txGetFigSetData(fs 1)
> db:0x27a9dc1a
id~>objType
> "polygon"
```

The above functions return the first and the second object from the FigSet `fs`, respectively.

© 2020 All Rights Reserved.

txAppendObjectToFigSet

```
txAppendObjectToFigSet(
     T_figSet
     g_figObject
     )
     => t / nil
```

Description

Appends an object to the end of the specified FigSet. The object can be a figure or a FigSet.

Arguments

T_figSet	ID of the FigSet.
g_figObject	ID of the appended object.

Value Returned

`t`	The object was appended.
`nil`	Returned in case of failure.

Example

```
fs12 = txCreateFigSet(cv "fs12")
> tx@0x19d35cf0
txAppendObjectToFigSet(fs12 txCreateFigSet(cv "fs0"))
> t
txAppendObjectToFigSet(fs12 dbCreateRect(cv '(0 "drawing") '((0 0) (200 100))))
> t
fs12~>length
> 2
```

Appends a FigSet object (`fs0`) and a CDB object (a rectangle) to the FigSet `fs12`.

© 2020 All Rights Reserved.

txClearFigSet

```
txClearFigSet(
     T_figSet
     )
     => t / nil
```

Description

Clears the contents of the specified FigSet.

Arguments

T_figSet	ID of the FigSet.

Value Returned

`t`	The contents were cleared.
`nil`	Returned in case of failure.

Example

```
fs0 = txCreateFigSet(cv "fs0")
> tx@0x19d35cf0
fs1 = txCreateFigSet(cv "fs1")
> tx@0x19a46350
txAppendObjectToFigSet(fs1 fs0)
> t
fs1~>length
> 1
txClearFigSet(fs1)
> t
fs1~>length
> 0
```

Clears the contents of the FigSet `fs1`.

© 2020 All Rights Reserved.

txCloneFigSet

```
txCloneFigSet(
     T_figSet1
     T_figSet2
     )
     => t / nil
```

Description

Makes FigSet1 a clone of FigSet2 by replacing the contents of FigSet1with equivalent entries from FigSet2. It creates references to the objects; it does not make copies of the objects. The name of FigSet1 is preserved.

Arguments

T_figSet1	ID of the target FigSet.
T_figSet2	ID of the source FigSet.

Value Returned

`t`	The cloning is successful.
`nil`	Returned in case of failure.

Example

```
fs_src = txCreateFigSet(cv "fs_src")
> tx@0x2294f7c0
fs_dest = txCreateFigSet(cv "fs_dest")
> tx@0x2294ff10
txAppendObjectToFigSet(fs_src txCreateFigSet(cv "fs0"))
> t
txCloneFigSet(fs_dest fs_src)
> t
txGetFigSetData(fs_dest 0)~>name
> "fs0"
```

Makes `fs_dest` a clone of `fs_src`.

© 2020 All Rights Reserved.

txConcatFigSet

```
txConcatFigSet(
     T_figSet1
     T_figSet2
     )
     => t / nil
```

Description

Appends the contents of FigSet2 to the end of FigSet1.

Arguments

T_figSet1	ID of the target FigSet.
T_figSet2	ID of the source FigSet.

Value Returned

`t`	The contents were appended successfully.
`nil`	Returned in case of failure.

Example

```
fs_src = txCreateFigSet(cv "fs_src")
> tx@0x1fc106d0
fs_dest = txCreateFigSet(cv "fs_dest")
> tx@0x1fc10e20
txAppendObjectToFigSet(fs_src txCreateFigSet(cv "fs0"))
> t
txAppendObjectToFigSet(fs_src txCreateFigSet(cv "fs1"))
> t
txConcatFigSet(fs_dest fs_src)
> t
txGetFigSetData(fs_dest 0)~>name
> "fs0"
txGetFigSetData(fs_dest 1)~>name
> "fs1"
```

Appends the contents of `fs_dest` to the end of `fs_src`.

© 2020 All Rights Reserved.

txCopyFigSet

```
txCopyFigSet(
     T_srcFigSet
     l_transform
     )
     => T_newFigSet / nil
```

Description

Copies each figure from the source FigSet and applies to it the specified transform to create a new FigSet with an automatically generated name.

Arguments

T_srcFigSet	ID of the source FigSet.
l_transform	`list(`*g_offset t_transform* `[`*f_scale*`])` A transformation object. where, ■ *g_offset*: `list(`*f_offsetX f_offsetY*`)` An offset vector. ■ *t_transform* A transformation name. Valid values are `"R0"`, `"R90"`, `"R180"`, `"R270"`, `"MX"`, `"MXR90"`, `"MY"`, and `"MYR90"`; or, their corresponding symbols such as `'R0`, `'R90`, `'R180`, `'R270`, `'MX`, `'MXR90`, `'MY`, and `'MYR90`. ■ *f_scale* A scale ratio (optional). Default value is `1.0`.

Value Returned

T_newFigSet	ID of the created FigSet.
`nil`	Returned in case of failure.

Example

```
fs0 = txCreateFigSet(cv "fs0")
```


© 2020 All Rights Reserved.

```
> tx@0x2c0295c0
txAppendObjectToFigSet(fs0 dbCreateRect(cv '(0 "drawing") '((0 0) (200 100))))
> t
fs1 = txCopyFigSet(fs0 '(((0 0) "R90")))
> tx@0x2c04f190
txGetFigSetData(fs1 0)~>bBox
> ((-100.0 0.0)
>     (0.0 200.0)
> )
```

Copies the source FigSet `fs0` and applies to it the `R90` transform to create a new FigSet `fs1`.

© 2020 All Rights Reserved.

txFindFigSet

```
txFindFigSet(
     d_cellView
     t_name
     )
     => T_figSet / nil
```

Description

Searches for a FigSet with the given name in the specified cellview. This function also searches for a FigSetGroup with the same name and automatically resurrects the corresponding FigSet from it, if required.

Arguments

d_cellView	ID of the cellview.
t_name	Name of the FigSet.

Value Returned

T_figSet	ID of the found FigSet.
nil	Returned in case of failure.

Example

```
txFindFigSet(cv "FS_0")
> tx@02af68ce
```

Searches for FigSet `FS_0` in the cellview `cv`.

© 2020 All Rights Reserved.

txDeleteFigSet

```
txDeleteFigSet(
     T_figSet
     [ g_deep = nil ]
     )
     => t / nil
```

Description

Deletes the specified FigSet. If the optional argument `g_deep` is set to true, then the FigSets referenced by the specified FigSet will also be recursively deleted.

Arguments

T_figSet	ID of the FigSet.
g_deep	A flag that determines whether the FigSets referenced by the specified FigSet should be deleted.

Value Returned

`t`	The specified FigSet was deleted successfully.
`nil`	Returned in case of failure.

Example

```
fs0 = txCreateFigSet(cv "fs0")
> tx@0x19d35cf0
fs1 = txCreateFigSet(cv "fs1")
> tx@0x19a46350
txAppendObjectToFigSet(fs1 fs0)
> t
txDeleteFigSet(fs1 t)
> t
txFindFigSet(cv "fs0")
> nil
txFindFigSet(cv "fs1")
> nil
```

Deletes the FigSet `fs1` and the FigSet referenced by it, `fs0`, from the cellview `cv`.

© 2020 All Rights Reserved.

txGetCellViewFigSets

```
txGetCellViewFigSets(
     d_cellView
     )
     => l_figSets / nil
```

Description

Returns the list of FigSets in the specified cellview.

Arguments

d_cellView	ID of the cellview.

Value Returned

l_figSets	List of FigSets.
nil	Returned in case of failure.

Example

```
txGetCellViewFigSets(cv)
> (tx@0x19d35cf0 tx@0x19a46350)
```

Returns the FigSets, `tx@0x19d35cf0` and `tx@0x19a46350`, in the cellview `cv`.

© 2020 All Rights Reserved.

txGetFigSetBBox

```
txGetFigSetBBox(
     T_figSet @key
     (onlyTheseLayers l_onlyTheseLayers)
     (onlyTheseLPPs l_onlyTheseLPPs)
     (notTheseLayers l_notTheseLayers)
     (notTheseLPPs l_notTheseLPPs)
     (onlyTheseTypes l_onlyTheseTypes)
     (notTheseTypes l_notTheseTypes)
     )
     => l_bBox / nil
```

Description

Calculates the bounding box for the objects in the specified FigSet. The caller can optionally specify included and excluded layers, included and excluded layer-purpose pairs (LPPs), and included and excluded object types. The bounding box values are returned in user units.

Arguments

T_figSet	ID of the FigSet.
l_onlyTheseLayers	`list([t_layerName | n_layerNum] ...)` A list of layer names or layer numbers. If specified, only the layers specified in the list are considered during the bounding box calculation.

l_onlyTheseLPPs	`list([l_LPP] ...)`
	A list of layer-purpose pairs. If specified, only the layer-purpose pairs specified in the list are considered during the bounding box calculation.
	where,
	`l_LPP: list([t_layerName | n_layerNum] t_purposeName)`
	A layer-purpose pair.

l_notTheseLayers	`list([t_layerName | n_layerNum] ...)` A list of layer names or layer numbers. If specified, the layers specified in the list are excluded from the bounding box calculation.

© 2020 All Rights Reserved.

l_notTheseLPPs	`list([l_LPP] ...)` A list of layer-purpose pairs. If specified, the layer-purpose pairs specified in the list are excluded from the bounding box calculation.
l_onlyTheseTypes	`list(t_typeName ...)` A list of CDB type names. If specified, only the objects of the types specified in the list are considered during the bounding box calculation.
l_notTheseTypes	`list(t_typeName ...)` A list of CDB type names. If specified, the objects of the types specified in the list are excluded from the bounding box calculation.

Value Returned

l_bBox	Bounding box for the specified objects.
`nil`	Returned in case of failure.

Example

```
rect0 = dbCreateRect(top '(0 "drawing") '((0 0) (100 100)))
> db:0x2e39ec9e
ellipse0 = dbCreateEllipse(top '(0 "drawing") '((0 0) (500 200)))
> db:0x2e39ec9f
fs0 = txCreateFigSet(top "fs0")
> tx@0x318ce1a0
txAppendObjectToFigSet(fs0 rect0)
> t
txAppendObjectToFigSet(fs0 ellipse0)
> t
txGetFigSetBBox(fs0)
> ((0.0 0.0)
>     (500.0 200.0)
> )
```

Calculates the bounding box for the objects in the FigSet `fs0`.

© 2020 All Rights Reserved.

```
txGetFigSetBBox(fs0 ?onlyTheseTypes '("rect"))
> ((0.0 0.0)
>     (100.0 100.0)
> )
```

Calculates the bounding box for the objects of type `rect` in the FigSet `fs0`.

© 2020 All Rights Reserved.

txIsFigSet

```
txIsFigSet(
     g_object
     )
     => t / nil
```

Description

Determines whether the specified object is a FigSet object.

Arguments

g_object	ID of the object.

Value Returned

t	The object is a FigSet object.
nil	Returned in case of failure.

Example

```
txIsFigSet(fs0)
> t
```

The object `fs0` is a FigSet object.

© 2020 All Rights Reserved.

txMakeFigSet

```
txMakeFigSet(
     d_cellView
     d_figSetGroup
     )
     => T_figSet / nil
```

Description

Creates a FigSet from the given FigSetGroup in the specified cellview.

The function can be used to resurrect a FigSet that had been saved to a FigSetGroup earlier. The name and contents of the FigSet will be extracted from the specified FigSetGroup. If a FigSet with the specified name already exists, its contents will be replaced.

Arguments

d_cellView	ID of the cellview.
d_figSetGroup	ID of the FigSetGroup.

Value Returned

T_figSet	ID of the created FigSet.
nil	Returned in case of failure.

Example

```
txAppendObjectToFigSet(figset12 txCreateFigSet(cv "figset1"))
> t
txAppendObjectToFigSet(figset12 txCreateFigSet(cv "figset2"))
> t
group0 = dbMakeFigSetGroup(cv figset12)
> db:0x27a9ec1a
txDeleteFigSet(figset12)
> t
txFindFigSet(cv "figset12")
> nil
fs = txMakeFigSet(cv group0)
> tx@0x2af32df0
```


© 2020 All Rights Reserved.

```
txGetFigSetData(fs 0)~>name
> "figset1"
txGetFigSetData(fs 1)~>name
> "figset2"
```

Creates a FigSet `fs` from the FigSetGroup `group0` in the cellview `cv`.

© 2020 All Rights Reserved.

txMoveFigSet

```
txMoveFigSet(
     T_figSet
     l_transform
     )
     => t / nil
```

Description

Moves each figure in the given FigSet using the specified transform.

Arguments

T_figSet	ID of the FigSet.
l_transform	`list(`*g_offset t_transform* `[`*f_scale*`])` A transformation object. where, ■ *g_offset*: `list(`*f_offsetX f_offsetY*`)` An offset vector. ■ *t_transform* A transformation name. Valid values are `"R0"`, `"R90"`, `"R180"`, `"R270"`, `"MX"`, `"MXR90"`, `"MY"`, and `"MYR90"`; or, their corresponding symbols such as `'R0`, `'R90`, `'R180`, `'R270`, `'MX`, `'MXR90`, `'MY`, and `'MYR90`. ■ *f_scale* A scale ratio (optional). Default value is `1.0`.

Value Returned

`t`	The figures were transformed successfully.
`nil`	Returned in case of failure.

Example

```
fs0 = txCreateFigSet(cv "fs0")
> tx@0x254996a0
```


© 2020 All Rights Reserved.

```
txAppendObjectToFigSet(fs0 dbCreateRect(cv '(0 "drawing") '((0 0) (200 100))))
> t
txMoveFigSet(fs0 '(((0 0) "R90")))
> t
txGetFigSetData(fs0 0)~>bBox
> ((-100.0 0.0)
>     (0.0 200.0)
> )
```

Transforms the figures in the FigSet `fs0` by the transformation value of `R90`.

© 2020 All Rights Reserved.

txPrependObjectToFigSet

```
txPrependObjectToFigSet(
     T_figSet
     g_figObject
     )
     => t / nil
```

Description

Inserts an object in the beginning of the specified FigSet. The object may be a figure or a FigSet.

Arguments

T_figSet	ID of the FigSet.
g_figObject	ID of the object.

Value Returned

`t`	The object was inserted.
`nil`	Returned in case of failure.

Example

```
fs12 = txCreateFigSet(cv "fs12")
> tx@0x19d35cf0
txAppendObjectToFigSet(fs12 txCreateFigSet(cv "fs1"))
> t
txPrependObjectToFigSet(fs12 txCreateFigSet(cv "fs0"))
> t
txGetFigSetData(fs12 0)~>name
> "fs0"
```

Inserts the object `fs0` in the beginning of the FigSet `fs12`.

© 2020 All Rights Reserved.

txRemoveFigSetData

```
txRemoveFigSetData(
     T_figSet
     n_entry
     )
     => t / nil
```

Description

Removes the nth object from the specified FigSet.

Arguments

T_figSet	ID of the FigSet.
n_entry	The index of the object to remove.

Value Returned

`t`	The object was removed successfully.
`nil`	Returned in case of failure.

Example

```
fs12 = txCreateFigSet(cv "fs12")
> tx@0x19d35cf0
txAppendObjectToFigSet(fs12 txCreateFigSet(cv "fs0"))
> t
txAppendObjectToFigSet(fs12 txCreateFigSet(cv "fs1"))
> t
txRemoveFigSetData(fs12 0)
> t
txGetFigSetData(fs12 0)~>name
> "fs1"
```

Removes the object at the first position from the FigSet `fs12`.

© 2020 All Rights Reserved.

txRemoveObjectFromFigSet

```
txRemoveObjectFromFigSet(
     T_figSet
     g_object
     )
     => t / nil
```

Description

Removes the specified object from the given FigSet, if the object is a member of this FigSet. This object can be a figure or a FigSet.

Arguments

T_figSet	ID of the FigSet.
g_object	ID of the object.

Value Returned

`t`	The object was removed successfully.
`nil`	Returned in case of failure.

Example

```
fs0 = txCreateFigSet(cv "fs0")
> tx@0x19d35cf0
fs1 = txCreateFigSet(cv "fs1")
> tx@0x19a46350
txAppendObjectToFigSet(fs1 fs0)
> t
txRemoveObjectFromFigSet(fs1 fs0)
> t
```

Removes the object `fs0` from the FigSet `fs1`.

© 2020 All Rights Reserved.

FigSetGroup Functions

The contents of a FigSetGroup are created from a FigSet. A FigSetGroup is a method to store the contents of a FigSet in database for use in a different session.

FigSetGroup functions are used to create, delete, access, manage, and search the FigSetGroup objects.

Supported functions include:

- dbMakeFigSetGroup
- dbFindFigSetGroup
- dbFindFigSetGroupByName
- dbGetFigSetGroupName
- dbGetCellViewFigSetGroups
- dbIsFigSetGroup
- dbDeleteFigSetGroup

© 2020 All Rights Reserved.

dbMakeFigSetGroup

```
dbMakeFigSetGroup(
     d_cellView
     T_figSet
     )
     => d_figSetGroup / nil
```

Description

Creates a FigSetGroup in the specified cellview and populates it with the contents of the specified FigSet.

The name for this FigSetGroup is the same as that of the FigSet. If a FigSetGroup with this name already exists, its contents are cleared, and it is populated from the FigSet.

Arguments

d_cellview	ID of the cellview.
T_figSet	ID of the FigSet.

Value Returned

d_figSetGroup	ID of the created FigSetGroup.
`nil`	Returned in case of failure.

Example

```
fs1 = txCreateFigSet(cv "figSet1")
> tx@0x362ed0d0
txAppendObjectToFigSet(fs1 txCreateFigSet(cv "figSet2"))
> t
fsgroup1 = dbMakeFigSetGroup(cv fs1)
> db:0x32e3f19b
(mapcar 'dbGetFigSetGroupName dbGetCellViewFigSetGroups(cv))
> ("figSet1" "figSet2")
```

Creates a FigSetGroup `fsgroup1` in the cellview `cv`.

© 2020 All Rights Reserved.

dbFindFigSetGroup

```
dbFindFigSetGroup(
     d_cellView
     T_figSet
     )
     => d_figSetGroup / nil
```

Description

Searches the specified cellview for a FigSetGroup that has the same name as the specified FigSet.

Arguments

d_cellView	ID of the cellview.
T_figSet	ID of the FigSet.

Value Returned

d_figSetGroup	ID of the FigSetGroup found.
nil	Returned in case of failure.

Example

```
dbFindFigSetGroup(cv figset1)
> db:0x32e3f19b
```

Searches for the FigSet name `figset1` in the cellview `cv`.

© 2020 All Rights Reserved.

dbFindFigSetGroupByName

```
dbFindFigSetGroupByName(
     d_cellView
     t_name
     )
     =>d_figSetGroup / nil
```

Description

Searches for the specified FigSetGroup name in the given cellview.

Arguments

d_cellView	ID of the cellview.
t_name	Name of the FigSetGroup.

Value Returned

d_figSetGroup	ID of the FigSetGroup found.
nil	Returned in case of failure.

Example

```
dbFindFigSetGroupByName(cv "fs0")
> db:0x32e3f19b
```

Searches for the FigSetGroup name `fs0` in the cellview `cv`.

© 2020 All Rights Reserved.

dbGetFigSetGroupName

```
dbGetFigSetGroupName(
     d_figSetGroup
     )
     => t_name / nil
```

Description

Returns the name of the specified FigSetGroup object.

Arguments

d_figSetGroup	ID of the FigSetGroup object.

Value Returned

t_name	Name of the FigSetGroup object.
nil	Returned in case of failure.

Example

```
dbGetFigSetGroupName(fsg1)
> "FS_0"
```

The name of the `fsg1` object is `FS_0`.

© 2020 All Rights Reserved.

dbGetCellViewFigSetGroups

```
dbGetCellViewFigSetGroups(
     d_cellView
     )
     => l_figSetGroups / nil
```

Description

Returns a list of FigSetGroups in the specified cellview.

Arguments

d_cellView	ID of the cellview.

Value Returned

l_figSetGroups	The list of FigSetGroups.
nil	Returned in case of failure.

Example

```
dbGetCellViewFigSetGroups(cv)
> (db:0x32e3f19b db:0x32e3f19c)
```

Returns the FigSetGroups `db:0x32e3f19b` and `db:0x32e3f19c` in the cellview `cv`.

© 2020 All Rights Reserved.

dbIsFigSetGroup

```
dbIsFigSetGroup(
     d_object
     )
     => t / nil
```

Description

Determines whether the specified object is a FigSetGroup.

Arguments

d_object	ID of the object.

Value Returned

`t`	The object is a FigSetGroup.
`nil`	The object is not a FigSetGroup.

Example

```
dbIsFigSetGroup(dbId)
> nil
```

The object `dbId` is not a FigSetGroup.

© 2020 All Rights Reserved.

dbDeleteFigSetGroup

```
dbDeleteFigSetGroup(
     d_figSetGroup
     )
     => t / nil
```

Description

Deletes the specified FigSetGroup, including the hierarchy of FigSetGroups that may exist in this FigSetGroup.

Arguments

d_figSetGroup	ID of the FigSetGroup to delete.

Value Returned

`t`	The FigSetGroup was deleted.
`nil`	Returned in case of failure.

Example

```
fs1 = txCreateFigSet(cv "figSet1")
> tx@0x362ed0d0
txAppendObjectToFigSet(fs1 txCreateFigSet(cv "figSet2"))
> t
fsgroup1 = dbMakeFigSetGroup(cv fs1)
> db:0x32e3f19b
dbGetCellViewFigSetGroups(cv)
> (db:0x32e3f19b db:0x32e3f19c)
dbDeleteFigSetGroup(fsgroup1)
> t
dbGetCellViewFigSetGroups(cv)
> nil
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

13

Multi-Patterning Technology Database Access Functions (ICADVM20.1 Only)

Multi-patterning technology uses two or more mask processes to manufacture a design layer. Colors are used to represent the individual masks. The following section describes the SKILL functions that support photo mask coloring of objects and coloring methods for MPT.

For more information about multi-patterning lithography and how to use Virtuoso® Multi-Patterning Technology (MPT) for designs that require two or more masks for a drawn layer, see *Virtuoso® Multi-Patterning Technology User Guide* in the ICADVM20.1 release.

Supported functions include:

- dbCellViewAreLayerShiftsValid
- dbCellViewClearLayerShifts
- dbCellViewHasLayerShifts
- dbCellViewUpdateLayerShifts
- dbCellViewInitForLayerShifting
- dbClearUnusedOccShapeColorData
- dbInstGetLayerShifts
- dbInstGetLayerShifts2
- dbInstHasLayerShifts
- dbInstSetLayerShifts
- dbInstSetLayerShifts2
- dbInstClearLayerShifts
- dbIsShapeColoringAllowed
- dbIsShapeColored

© 2020 All Rights Reserved.

- dbGetShapeColor
- dbSetShapeColor
- dbIsShapeColorLocked
- dbSetShapeColorLocked
- dbIsBlockageColored
- dbIsBlockageColoringAllowed
- dbGetBlockageColor
- dbSetBlockageColor
- dbIsViaColorStateLocked
- dbSetViaColorStateLocked
- dbIsViaColorStateLayerLocked
- dbSetViaColorStateLayerLocked
- dbGetViaLayerNumColorMasks
- dbGetViaLayer1Control
- dbGetViaLayer2Control
- dbGetViaCutLayerControl
- dbGetViaLayerControl
- dbSetViaLayer1Control
- dbSetViaLayer2Control
- dbSetViaCutLayerControl
- dbSetViaLayerControl
- dbSetViaColorInfo
- dbIsTrackPatternColored
- dbIsTrackPatternColorAlternating
- dbIsTrackPatternColoringAllowed
- dbGetTrackPatternFirstTrackColor
- dbSetTrackPatternFirstTrackColor

© 2020 All Rights Reserved.

- dbSetTrackPatternColorAlternating
- dbCreateSameMaskGroup
- dbCreateDiffMaskGroup
- dbGetSameMaskDiffMaskGroups
- dbGetShapeSameMaskGroups
- dbColorShapeQuery
- dbGetIntegrationColorModel
- dbGetColorModel
- dbSetIntegrationColorModel
- dbSetColorModel
- dbRemoveLayerColorData
- dbColorShapeQuery2
- dbGetColoredOccShapes
- dbGetShapeEffectiveColor
- dbSetOccShapeColor
- dbSetOccShapeColorLocked
- dbGetHierColorOverride
- dbGetShapeColorLockType
- dbSetShapeColorLockType
- dbGetViaLayerColorLockType
- dbSetViaLayerColorLockType
- dbCellViewHasUnknownColorLockChanges
- dbGetShapeEffectiveColorLockType
- dbCellViewResetAllColorLockTypes
- dbSetOccShapeColorLockType

© 2020 All Rights Reserved.

dbCellViewAreLayerShiftsValid

```
dbCellViewAreLayerShiftsValid(
     d_cellviewId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns a Boolean depending on whether the layer shift information on the instances in the specified cellview is synchronized with the layers and color information in the bound tech. This function is currently intended only for the advanced nodes release.

Layer shift information can become invalid if the cellview is not bound to a tech, or if the bound tech is modified after the layer shifts were specified on the instances. If the layer shift information is not valid, use the dbCellViewUpdateLayerShifts function to reset the layer shift information on the instances in the cellview.

Arguments

d_cellviewId	Specifies the cellview ID

Value Returned

`t`	Returns `t` if there is a valid per-layer shift on the instances in the cellview
`nil`	Returns `nil` if there is no valid per-layer shift on the instances in the cellview

Example

```
if( !dbCellViewAreLayerShiftsValid(cvId)
    dbCellViewUpdateLayerShifts(cvId)
)
```


© 2020 All Rights Reserved.

dbCellViewClearLayerShifts

```
dbCellViewClearLayerShifts(
     d_cellviewID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Clears the layer shift information from all instances in the specified cellview. Any cached layer shift specification for the tech bound to the cellview is left intact.

Arguments

d_cellviewID	Specifies the cellview ID

Value Returned

`t`	Returns `t` if the per-layer color shifting information is successfully removed
`nil`	Returns `nil` if the per-layer color shifting information is not removed from the specified cellview

Example

```
dbCellViewClearLayerShifts(dbId)
```


© 2020 All Rights Reserved.

dbCellViewHasLayerShifts

```
dbCellViewHasLayerShifts(
     d_cellviewID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns a Boolean indicating whether there is layer shift information on any instance in the specified cellview.

Arguments

d_cellviewID	Specifies the cellview ID

Value Returned

`t`	Returns `t` if there is layer shift information on any instance in the specified cellview
`nil`	Returns `nil` if there is no layer shift information on any instance in the specified cellview

Example

```
hasShifts = dbCellViewHasLayerShifts(cvId)
```


© 2020 All Rights Reserved.

dbCellViewUpdateLayerShifts

```
dbCellViewUpdateLayerShifts(
     d_cellviewID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Creates or rebuilds the layer shift specification cache with the layers that are in the tech bound to the specified cellview. As a result, the per-layer shift information on the instances in the cellview is removed.

Note: If there exists a cellview with ID `cv` and instance with ID `inst` in the same cellview, you must call dbCellViewUpdateLayerShifts`(cv)` before calling dbInstSetLayerShifts2`(inst...)`. Otherwise, `nil` is returned and layer shifts are not set.

Arguments

d_cellviewID	Specifies the cellview ID

Value Returned

`t`	Returns `t` if per-layer color specifications are successfully updated
`nil`	Returns `nil` if per-layer color specifications are not updated

Example

```
if( !dbCellViewAreLayerShiftsValid(cvId)
    dbCellViewUpdateLayerShifts(cvId)
)
```


© 2020 All Rights Reserved.

dbCellViewInitForLayerShifting

```
dbCellViewInitForLayerShifting(
     d_cellviewID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Initializes the layer shift data for the given cell.

It is an error if the technology is not bound to the design and there are colorShifts specified on any instances in the block.

Arguments

d_cellviewID	Specifies the cellview ID

Value Returned

`t`	Returns `t` if initialization is successful.
`nil`	Returns `nil` if initialization failed.

Example

```
dbCellViewInitForLayerShifting(dbObject)
```


© 2020 All Rights Reserved.

dbClearUnusedOccShapeColorData

```
dbClearUnusedOccShapeColorData(
     d_cellView
     [ ?dryRun { t | nil } ]
     [ ?ignoreUnboundRefs { t | nil } ]
     )
     => x_numClearedObjects / nil
```

Description

(ICADVM20.1 Only) Opens hierarchy below the given cellview and removes all occShape coloring data that does not have any effect on the display of coloring data in the opened hierarchy. By default, if some instances or vias in the hierarchy cannot be bound, the function prints a warning message and returns `nil`, but cleanup can be forced with the `ignoreUnboundRefs` argument.

Suppose you do out-of-context edits, such as destroy, stretch, or move the master shape to a design that contains master shape of the hierarchical color lock (HCL). Now if this is the shape on which HCL is set, then HCL might not be visible when you open this top design. In this case, HCL data in the top design is retained but becomes unused. This function helps cleanup this data.

Note: Until hierarchy is fully open it is not possible to distinguish between used or unused HCL data. The dbClearUnusedOccShapeColorData helps you view the unused data and clear it. This function does not cleanup HCL violations.

Arguments

d_cellView	Database ID of the cellview from which the unused occShape coloring data will be cleared.
?dryRun	When set to `t`, opens hierarchy below the specified cellview, and calculates the number of objects containing unused occShape coloring data, but does not remove this data. Default Value: `nil`
?ignoreUnboundRefs	When set to `t`, clears unused occShape coloring data even if hierarchy cannot be fully opened. Default Value: `nil`

© 2020 All Rights Reserved.

Value Returned

x_numClearedObjects	A non-negative number of objects cleared. If `dryRun` is set to `t`, this number indicates how many objects are found with the specified cellview, without clearing the data.
`nil`	Hierarchy below the given cellview cannot be opened or some other internal error occurred. Nothing was cleared.

Example

```
;; Open cellview 'myLib/myCell/myLayout":
window = geOpen(?lib "myLib" ?cell "myCell" ?view "myLayout")
cellView = geGetEditCellView(window)

;; Just open hierarchy and count unused data:
dbClearUnusedOccShapeColorData(cellview ?dryRun t)  ; for example, returns 5

;; Remove unused data:
dbClearUnusedOccShapeColorData(cellview)  ; should return 5, too

;; Save changes:
dbSave(cellview)
```


© 2020 All Rights Reserved.

dbInstGetLayerShifts

This function is no longer supported. Use dbInstGetLayerShifts2 instead.

© 2020 All Rights Reserved.

dbInstGetLayerShifts2

```
dbInstGetLayerShifts2(
     d_instId
     )
     => l_layerShifts / nil
```

Description

(ICADVM20.1 Only) Returns a list of shifts that are applied to the layers in the master of the specified instance.

The returned list will consist of sublists, each representing layer-shift pairs.

Arguments

d_instId	Specifies the instance ID

© 2020 All Rights Reserved.

Value Returned

l_layerShifts	Returns a list of layer-shift pairs, representing color shifts on the layers specified on the shapes in this instance in the following format: `list(`*g_layer t_shift*`)` where, *g_layer* is the layer number and *t_shift* is one of the following: ■ `noShift`: There is no color shifting on this layer. ■ `shift1Shift`: Shift colors by one (mask1Color to mask2Color, mask2Color to mask3Color, mask3Color to mask1Color). ■ `shift2Shift`: Shift colors by two (mask1Color to mask3Color, mask2Color to mask1Color, mask3Color to mask2Color). ■ `fixedMask1Shift`: Shapes with mask1Color are not changed; shift mask2Color to mask3Color, and mask3Color to mask2Color. ■ `fixedMask2Shift`: Shapes with mask2Color are not changed; shift mask1Color to mask3Color, and mask3Color to mask1Color. ■ `fixedMask3Shift`: Shapes with mask3Color are not changed; shift mask1Color to mask2Color, and mask2Color to mask1Color.
`nil`	There are no layer shifts specified on the instance.

Example

```
shifts = dbInstGetLayerShifts2(inst)
=> shifts = ((20 "shift1Shift") (21 "fixedMask2Shift"))
```

Returns the layer-shift pairs for instance `inst`.

© 2020 All Rights Reserved.

dbInstHasLayerShifts

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

Description

(ICADVM20.1 Only) Returns a Boolean value indicating whether per-layer color shifts are specified on the instance.

Arguments

d_instId	Specifies instance ID

Value Returned

`t`	Returns `t` if per-layer color shifts exist on the specified instance
`nil`	Returns `nil` if per-layer color shifts do not exist on the specified instance

Example

```
hasShift = dbInstHasLayerShifts(instId)
```


© 2020 All Rights Reserved.

dbInstSetLayerShifts

This function is no longer supported. Use dbInstSetLayerShifts2 instead.

© 2020 All Rights Reserved.

dbInstSetLayerShifts2

```
dbInstSetLayerShifts2(
     d_instId
     l_layerShifts
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the given *l_layerShifts* on the specified instance.

All layer shift settings can be applied to layers with three masks. Only `noShift` and `shift1Shift` can be applied to layers with two masks.

Layer shifting cannot be run in the following conditions:

- The *l_layerShifts* argument list has a layer that is not colorable.
- There are too many colorable layers in the technology. Contact Cadence Customer Support for assistance.

Note: You must initialize the layer shift information for a cellview *before* setting them on the instances. To do this, run dbCellViewUpdateLayerShifts(*cv*) where *cv* is the cellview ID. Otherwise, when you run `dbInstSetLayerShifts2` on these instances, `nil` is returned and layer shifts are not set.

Arguments

d_instId	Specifies the instance ID

© 2020 All Rights Reserved.

l_layerShifts	Lists the layer-shift pairs to be set on the specified instance. Each layer-shift pair is represented as `list(`*g_layer t_shift*`)` where, *g_layer* is the layer name or layer number and *t_shift* is one of the following: ■ `noShift`: There is no color shifting on this layer. ■ `shift1Shift`: Shifts colors by one (mask1Color to mask2Color, mask2Color to mask3Color, mask3Color to mask1Color). ■ `shift2Shift`: Shifts colors by two (mask1Color to mask3Color, mask2Color to mask1Color, mask3Color to mask2Color). ■ `fixedMask1Shift`: Shapes with mask1Color are not changed; shifts mask2Color to mask3Color, and mask3Color to mask2Color. ■ `fixedMask2Shift`: Shapes with mask2Color are not changed; shifts mask1Color to mask3Color, and mask3Color to mask1Color. ■ `fixedMask3Shift`: Shapes with mask3Color are not changed; shifts mask1Color to mask2Color, and mask2Color to mask1Color.

Value Returned

`t`	Returns `t` if the master layer shifting information is successfully set
`nil`	Returns `nil` if an error occurs while setting the master layer shifting information

Example

```
shiftVals = list(list(30 "noShift") list(31 "fixedMask2Shift") list("Metal3"
"shift1Shift"))
dbInstSetLayerShifts2(inst shiftVals)
```

Sets the layer-shift pairs for instance `inst` on layer number `30`, layer number `31`, and `Metal3` to `noShift`, `fixedMask2Shift`, and `shift1Shift`, respectively.

© 2020 All Rights Reserved.

dbInstClearLayerShifts

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

Description

(ICADVM20.1 Only) Removes all per-layer color shifting information from the specified instance.

Arguments

d_instId	Specifies instance ID

Value Returned

`t`	Returns `t` if the per-layer color shifting information is successfully removed from the specified instance
`nil`	Returns `nil` if the per-layer color shifting information is not removed from the specified instance

Example

```
dbInstClearLayerShifts(instId)
```


© 2020 All Rights Reserved.

dbIsShapeColoringAllowed

```
dbIsShapeColoringAllowed(
     d_shapeID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Indicates whether the specified shape is of a type that can be colored or not. Only the following shapes can be colored: arcs, lines, donuts, ellipses, pathSegs, paths, rects, and polygons.

Note: This function does not check whether the layer associated with the shape is colorable or not, as defined in the technology database.

Arguments

d_shapeID	Specifies the shape ID

Value Returned

`t`	Returns `t` if the shape can be colored
`nil`	Returns `nil` if the shape cannot be colored

Example

```
dbIsShapeColoringAllowed( dbCreateRect( cvId "M1" "drawing" '((0 0) (100 100)) ) )
=> t
dbIsShapeColoringAllowed( dbCreateDot( cvId "M1" "drawing" '(0 0) ) ) => nil
```


© 2020 All Rights Reserved.

dbIsShapeColored

```
dbIsShapeColored(
     d_shapeID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Indicates whether the specified shape is colored in a color other than `grayColor`.

Arguments

d_shapeID	Specifies the shape ID. **Note:** To check the shape's colorability use dbIsShapeColoringAllowed.

Value Returned

`t`	Returns `t` if the shape is not colored in `grayColor`
`nil`	Returns `nil` if the shape is colored in `grayColor` or the function encounters an error during execution

Example

```
dbIsShapeColored( coloredRect ) => t
dbIsShapeColored( dbCreateRect( cvId "M1" "drawing" '((0 0) (100 100)) ) ) => nil
; newly created shapes are colored in gray color.
```


© 2020 All Rights Reserved.

dbGetShapeColor

```
dbGetShapeColor(
     d_shapeId
     )
     => t_maskColor / nil
```

Description

(ICADVM20.1 Only) Returns the assigned color of the specified shape. By default, all newly created shapes are of the `grayColor` color type, which represents the currently assigned color of the shape.

Arguments

d_shapeId	ID of the shape whose color is to be returned. **Note:** To check the shape's colorability use dbIsShapeColoringAllowed.

Value Returned

t_maskColor	Returns a string representing the color of the shape. The shape color can be one of the following: `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, `"mask4Color"`, `"mask5Color"`, `"mask6Color"`, `"mask7Color"`, or `"mask8Color"`.
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbGetShapeColor( coloredRect ) => "mask2Color"
dbGetShapeColor( dbCreateRect( cvId "M1" "drawing" '((0 0) (100 100)) ) ) =>
"grayColor"   ; newly created shapes are colored in gray color.
```


© 2020 All Rights Reserved.

dbSetShapeColor

```
dbSetShapeColor(
     d_shapeId
     t_maskColor
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Assigns the specified color to the given shape if the color state is unlocked.

If you try to set the color of a non-colorable shape, or if the layer associated with the shape is unbound or does not support coloring, the function results in an error.

Arguments

d_shapeId	Specifies ID of the shape whose color is to be set. **Note:** To check the shape's colorability use dbIsShapeColoringAllowed.
t_maskColor	Color to be set for the shape. Valid values: `grayColor`, `mask<mask number>Color`, where <*mask number*> is 1, 2, and so on, up to the number of masks defined in the technology library for the layer.

Value Returned

`t`	Returns `t` if the function runs successfully.
`nil`	Returns `nil` if there is an error.

Example

```
dbSetShapeColor(coloredRect "mask1Color" ) => t
dbGetShapeColor(coloredRect) => "mask1Color"
```


© 2020 All Rights Reserved.

dbIsShapeColorLocked

```
dbIsShapeColorLocked(
     d_shapeID
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Indicates whether the coloring for the specified shape is locked.

Arguments

d_shapeID	Specifies the Shape ID. **Note:** To check the shape's colorability, use dbIsShapeColoringAllowed.

Value Returned

`t`	Returns `t` if the coloring of the shapeID is locked.
`nil`	Returns `nil` if the coloring of the shapeID is not locked or if there is an error.

Example

```
dbSetShapeColorLocked( coloredRect t ) => t
dbIsShapeColorLocked( coloredRect ) => t
dbIsShapeColorLocked( dbCreateRect( cvId "M1" "drawing" '((0 0) (100 100)) ) ) =>
nil   ; newly created shapes have unlocked coloring
```


© 2020 All Rights Reserved.

dbSetShapeColorLocked

```
dbSetShapeColorLocked(
     d_shapeId
     g_setLocked
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Changes the color state of the shape.

If you try to set the color state of a non-colorable shape, it results in an error. It also results in an error if you set the color state of a shape whose color is `grayColor` because it is inconsistent to lock the coloring of an uncolored shape.

Arguments

d_shapeId	Specifies the shape ID. **Note:** To check the shape's colorability use dbIsShapeColoringAllowed.
g_setLocked	Specifies one of the two boolean values: `t` or `nil`. `t` to change the state to `locked` and `nil` to change the state to `unlocked`.

Value Returned

`t`	Returns `t` if the function runs successfully
`nil`	Returns `nil` if the function encounters an error during execution

Example

```
dbSetShapeColorLocked( coloredRect t ) => t
dbGetShapeColor( coloredRect ) => "mask2Color"

dbSetShapeColorLocked( coloredRect nil ) => t
dbGetShapeColor( coloredRect ) => "mask1Color"
```


© 2020 All Rights Reserved.

dbIsBlockageColored

```
dbIsBlockageColored(
     d_blockageId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Indicates whether the specified blockage is colored in a color other than `grayColor`.

Arguments

d_blockageId	Specifies the blockage ID

Value Returned

`t`	Returns `t` if the specified blockage is not colored in `grayColor`
`nil`	Returns `nil` if the blockage is colored in `grayColor` or there is an error

Example

```
dbIsBlockageColored( coloredBlockage ) => t
dbIsBlockageColored( dbCreateLayerBlockage( cvId "M1" "routing" '((0 0) (100 0)
(100 100)) ) ) => nil   ; newly created blockages are colored in gray color.
```


© 2020 All Rights Reserved.

dbIsBlockageColoringAllowed

```
dbIsBlockageColoringAllowed(
     d_blockageId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Indicates whether the specified blockage is of a blockage type that can be colored. Only layer blockages can be colored.

Note: This function does not check whether the layer associated with the blockage is colorable or not, as defined in the technology database.

Arguments

d_blockageId	Specifies the blockage ID

Value Returned

`t`	Returns `t` if the blockage can be colored
`nil`	Returns `nil` if the blockage cannot be colored or there is an error

Example

```
dbIsBlockageColoringAllowed( dbCreateLayerBlockage( cvId "M1" "routing" '((0 0)
(100 0) (100 100)) ) ) => t
```

```
dbIsBlockageColoringAllowed( dbCreateAreaBlockage( cvId '((0 0) (100 0) (100 100))
) ) => nil
```


© 2020 All Rights Reserved.

dbGetBlockageColor

```
dbGetBlockageColor(
     d_blockageId
     )
     => t_maskColor / nil
```

Description

(ICADVM20.1 Only) Returns the color assigned to the given blockage. The default color type is `grayColor`.

Arguments

d_blockageId	Specifies the blockage ID

Value Returned

t_maskColor	Returns the color assigned to the blockage. Possible values include: `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, `"mask4Color"`, `"mask5Color"`, `"mask6Color"`, `"mask7Color"`, `"mask8Color"`. `"blackColor"`, or `"multiColor"`
`nil`	Returns `nil` if the function encounters an error during execution

Example

```
dbSetBlockageColor( layerBlockage "multiColor" ) => t
dbGetBlockageColor( layerBlockage ) => "multiColor"
```


© 2020 All Rights Reserved.

dbSetBlockageColor

```
dbSetBlockageColor(
     d_blockageID
     t_maskColor
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Assigns color to the specified blockage.

If you try to assign color to the blockage that cannot be colored, results in an error.

Arguments

d_blockageID	Specifies the blockage ID
t_maskColor	Color to be assigned to the blockage. Valid values are: `"grayColor"`, `"blackColor"`, `"multi-Color"`, or `"mask<mask number>Color"`, where *<mask number>* is 1, 2, and so on, up to the number of masks defined in the technology library for the layer.

Value Returned

`t`	Returns `t` if the function runs successfully
`nil`	Returns `nil` if the function encounters an error

Example

```
dbSetBlockageColor( areaBlockage "mask1Color" ) => nil
dbSetBlockageColor( layerBlockage "blackColor" ) => t
dbGetBlockageColor( layerBlockage ) => "blackColor"
```


© 2020 All Rights Reserved.

dbIsViaColorStateLocked

```
dbIsViaColorStateLocked(
     d_viaId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns a Boolean indicating whether the coloring on the via is locked. The default value is `nil`.

Arguments

d_viaId	Specifies the via ID

Value Returned

`t`	Returns `t` if the coloring on the via is locked
`nil`	Returns `nil` if the coloring on the via is not locked or an error occurs

Example

```
;Example 1
dbIsViaColorStateLocked( dbCreateVia( cvId viaDefId '(0 0) "R0" ) )
=> nil

;Example 2
dbSetViaColorStateLocked( via t ) => t
dbIsViaColorStateLocked( via ) => t
```


© 2020 All Rights Reserved.

dbSetViaColorStateLocked

```
dbSetViaColorStateLocked(
     d_viaId
     g_locked
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the color state for the specified via.

Arguments

d_viaId	Specifies the via ID
g_locked	Specifies one of the two boolean values: `t` or `nil`. `t` to change the state to `locked` and `nil` to change the state to `unlocked`.

Value Returned

`t`	Returns `t` if the coloring on the via is successfully set
`nil`	Returns `nil` if the coloring is not set or an error occurs during execution

Example

```
dbSetViaColorStateLocked( via t ) => t
dbIsViaColorStateLocked( via ) => t
dbSetViaCutLayerControl( via "shift1" ) => nil
```


© 2020 All Rights Reserved.

dbIsViaColorStateLayerLocked

```
dbIsViaColorStateLayerLocked(
     d_dbViaId
     t_dbViaLayerType
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Indicates whether the coloring for the specified individual layer in a via is locked.

Arguments

d_dbViaId	Database ID of a via.
t_dbViaLayerType	Name of a valid via layer type. Valid values: `layer1`, `layer2`, `cutLayer`

Value Returned

`t`	Coloring state for the specified individual layer in a via is locked.
`nil`	Coloring state for the specified individual layer in a via is not locked or if there is an error.

Example

```
dbSetViaColorStateLayerLocked(via1t_m2_m1 "layer1" t) => t
dbIsViaColorStateLayerLocked(via1t_m2_m1 "layer1") => t
dbIsViaColorStateLocked(via1t_m2_m1) => t
```

Coloring state for the layer `layer1` in the via `via1t_m2_m1` is locked.

© 2020 All Rights Reserved.

dbSetViaColorStateLayerLocked

```
dbSetViaColorStateLayerLocked(
     d_dbViaId
     t_dbViaLayerType
     g_setLocked
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Locks and unlocks the color state for individual layers in a via.

Arguments

d_dbViaId	Database ID of a via.
t_dbViaLayerType	Name of a valid via layer type.
	Valid values: `layer1`, `layer2`, `cutLayer`
g_setLocked	Specifies the lock state for the specified layer: `t` locks the layer, `nil` unlocks the layer

Value Returned

`t`	Color state for the specified layer in the given via is changed.
`nil`	The operation failed.

Example

```
dbSetViaColorStateLayerLocked(via1t_m2_m1 "layer1" t) => t
dbIsViaColorStateLayerLocked(via1t_m2_m1 "layer1") =>  t
dbIsViaColorStateLocked(via1t_m2_m1) => t
```

Color state for the layer `layer1` in the via `via1t_m2_m1` is changed to locked.

© 2020 All Rights Reserved.

dbGetViaLayerNumColorMasks

```
dbGetViaLayerNumColorMasks(
     d_viaId
     t_viaLayerType
     )
     => t_layerName / nil
```

Description

(ICADVM20.1 Only) Returns the number of supported color masks for specified layer of the given via.

Arguments

d_viaId	Specifies the via ID.
t_viaLayerType	Specifies the via layer type. Valid values: `layer1`, `layer2`, `cutLayer`

Value Returned

n_numMasks	Returns the number of supported color masks for the specified layer of the via.
`nil`	Returns `nil` if an error occurs (for example, invalid via ID or layer type).

Example

```
cv = dbOpenCellViewByType( "tech" "design" "layout" "maskLayout" "w" )
viaDef_M2_M1 = techFindViaDefByName( tf "M2_M1" )
via1t_m2_m1 = dbCreateVia( cv viaDef_M2_M1 10:10 "R0" )
techSetLayerNumColorMasks( tf "metal1" 3 )
techSetLayerNumColorMasks( tf "metal2" 8 )

dbGetViaLayerNumColorMasks( via1t_m2_m1 "layer1" )
=> 8

dbGetViaLayerNumColorMasks( via1t_m2_m1 "layer2" )
=> 3

dbGetViaLayer( via1t_m2_m1 "cutLayer" )
```


© 2020 All Rights Reserved.

dbGetViaLayer1Control

```
dbGetViaLayer1Control(
     d_viaId
     )
     => t_colorControl / nil
```

Description

(ICADVM20.1 Only) Returns the color control assigned to layer1 of the via. Default value for color control is `noShift`.

Arguments

d_viaId	Specifies the via ID

Value Returned

t_colorControl	Returns the color control assigned to the layer2 of the via. Valid values: `noColor`, `noShift`, `shift1`, `shift2`,`shift2`,`shift3`,`shift4`,`shift5`,`shift6`,or `shift7`.
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbGetViaLayer1Control( dbCreateVia( cvId viaDefId '(0 0) "R0" ) )
=> "noShift"

dbGetViaLayer1Control( grayVia )
=> "noColor"
```


© 2020 All Rights Reserved.

dbGetViaLayer2Control

```
dbGetViaLayer2Control(
     d_viaId
     )
     => t_colorControl / nil
```

Description

(ICADVM20.1 Only) Returns the color control assigned to layer 2 of the via. Default value for color control is `noShift`.

Arguments

d_viaId	Specifies the via ID

Value Returned

t_colorControl	Returns the color control assigned to the layer2 of the via. Valid values: `noColor`, `noShift`, `shift1`, `shift2`,`shift2`,`shift3`,`shift4`,`shift5`,`shift6`,or `shift7`.
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbGetViaLayer2Control( dbCreateVia( cvId viaDefId '(0 0) "R0" ) )
=> "noShift"

dbGetViaLayer2Control( grayVia )
=> "noColor"
```


© 2020 All Rights Reserved.

dbGetViaCutLayerControl

```
dbGetViaCutLayerControl(
     d_viaId
     )
     => t_colorControl / nil
```

Description

(ICADVM20.1 Only) Returns the color control assigned to the cut layer of the via. Default color control is `noShift`.

Arguments

`d_viaId`	Specifies the via ID

Value Returned

`t_colorControl`	Returns the color control assigned to the cut layer of the via. Valid values: `noColor`, `noShift`, `shift1`, `shift2`,`shift2`,`shift3`,`shift4`,`shift5`,`shift6`,or `shift7`.
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbGetViaCutLayerControl( dbCreateVia( cvId viaDefId '(0 0) "R0" ) ) => "noShift"
dbGetViaCutLayerControl( grayVia ) => "noColor"
```


© 2020 All Rights Reserved.

dbGetViaLayerControl

```
dbGetViaLayerControl(
     d_dbViaId
     t_dbViaLayerType
     )
     => t_colorControl / nil
```

Description

(ICADVM20.1 Only) Returns the color control assigned to the specified layer in the given via.

Arguments

d_dbViaId	Database ID of a via.
t_dbViaLayerType	Name of a valid via layer type. Valid values: `layer1`, `layer2`, `cutLayer`

Value Returned

t_colorControl	Returns the color control assigned to the specified layer in the given via. Valid values: `noColor`, `noShift`, `shift1`, `shift2`,`shift2`,`shift3`,`shift4`,`shift5`,`shift6`,or `shift7`.
`nil`	The operation failed.

Example

```
cv = dbOpenCellViewByType("tech" "design" "layout" "maskLayout" "w")
viaDef_M2_M1 = techFindViaDefByName(tf "M2_M1")
via1t_m2_m1 = dbCreateVia(cv viaDef_M2_M1 10:10 "R0")

dbSetViaLayerControl(via1t_m2_m1 "layer2" "shift2")
=> t
```

Color control `shift2` is set on the layer `layer2` in the via `via1t_m2_m1`.

```
dbGetViaLayer2Control( via1t_m2_m1 )
=> "shift2"
```


© 2020 All Rights Reserved.

Returns color control for `layer2` in the via `via1t_m2_m1` as `shift2`, using the function `dbGetViaLayer2Control`.

```
dbGetViaLayerControl( via1t_m2_m1 "layer2")
=> "shift2"
```

Returns color control for `layer2` in the via `via1t_m2_m1` as `shift2`, using the function `dbGetViaLayerControl` by specifying the name of the layer as one of the arguments.

© 2020 All Rights Reserved.

dbSetViaLayer1Control

```
dbSetViaLayer1Control(
     d_viaId
     t_colorControl
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the color control for layer1 shapes of the via master.

It is an error to assign color control to the layer without allowed colored masks or if via color state is locked.

Arguments

d_viaId	Specifies the via ID
t_colorControl	Specifies the string representing color control to be assigned to the layer1 shapes of the via master. Valid values: `noColor`, `noShift`, `shift1`, and so on, up to `shift`*x*, where *x* is an integer equal to (number of masks for the via layer - `1`). For example, the valid values for a 3-mask via layer are: `noColor`, `noShift`, `shift1`, and `shift2`.

Value Returned

`t`	Returns `t` if the color controls are successfully set
`nil`	Returns `nil` if the color controls are not set or an error occurs during execution

Example

```
;Example 1
dbSetViaLayer1Control( via "shift2" )
=> t
dbSetViaColorStateLocked( via t )
=> t
dbSetViaLayer1Control( via "shift1" )
=> nil
```


© 2020 All Rights Reserved.

```
; Returns an error / nil because the color state is locked
```


© 2020 All Rights Reserved.

dbSetViaLayer2Control

```
dbSetViaLayer2Control(
     d_viaId
     t_colorControl
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the color control for layer2 shapes of the via master.

It is an error to assign color control to the layer without allowed colored masks or if via color state is locked.

Arguments

d_viaId	Specifies the via ID
t_colorControl	Specifies the string representing color control to be assigned to the layer1 shapes of the via master. Valid values: `noColor`, `noShift`, `shift1`, and so on, up to `shift`*x*, where *x* is an integer equal to (number of masks for the via layer - `1`). For example, the valid values for a 3-mask via layer are: `noColor`, `noShift`, `shift1`, and `shift2`.

Value Returned

`t`	Returns `t` if the color controls are successfully set
`nil`	Returns `nil` if the color controls are not set or an error occurs during execution

Example

```
dbSetViaLayer2Control( via "shift2" )
=> t
dbSetViaColorStateLocked( via t )
=> t
dbSetViaLayer2Control( via "shift1" )
=> nil
; Returns an error / nil because the color state is locked
```


© 2020 All Rights Reserved.

dbSetViaCutLayerControl

```
dbSetViaCutLayerControl(
     d_viaId
     t_colorControl
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the color control for the cut layer shapes of the via master.

It is an error to assign color control to the layer without allowed colored masks or if via color state is locked.

Arguments

d_viaId	Specifies the via ID
t_colorControl	Specifies the string representing color control to be assigned to the cut layer shapes of the via master. Valid values: `noColor`, `noShift`, `shift1`, and so on, up to `shift`*x*, where *x* is an integer equal to (number of masks for the via layer – `1`). For example, the valid values for a 3-mask via layer are: `noColor`, `noShift`, `shift1`, and `shift2`.

Value Returned

`t`	Returns t is the color control is successfully set
`nil`	Returns nil is the color control is not set or an error occurs during execution

Example

```
dbSetViaCutLayerControl( via "shift2" ) => t
dbSetViaColorStateLocked( via t ) => t
dbSetViaCutLayerControl( via "shift1" ) => nil ; returns nil because the color state
is locked
```


© 2020 All Rights Reserved.

dbSetViaLayerControl

```
dbSetViaLayerControl(
     d_dbViaId
     t_dbViaLayerType
     t_colorControl
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the color control for the specified layer in the given via.

Arguments

d_dbViaId	Database ID of a via.
t_dbViaLayerType	Name of a valid via layer type. Valid values: `layer1`, `layer2`, `cutLayer`
t_colorControl	Color control that can be assigned to the specified layer in the given via. Valid values: `noColor`, `noShift`, `shift1`, and so on, up to `shiftx`, where *x* is an integer equal to (number of masks for the via layer - `1`). For example, the valid values for a 3-mask via layer are: `noColor`, `noShift`, `shift1`, and `shift2`.

Value Returned

`t`	Color controls for the specified layer in the given via are successfully changed.
`nil`	The operation failed.

Example

```
cv = dbOpenCellViewByType("tech" "design" "layout" "maskLayout" "w")
viaDef_M2_M1 = techFindViaDefByName(tf "M2_M1")
via1t_m2_m1 = dbCreateVia(cv viaDef_M2_M1 10:10 "R0")

dbSetViaLayerControl(via1t_m2_m1 "layer2" "shift2")
=> t
```


© 2020 All Rights Reserved.

```
dbSetViaLayerControl(via1t_m2_m1 "layer1" "shift1")
=> t
dbSetViaLayerControl(via1t_m2_m1 "cutLayer" "shift1")
=> t
```

In this example, the following color controls are assigned to the specified layers in the via `via1t_m2_m1`:

- Color control `shift2` to the layer `layer2`.
- Color control `shift1` to the layer `layer1`.
- Color control `shift1` to the layer `cutLayer`.

© 2020 All Rights Reserved.

dbSetViaColorInfo

```
dbSetViaColorInfo(
     d_dbViaId
     l_layer1ControlInfo
     l_layer2ControlInfo
     l_cutLayerControlInfo
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Updates the color control and color state for the three via layers (via layer1, via layer2, and cut layer) of the given via.

A warning message is displayed with a return value of `nil` for conditions such as the following:

- The color information argument is not set to `nil` for a via layer that is not colorable.
- The color information is not valid (for example, the color control value is `noColor` and the lock state is `t`).

© 2020 All Rights Reserved.

Arguments

d_dbViaId Database ID of the via.

l_layer1ColorInfo `list( t_colorControl g_locked ) | nil`

Specifies the color information for via layer1 as a list of the following:

- *t_colorControl*

  Color control to assign to the via layer.

  Valid values: `noColor`, `noShift`, `shift1`, and so on, up to `shiftx`, where *x* is an integer equal to the number of masks for the via layer - `1`. For example, the valid values for a 3-mask via layer are: `noColor`, `noShift`, `shift1`, and `shift2`.

- *g_locked*

  Specifies one of two Boolean values: `t` or `nil`, where `t` sets the via layer state to locked, and `nil` sets the state to unlocked.

Use `nil` for no change to the via layer.

l_layer2ColorInfo `list( t_colorControl g_locked ) | nil`

Specifies the color information for via layer2, in the same format as *l_layer1ColorInfo.*

l_cutLayerColorInfo

`list( t_colorControl g_locked ) | nil`

Specifies the color information for the via cut layer, in the same format as *l_layer1ColorInfo.*

Value Returned

`t` Color information for the via was successfully changed.

`nil` Color information for the via was not changed or an error occurred.

Example

© 2020 All Rights Reserved.

```
; Set environment.
w = dbOpenCellViewByType("lib" "design" "layout" "maskLayout" "w")
cv = w->cellView
tf = techGetTechFile(cv)
techCreateLayer(tf 1 "M1")
techCreateLayer(tf 2 "M2")
techCreateLayer(tf 3 "V0")
; Make the metal layers colorable.
techSetLayerNumColorMasks(tf "M1" 2)
techSetLayerNumColorMasks(tf "M2" 2)
vd = techCreateStdViaDef(tf "std" "M1" "M2"
                         list("V0" 1 1 1) list(1 1 '(1 1))
                         '(1 1) '(1 1) '(1 1) '(1 1) '(1 1))
v1 = dbCreateVia(cv vd 5:0 "R0")
v2 = dbCreateVia(cv vd 10:0 "R0")
; Set color information for v1 and v2.
dbSetViaColorInfo(v1 list("noColor" nil) list("shift1" t) nil)
dbSetViaColorInfo(v2 list("noShift" t) nil nil)
; Check results.
dbGetViaLayerControl(v1 "layer1")             ;  => "noColor"
dbIsViaColorStateLayerLocked(v1 "layer1")     ;  => nil
dbGetViaLayerControl(v1 "layer2")             ;  => "shift1"
dbIsViaColorStateLayerLocked(v1 "layer2")     ;  => t
dbGetViaLayerControl(v1 "cutLayer")           ;  => "noShift"
dbIsViaColorStateLayerLocked(v1 "cutLayer")   ;  => nil

dbGetViaLayerControl(v2 "layer1")             ;  => "noShift"
dbIsViaColorStateLayerLocked(v2 "layer1")     ;  => t
dbGetViaLayerControl(v2 "layer2")             ;  => "noColor"
dbIsViaColorStateLayerLocked(v2 "layer2")     ;  => nil
dbGetViaLayerControl(v2 "cutLayer")           ;  => "noShift"
dbIsViaColorStateLayerLocked(v2 "cutLayer")   ;  => nil
```


© 2020 All Rights Reserved.

dbIsTrackPatternColored

```
dbIsTrackPatternColored(
     d_trackPatternId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns a Boolean value that indicates whether the first track of the specified track pattern is colored in a color other than gray.

(ICADVM20.1 Only) To color the track pattern, use dbSetTrackPatternFirstTrackColor or dbSetTrackPatternColorAlternating.

Arguments

d_trackPatternId	Specifies the track pattern ID

Value Returned

`t`	Returns `t` if the track pattern is colored
`nil`	Returns `nil` if the track pattern is not colored or an error occurs during execution

Examples

```
dbIsTrackPatternColored( dbCreateTrackPattern( cvId t 100 10 10 "M1") ) => nil
```

Creates a track pattern, then checks for coloring on the track pattern. Newly created track patterns are not colored.

```
tpID = dbCreateTrackPattern( cvId t 100 10 10 "M1") )
dbSetTrackPatternFirstTrackColor( tpID "mask2Color" )
dbIsTrackPatternColored( tpID ) => t
```

Creates a track pattern, then sets the first track color to `mask2Color`, which makes the track pattern colored.

© 2020 All Rights Reserved.

dbIsTrackPatternColorAlternating

```
dbIsTrackPatternColorAlternating(
     d_trackPatternId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns a Boolean value indicating whether the coloring for the specified track pattern is alternating.

Arguments

d_trackPatternId	Specifies the track pattern ID

Value Returned

`t`	Returns `t` if the track pattern coloring is alternating. **Note:** Newly created track patterns are alternating by default, even though they are not colored.
`nil`	Returns `nil` if the track pattern coloring is not alternating or if there is an error

Example

```
dbIsTrackPatternColorAlternating( coloredAltTrackPattern ) => t
dbIsTrackPatternColorAlternating( coloredAllSameTrackPattern ) => nil
dbIsTrackPatternColorAlternating( dbCreateTrackPattern( cvId t 100 10 10 "M1") )
=> t
```


© 2020 All Rights Reserved.

dbIsTrackPatternColoringAllowed

```
dbIsTrackPatternColoringAllowed(
     d_trackPatternId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns a Boolean indicating whether the specified track pattern can be colored. Only track patterns with specified track layer can be colored.

Arguments

d_trackPatternId	Specifies the track pattern ID that you want to check

Value Returned

`t`	Returns `t` if the track pattern can be colored
`nil`	Returns `nil` if the track pattern cannot be colored or an error occurs during execution

Example

```
dbIsTrackPatternColoringAllowed( dbCreateTrackPattern( cvId t 100 10 10 "M1") ) =>
t
dbIsTrackPatternColoringAllowed( dbCreateTrackPattern( cvId t 100 10 10) ) => nil
```


© 2020 All Rights Reserved.

dbGetTrackPatternFirstTrackColor

```
dbGetTrackPatternFirstTrackColor(
     d_trackPatternId
     )
     => t_maskColor / nil
```

Description

(ICADVM20.1 Only) Returns the color assigned to the first track of the specified track pattern. Default color type is `grayColor`.

Arguments

`d_trackPatternId`	Specifies the track pattern ID

Value Returned

`t_maskColor`	Returns a string representing the color assigned to the first track of the specified track pattern. Possible values are `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`.
`nil`	Returns `nil` if the color is cannot be returned or an error occurs during execution.

Example

```
dbSetTrackPatternFirstTrackColor( layerBlockage "mask1Color" ) => t
dbGetTrackPatternFirstTrackColor( layerBlockage ) => "mask1Color"
```


© 2020 All Rights Reserved.

dbSetTrackPatternFirstTrackColor

```
dbSetTrackPatternFirstTrackColor(
     d_trackPatternId
     t_maskColor
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Assigns a color to the first track of the specified track pattern.

It is an error to assign color to the first track of the track pattern that cannot be colored.

Arguments

d_trackPatternId	Specifies the track pattern ID
t_maskColor	Specifies a string representing color to be assigned to the first track of the specified track pattern. Possible values are `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`

Value Returned

`t`	Returns `t` if the color is successfully assigned to the first track
`nil`	Returns `nil` if the color is not assigned to the first track or an error occurs during execution

Example

```
dbSetTrackPatternFirstTrackColor( trackPatternWithoutTrackLayer "mask1Color" ) =>
nil
dbSetTrackPatternFirstTrackColor( trackPattern "mask1Color" ) => t
dbGetTrackPatternFirstTrackColor( trackPattern ) => "mask1Color"
```


© 2020 All Rights Reserved.

dbSetTrackPatternColorAlternating

```
dbSetTrackPatternColorAlternating(
     d_trackPatternId
     g_isAlternating
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the coloring type of the track pattern.

Arguments

d_trackPatternId	Specifies the track pattern ID
g_isAlternating	Specifies whether the same color should be used for all tracks in the pattern (`nil`) or the track pattern should cycle through the colors for the layer (`t`). The default is `t`.

Value Returned

`t`	Returns `t` if the specified coloring type is successfully set
`nil`	Returns `nil` if the coloring type is not set

Example

```
dbSetTrackPatternColorAlternating( trackPattern nil ) => t
dbIsTrackPatternColorAlternating( trackPattern ) => nil
```


© 2020 All Rights Reserved.

dbCreateSameMaskGroup

```
dbCreateSameMaskGroup(
     d_cellView
     t_groupName
     )
     => d_group / nil
```

Description

(ICADVM20.1 Only) Creates a same-mask, color shape group (*d_group*) in the specified cellview with the given name. The name must be unique amongst all same-mask groups in the cellview. You can only add shapes to the resulting group.

A same-mask, color shape group allows the user to specify a group of shapes on a layer that must be on the same color mask.

Note: The same mask groups created by the `dbCreateSameMaskGroup` function are protected by default. They cannot be returned in collections (for example, `cv~>groups`).

Arguments

d_cellView	Specifies the cellview ID
t_groupName	Specifies the name of a group that is to be created

Value Returned

d_group	Returns the ID of the diff-mask group created
`nil`	Returns `nil` if an error occurs during execution

Example

```
groupMC1 = dbCreateSameMaskGroup( cvId "MC1Group" )
dbAddObjectToGroup( groupMC1 rectM1 )
dbAddObjectToGroup( groupMC1 circleM3 )
```

Related Information

dbGetSameMaskDiffMaskGroups

© 2020 All Rights Reserved.

dbCreateDiffMaskGroup

```
dbCreateDiffMaskGroup(
     d_cellView
     t_groupName
     d_sameMaskGroup1
     d_sameMaskGroup2
     )
     => d_group / nil
```

Description

(ICADVM20.1 Only) Creates a diff-mask, color shape group (`d_group`) in the specified cellview with a pair of same-mask color shape groups and with the given name. The name must be unique amongst all diff-mask groups in the cellview.

A diff-mask, color shape group allows you to specify that the two same-mask color shapes must be on different color masks.

Arguments

d_cellView	Specifies the cellview ID
t_groupName	Specifies the name of a group that is to be created
d_sameMaskGroup1	Specifies the first group from the pair of groups of the shapes (of different colors)
d_sameMaskGroup2	Specifies the second group from the pair of groups of the shapes (of different colors)

Value Returned

d_group	Returns the ID of the diff-mask group created
`nil`	Returns `nil` if an error occurs during execution

Example

```
groupMC1 = dbCreateSameMaskGroup( cvId "MC1Group" )
dbAddObjectToGroup( groupMC1 rectM1 )
dbAddObjectToGroup( groupMC1 circleM3 )

groupMC2 = dbCreateSameMaskGroup( cvId "MC2Group" )
```


© 2020 All Rights Reserved.

```
dbAddObjectToGroup( groupMC2 rectM2 )

groupDMC = dbCreateDiffMaskGroup( cvId "DMCGroup" groupMC1 groupMC2 )

groupMC3 = dbCreateSameMaskGroup( cvId "MC3Group" )
dbAddObjectToGroup( groupMC3 polygonM1 )

dbAddObjectToGroup( groupDMC groupMC3 )
```


© 2020 All Rights Reserved.

dbGetSameMaskDiffMaskGroups

```
dbGetSameMaskDiffMaskGroups(
     d_sameMaskGroupId
     )
     => l_listofGroups / nil
```

Description

(ICADVM20.1 Only) Accepts an object with the same mask color group as a parameter and returns a list of all groups of type 'diffMaskColor'.

Arguments

d_sameMaskGroupId	An object of the same mask color group

Value Returned

l_listofGroups	Returns a list of all groups of type 'diffMaskColor'
`nil`	Returns `nil` if no groups of type 'diffMaskColor' exist or error occurs during execution

Example

```
dbGetSameMaskDiffMaskGroups(groupId)
=> (db:0x32fadc1c)
```


© 2020 All Rights Reserved.

dbGetShapeSameMaskGroups

```
dbGetShapeSameMaskGroups(
     d_shapeId
     )
     => l_listofGroups / nil
```

Description

(ICADVM20.1 Only) Accepts the object of the shape for which you want to retrieve a list of all groups of type 'sameMaskColor'.

Arguments

d_shapeId	Specifies the shape ID.

Value Returned

l_listofGroups	Returns a list of all groups of type 'sameMaskColor'
`nil`	Returns `nil` if no groups of type 'sameMaskColor' exist or error occurs during execution

Example

```
dbGetShapeSameMaskGroups(rectId)
=> (db:0x32fadc1c)
```


© 2020 All Rights Reserved.

dbColorShapeQuery

```
dbColorShapeQuery(
     d_cellViewId
     l_layerPurposePair
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => l_shapeColorPairs / nil
```

Description

(ICADVM20.1 Only) Returns a list of all shapes and their effective color, where the bounding boxes of the shapes overlap the region specified by *l_bBox*.

Note: The effective color of a shape is the color after applying any hierarchical shifting that may be specified at different levels of the design hierarchy. This effective color may not be the color assigned to the shape.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

© 2020 All Rights Reserved.

Arguments

d_cellviewId	Specifies the cellview ID
l_layerPurposePair	Specifies the layer-purpose pair
l_bBox	Specifies the region that should overlap bounding boxes of all returned shapes
x_startLevel	Specifies the level of the design hierarchy in which to start the querying process. If the startLevel argument is not specified, the start level defaults to 0.
x_stopLevel	Specifies the level of the design hierarchy in which to stop the querying process. If the stopLevel argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to x_stopLevel if it encounters a cellview which is not currently loaded.
n_filterSize	Specifies the shapes filter size. The query produces only shapes with a size larger than the filterSize. If no filterSize argument is specified, the blockage filter size defaults to 0.
l_transform	Describes the initial transformation to be used during the region querying process, such as offset, rotation, and/or magnification. If no transform argument is specified, the initial formation defaults to offset (0,0),and orientation R0.

Value Returned

l_shapeColorPairs	Returns a list of pairs (shape ID, shape color)
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbColorShapeQuery( cvId '("M1" "drawing") '((0 0) (100 100)) )
    => ((sh1Id "mask1Color") (sh2Id "grayColor"))
```


© 2020 All Rights Reserved.

dbGetIntegrationColorModel

```
dbGetIntegrationColorModel(
     d_cellViewId
     )
     => t_integType / nil
```

Description

(ICADVM20.1 Only) Returns a string value of the *integration color model* constraint for the specified cellview. Possible values are `any` or `locked`.

Arguments

`d_cellViewId`	Specifies the cellview ID

Value Returned

`t_integType`	Returns a string value representing the integration color model constraint for specified cellview
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbGetIntegrationColorModel( cvId )
=> "locked"
```


© 2020 All Rights Reserved.

dbGetColorModel

```
dbGetColorModel(
     d_cellViewId
     )
     => t_type / nil
```

Description

(ICADVM20.1 Only) Returns a string value of the *color model* constraint for the specified cellview. Possible values are `any`, `layerBased`, or `locked`.

Arguments

d_cellViewId	Specifies the cellview ID

Value Returned

t_type	Returns a string value representing the color model constraint for specified cellview
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbSetColorModel( cvId "layerBased" ) ; setting the color model constraint
=> t
dbGetColorModel( cvId ) ; retrieving the color model constraint
=> "layerBased"
```


© 2020 All Rights Reserved.

dbSetIntegrationColorModel

```
dbSetIntegrationColorModel(
     d_cellViewId
     t_integType
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the value of the *integration color model* constraint for the specified cellview. Possible values are `any` or `locked`.

Arguments

d_cellViewId	Specifies the cellview ID
t_integType	Specifies the value of the integration *color model* constraint you want to set

Value Returned

`t`	Returns `t` if the value is successfully set
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbSetIntegrationColorModel( cvId "any" ) => t

dbSetIntegrationColorModel( cvId "layerBased" ) => nil   ; no "layerBased"
integration color model
```


© 2020 All Rights Reserved.

dbSetColorModel

```
dbSetColorModel(
     d_cellViewId
     t_type
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the value of the *color model* constraint for the specified cellview. It specifies how you want the contents of the block to be color-shifted. Possible values are `any`, `layerBased`, or `locked`.

Arguments

d_cellViewId	Specifies the cellview ID
t_type	Specifies the value of the *color model* constraint you want to set

Value Returned

`t`	Returns `t` if the value is successfully set
`nil`	Returns `nil` if an error occurs during execution

Example

```
dbSetColorModel( cvId "layerBased" )
=> t
```


© 2020 All Rights Reserved.

dbRemoveLayerColorData

```
dbRemoveLayerColorData(
     d_cellViewId
     t_layerName
     g_locked
     [ g_via g_hcl g_instColorShift g_blockage g_trackPattern ]
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Removes the color data from the specified layer in the given cellview. By default, all color data is removed. You can optionally preserve the via color data, hierarchical color locks, layer shift information, blockage color data, and track pattern color data.

© 2020 All Rights Reserved.

Arguments

d_cellViewId	ID of the cellview.
t_layerName	Name of the layer.
g_locked	If `t`, removes the layer color data regardless of the locked color state. If `nil`, the locked shapes and vias on that layer are not affected.
g_via	(Default) If `t`, removes the via color data. If `nil`, the via color data is not affected.
g_hcl	(Default) If `t`, removes the hierarchical color locks. If `nil`, the hierarchical color locks are not affected.
g_instColorShift	(Default) If `t`, removes the layer shift information for the cell-view. If `nil`, the layer shift information for the cellview is not affected.
g_blockage	(Default) If `t`, removes the blockage color data. If `nil`, the blockage color data is not affected.
g_trackPattern	(Default) If `t`, removes the track pattern color data. If `nil`, the track pattern color data is not affected.

Value Returned

`t`	Returns `t` if the color data is successfully removed.
`nil`	Returns `nil` in case of a failure.

Example

```
dbRemoveLayerColorData( cv "M1" t )
> t
```

Removes all the color data from the `M1` layer of the `cv` cellview.

```
dbRemoveLayerColorData( cv "M1" t t t t t nil )
```

Removes all the color data from the `M1` layer of the `cv` cellview, except the track pattern color data.

© 2020 All Rights Reserved.

dbColorShapeQuery2

```
dbColorShapeQuery2(
     d_cellViewId
     l_layerPurposePair
     l_bBox
     [ x_startLevel ]
     [ x_stopLevel ]
     [ n_filterSize ]
     [ l_transform ]
     )
     => l_occShapesWithColorAndState / nil
```

Description

(ICADVM20.1 Only) Returns a list of shapes and occurrence shapes within a given region, and the coloring information for each shape. The coloring information includes the color, whether the color is locked, and whether the coloring is due to the color attribute or hierarchical color locking on the shape.

© 2020 All Rights Reserved.

Arguments

d_cellViewId	ID of the cellview.
l_layerPurposePair	List of layer-purpose pairs.
l_bBox	List of coordinates representing a bounding box.
x_startLevel	The start hierarchy level for the query.
x_stopLevel	The stop hierarchy level for the query.
n_filterSize	The filter size for shapes.
l_transform	The transform applied on the query.

© 2020 All Rights Reserved.

Value Returned

l_occShapesWithColorAndState

((*l_occShape t_color b_state b_isHCL*)...)
A list of elements.

- *l_occShape*
  is
  list(*l_hierPath d_shapeId*)
  The hierarchical path to a shape.

  where,

  - *l_hierPath*
    is
    list((*d_refId* | *l_mosaicElem*)…)

    where,

    - *d_refId*
      The database ID of the non-mosaic instance or via.

    - *l_mosaicElem*
      is
      list(*d_mosaicId x_row x_col*)
      Represents a mosaic element in the hierarchy path.

      where,

      d_mosaicId
      The database ID of the mosaic instance.

      x_row
      Row in which *d_shapeId* occurs inside this mosaic instance.

      x_col
      Column in which *d_shapeId* occurs inside this mosaic instance.

- *d_shapeId*
  The database ID of the shape.

© 2020 All Rights Reserved.

- *t_color*
  Name of the color mask (`"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, `"mask4Color"`, `"mask5Color"`, `"mask6Color"`, `"mask7Color"`, or `"mask8Color"`).
- *b_state*
  A Boolean value that determines whether the color of the shape is locked.
- *b_isHCL*
  A Boolean value that specifies whether the shape color is due to color attribute (`nil`) or hierarchical color locking (`t`).

`nil` Returned in case of failure.

Examples

Example 1

```
dbColorShapeQuery2(cv list("Metal1" "drawing") css()~>bBox)
((((db:0x2edfd01a) db:0x56190228) "grayColor" nil nil)
 (((db:0x2edfd01a) db:0x5619022f) "mask1Color" t nil)
 (((db:0x2edfd01a) db:0x56190236) "mask2Color" t t)
)
```

Returns the list of `Metal1:drawing` shapes in the selected Pcell along with the information for each shape about its color, locked state, and whether the color is due to color attribute or hierarchical color locking (HCL). In the figure below, three shapes are reported. The first shape is not colored, the second is locked on `mask1Color` but not due to HCL, and the third is locked on `mask2Color` due to HCL.

© 2020 All Rights Reserved.

Example 2

```
(defun asNested (l)
   (if (cdr l) then
      (list (car l) (asNested (cdr l)))
   else
      l
   )  ; if
)  ; defun

occShapes = dbColorShapeQuery2( geGetEditCellView()
                                list( "M1" )
                                geGetEditCellView()~>bBox )

foreach(occShape occShapes
   hierPathTransform = dbGetInstTransform( asNested( caar( occShape ) ) )
   printf(" -- %L -- transform : %L \n" car(occShape) hierPathTransform )
); foreach
```

Converts the hierarchical path output from `dbColorShapeQuery2` using the **asNested** function to the format accepted by `dbGetInstTransform`. The following is sample output for this example:

```
 -- ((db:0x18c5e31a db:0x18c5de1a) db:0x18c5df9c) -- transform : ((1.9 2.0) "R0"
1.0)
 -- ((db:0x18c5d39a) db:0x18c5df9c) -- transform : ((1.9 2.04) "R0" 1.0)
```


© 2020 All Rights Reserved.

dbGetColoredOccShapes

```
dbGetColoredOccShapes(
     d_refId
     [ b_lockedOnly ]
     )
     => l_occShapesWithColorAndState / nil
```

Description

(ICADVM20.1 Only) Generates a list of all the occurrence shapes, associated with the specified instance or via, that were colored due to hierarchical color locking. If the optional argument *b_lockedOnly* is `t`, only the locked colored occurrence shapes are included.

Note: This query can be used by the abutment-aware PDK codes.

Arguments

d_refId	ID of an instance or a via.
b_lockedOnly	A Boolean value that determines whether only the locked colored occurrence shapes are returned. When set to `t`, returns only the locked colored occurrence shapes. When set to `nil`, returns all the colored occurrence shapes. Valid values are `t` and `nil`. The default value is `nil`.

© 2020 All Rights Reserved.

Value Returned

l_occShapesWithColorAndState

((*l_occShape t_color b_state*)...)
A list of elements.

- *l_occShape*
  is
  list(*l_hierPath d_shapeId*)
  The hierarchical path to a shape.

  where,

  - *l_hierPath*
    is
    list((*d_refId* | *l_mosaicElem*)…)

    where,

    - *d_refId*
      The database ID of the non-mosaic instance or via.

    - *l_mosaicElem*
      is
      list(*d_mosaicId x_row x_col*)
      Represents a mosaic element in the hierarchy path.

      where,

      d_mosaicId
      The database ID of the mosaic instance.

      x_row
      Row in which *d_shapeId* occurs inside this mosaic instance.

      x_col
      Column in which *d_shapeId* occurs inside this mosaic instance.

© 2020 All Rights Reserved.

- *d_shapeId*
  The database ID of the shape.
- *t_color*
  Name of the color mask ("mask1Color", "mask2Color", or "mask3Color", "mask4Color", "mask5Color", "mask6Color", "mask7Color", or "mask8Color").
- *b_state*
  A Boolean value that determines whether the color of the shape is locked.

`nil` Returned in case of failure and when there are no colored occurrence shapes that meet the criteria and are a result of hierarchical color locking.

Examples

Example 1

```
dbGetColoredOccShapes(css())
((((db:0x4d715a9a) db:0x25c6a81f) "mask1Color" t)
 (((db:0x4d715a9a) db:0x25c6a824) "mask1Color" t)
 (((db:0x4d715a9a) db:0x25c6a947) "mask2Color" t)
 (((db:0x4d715a9a) db:0x25c6a84c) "mask2Color" t)
)
```

Queries the current Pcell in the figure below. Returns four colored shapes found: two of the terminals have `mask2Color` from hierarchical color locking while the other two have `mask1Color` from hierarchical color locking.

Example 2

```
dbGetColoredOccShapes(css())
nil
```


© 2020 All Rights Reserved.

In the figure below, none of the terminals is colored, so the query returns `nil`.

Example 3

```
dbGetColoredOccShapes(css())
nil
```

In the figure below, the middle terminal is locked to `mask1Color` through the CDF parameter, not by hierarchical color locking, so it is not reported when the function is run.

© 2020
All Rights Reserved.

dbGetShapeEffectiveColor

```
dbGetShapeEffectiveColor(
     l_occShape
     )
     => l_colorAndColorState / nil
```

Description

(ICADVM20.1 Only) Implements a query for individual colored shapes. This function takes an occurrence shape (a list consisting of the absolute hierarchical path and the master shape) as an argument. It returns the color of the shape and two Boolean values that indicate whether the color state is locked and whether it was colored by hierarchical color locking.

Note: This function cannot be used for flat (non-hierarchical) shapes. Use dbGetShapeColor and dbIsShapeColorLocked for non-hierarchical shapes.

To get a list of shapes and occurrence shapes within a given region, and the coloring information for each shape, see dbColorShapeQuery2.

© 2020 All Rights Reserved.

Arguments

l_occShape `list(l_hierPath d_shapeId)`

where,

- *l_hierPath*
  is
  `list((d_refId | l_mosaicElem)…)`

  where,

  - *d_refId*
    The database ID of the non-mosaic instance or via.
  - *l_mosaicElem*
    is
    `list(d_mosaicId x_row x_col)`
    Represents a mosaic element in the hierarchy path.

    where,

    - *d_mosaicId*
      The database ID of the mosaic instance.
    - *x_row*
      Row in which *d_shapeId* occurs inside this mosaic instance.
    - *x_col*
      Column in which *d_shapeId* occurs inside this mosaic instance.
- *d_shapeId*
  The database ID of the shape.

© 2020 All Rights Reserved.

Value Returned

l_colorAndColorState	(*t_color b_isLocked b_isHCL*) A list of elements. where, ■ *t_color* Name of the color mask (`"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, `"mask4Color"`, `"mask5Color"`, `"mask6Color"`, `"mask7Color"`, or `"mask8Color"`) ■ *b_isLocked* A Boolean value that determines whether the color of the shape is locked. ■ *b_isHCL* A Boolean value that specifies whether the shape color is due to the color attribute (`nil`) or hierarchical color locking (`t`).
`nil`	Returned in case of failure.

Examples

Example 1–Printing color information for shapes in a mosaic

```
cv = dbOpenCellView("lib" "top" "layout")
mosaic = dbFindMosaicByName(cv "M1")
foreach(shape mosaic->master->shapes
   hierPath = list(list(mosaic 1 0))
   occShape = list(hierPath shape)
   colorInfo = dbGetShapeEffectiveColor(occshape)
   printf("color=%s, isLocked=%L, isHCL=%L\n"
           car(colorInfo) cadr(colorInfo) caddr(colorInfo)
          )
   )
```

Prints hierarchical coloring information (color, color state, and whether coloring is from hierarchical color locking) for each shape in the row 1 and column 0 element of the mosaic `M1`.

Example 2–Printing color information for hierarchical and non-hierarchical shapes

```
shapes = dbShapeQuery( deGetCellView() list("M1" "drawing") list(0:0 0:0) )
foreach( occ shapes
   if( listp(occ) then
```


© 2020 All Rights Reserved.

```
      path = reverse(cdr(reverse(occ))) ; gets all elements of the list but last
      shape = car(last(occ))
      println(dbGetShapeEffectiveColor(list(path shape)))
   else
      println(list(dbGetShapeColor(occ) dbIsShapeColorLocked(occ)))
   ) ; if
) ; foreach
```

Prints coloring information for `M1` shapes on the `drawing` purpose. The coloring information is printed by dbGetShapeEffectiveColor for hierarchical shapes, and by dbGetShapeColor and dbIsShapeColorLocked for non-hierarchical shapes.

© 2020 All Rights Reserved.

dbSetOccShapeColor

```
dbSetOccShapeColor(
     l_occShape
     t_color
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Assigns the specified color to the given occurrence shape. You can set the color for occurrence shapes with types `Path`, `PathSeg`, `Polygon`, and `Rect`.

© 2020 All Rights Reserved.

Arguments

l_occShape

`list(l_hierPath d_shapeId)`

where,

- *l_hierPath*
  is
  `list((d_refId | l_mosaicElem)…)`

  where,

  - *d_refId*
    The database ID of the non-mosaic instance or via.
  - *l_mosaicElem*
    is
    `list(d_mosaicId x_row x_col)`
    Represents a mosaic element in the hierarchy path.

    where,

    - *d_mosaicId*
      The database ID of the mosaic instance.
    - *x_row*
      Row in which *d_shapeId* occurs inside this mosaic instance.
    - *x_col*
      Column in which *d_shapeId* occurs inside this mosaic instance.
- *d_shapeId*
  The database ID of the shape to be colored.
  Supported shapes: `Path`, `PathSeg`, `Polygon`, `Rect`

t_color

Name of the color mask (`"grayColor"`, `"mask<mask number>Color"`, where `<mask number>` is 1, 2, and so on, up to the number of masks defined in the technology library for the layer). If the color is `"grayColor"` and there is a hierarchical color lock (HCL) on the occurrence shape, the HCL will be removed if the color state is unlocked.

© 2020 All Rights Reserved.

Value Returned

`t`	Color is assigned to the specified occurrence shape.
`nil`	Returned in case of an error. An error occurs when an attempt is made to: ■ Set the color of the occurrence shape, and the color state of the occurrence shape or the shape is found to be locked. Unlock the color state of the occurrence shape before changing its color. ■ Set the color mask to `'multi'` or `'black'`. ■ Set the color of an occurrence shape that is on an unbounded layer or on a layer that does not support coloring. ■ Set the color for occurrence shapes with a type other than `Path`, `PathSeg`, `Polygon`, or `Rect`.

Examples

```
cv = dbOpenCellViewByType("tech" "design" "layout" "maskLayout" "w")
top = dbOpenCellViewByType("tech" "top" "layout" "maskLayout" "w")
inst = dbCreateInst(top cv "InstInTop" '(0 0) "R0")
rect0 = dbCreateRect(cv '("metal2" "drawing") '((1 8.6) (5.3 6.6)))
path = list(list(inst) rect0)
dbSetOccShapeColor(path "mask2Color")
```

Creates a one-level occurrence shape `rect0`, then sets its color to `mask2Color`.

```
viaDef_M2_M1 = techFindViaDefByName(tf "M2_M1")
via1t_m2_m1  = dbCreateVia(cv viaDef_M2_M1 10:10 "R0")
path = list(list(via1t_m2_m1) nth(1 via1t_m2_m1~>viaHeader~>master~>shapes))
dbSetOccShapeColor(path "mask1Color")
```

Creates a via, then sets the color of a randomly picked shape from the via master to `mask1Color`.

```
rect1 = dbCreateRect(cv '("metal2" "drawing") '((0 7.6) (4.3 5.6)))
mosaic = dbCreateSimpleMosaic(top cv "MosaicInTop" list(0 0) "R0" 2 2 2 2)
path = list(list(list(mosaic 1 1)) rect1)
dbSetOccShapeColor(path "mask2Color")
dbGetShapeEffectiveColor(path)
```

Sets the color of the instance of `rect1` inside the mosaic to `mask2Color`.

```
dbSetOccShapeColor((list(list(inst1 inst2) rect) "mask1Color")
```

Sets the color of the `rect` shapes in the instances given by `inst1` and `inst2` to `mask1Color`.

© 2020 All Rights Reserved.

dbSetOccShapeColorLocked

```
dbSetOccShapeColorLocked(
     l_occShape
     g_setLocked
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Changes the color state of the specified occurrence shape. You can set the color state for occurrence shapes with types `Path`, `PathSeg`, `Polygon`, and `Rect`.

© 2020 All Rights Reserved.

Arguments

l_occShape `list(l_hierPath d_shapeId)`

where,

- *l_hierPath*
  is
  `list((d_refId | l_mosaicElem)…)`

  where,

  - *d_refId*
    The database ID of the non-mosaic instance or via.
  - *l_mosaicElem*
    is
    `list(d_mosaicId x_row x_col)`
    Represents a mosaic element in the hierarchy path.

    where,

    - *d_mosaicId*
      The database ID of the mosaic instance.
    - *x_row*
      Row in which *d_shapeId* occurs inside this mosaic instance.
    - *x_col*
      Column in which *d_shapeId* occurs inside this mosaic instance.
- *d_shapeId*
  The database ID of the shape.
  Supported shapes: `Path`, `PathSeg`, `Polygon`, `Rect`

g_setLocked `t|nil`

- `t`
  The occurrence shape will be `locked`.
- `nil`
  The occurrence shape will be `unlocked`.

© 2020 All Rights Reserved.

Value Returned

`t`	Color state of the specified occurrence shape is set.
`nil`	Returned in case of an error. An error occurs when an attempt is made to: ■ Set the color state of a non-colorable shape. ■ Set the color state of an occurrence shape that has color set to `"grayColor"`. ■ Set the color state for an occurrence shape with a type other than `Path`, `PathSeg`, `Polygon`, or `Rect`.

Examples

```
cv = dbOpenCellViewByType("tech" "design" "layout" "maskLayout" "w")
top = dbOpenCellViewByType("tech" "top" "layout" "maskLayout" "w")
inst = dbCreateInst(top cv "InstInTop" '(0 0) "R0")
rect0 = dbCreateRect(cv '("metal2" "drawing") '((1 8.6) (5.3 6.6)))
path = list(list(inst) rect0)
dbSetOccShapeColor(path "mask1Color")
dbSetOccShapeColorLocked(path t)
```

Creates a one-level occurrence shape `rect0`, then sets it color to `mask1Color` and locks the color.

```
viaDef_M2_M1 = techFindViaDefByName(tf "M2_M1")
via1t_m2_m1 = dbCreateVia(cv viaDef_M2_M1 10:10 "R0")
path = list(list(via1t_m2_m1) nth(1 via1t_m2_m1~>viaHeader~>master~>shapes))
dbSetOccShapeColor(path "mask2Color")
dbSetOccShapeColorLocked(path t)
```

Creates a via, then sets the color of a randomly picked shape from the via master to `mask2Color` and locks the color.

© 2020 All Rights Reserved.

dbGetHierColorOverride

```
dbGetHierColorOverride(
     l_occShape
     )
     => l_occShapeHCL / nil
```

Description

(ICADVM20.1 Only) Returns the occurrence shape with the topmost hierarchical color lock that defines the effective color of the given occurrence shape. `nil` is returned if the effective color of the given occurrence shape is not defined by a hierarchical color lock.

© 2020 All Rights Reserved.

Arguments

l_occShape `list(l_hierPath d_shapeId)`

where,

- *l_hierPath*
  is
  `list((d_refId | l_mosaicElem)…)`

  where,

  - *d_refId*
    The database ID of the non-mosaic instance or via.
  - *l_mosaicElem*
    is
    `list(d_mosaicId x_row x_col)`
    Represents a mosaic element in the hierarchy path.

    where,

    - *d_mosaicId*
      The database ID of the mosaic instance.
    - *x_row*
      Row in which *d_shapeId* occurs inside this mosaic instance.
    - *x_col*
      Column in which *d_shapeId* occurs inside this mosaic instance.
- *d_shapeId*
  The database ID of the shape.

Value Returned

l_occShapeHCL `list(l_hierPath d_shapeId)`

List representing an occurrence shape using the same format as the input argument (*l_occShape*).

© 2020 All Rights Reserved.

`nil`	There is no hierarchical color lock that defines the effective color for the given shape or there was an error in processing (for example, the occurrence does not exist).

Example

```
bot=dbOpenCellViewByType("tech" "bot" "layout" "maskLayout" "w")
mid=dbOpenCellViewByType("tech" "mid" "layout" "maskLayout" "w")
top=dbOpenCellViewByType("tech" "top" "layout" "maskLayout" "w")

topToMid = dbCreateInst(top mid "topToMid" '(30 0) "R0")
midToBot = dbCreateInst(mid bot "midToBot" '(0 0) "R0")

rect = dbCreateRect(bot '("metal2" "drawing") '((1 8.6) (5.3 6.6)))
dbSetOccShapeColor(list(list(midToBot) rect) "mask1Color")
dbSetOccShapeColorLocked(list(list(midToBot) rect) t)

hierPath = dbGetHierColorOverride(list(list(topToMid midToBot) rect))
```

Creates a hierarchical color lock on a shape `rect` in an instance, then returns the hierarchy path to that shape where the hierarchical color lock is set.

```
foreach( l dbColorShapeQuery2(top list("metal2" "drawing") top~>bBox )
   occShape = car(l)
   locked   = caddr(l)
   isHCL    = cadddr(l)
   when(locked && isHCL
      hcl = dbGetHierColorOverride( occShape )
      dbSetOccShapeColorLocked( hcl nil )
      dbSetOccShapeColor( hcl "grayColor" )

   )
)
```

Removes the color and hierarchical color lock on all the `metal2:drawing` shapes with a hierarchical color lock in the `top` cellview.

© 2020 All Rights Reserved.

dbGetShapeColorLockType

```
dbGetShapeColorLockType(
     d_shapeId
     )
     => t_lockType / nil
```

Description

(ICADVM20.1 Only) Returns the color lock type assigned to the specified shape. If the shape is not locked, `nil` is returned.

Arguments

d_shapeId	The database ID of a shape.

Value Returned

t_lockType	Color lock type assigned to the specified shape. Valid values: ■ `user` Coloring of the shape is locked and the lock type `"system"` is not set. ■ `system` Coloring of the shape is locked and the lock type `"system"` has been set using the dbSetShapeColorLockType SKILL function or some other Cadence application. By default, `"user"` lock type is assigned to locked shapes.
`nil`	Shape is not locked.

© 2020 All Rights Reserved.

Example

```
shape = css()
```

Returns database ID of the currently selected shape.

```
dbSetShapeColorLocked(shape nil)
```

Unlocks the shape color.

```
dbSetShapeColor(shape "mask1Color")
```

Changes shape color.

```
dbGetShapeColorLockType(shape)
=> nil
```

Shape color is unlocked.

```
dbSetShapeColorLocked(shape t)
```

Locks the shape color.

```
dbGetShapeColorLockType(shape)
=> "user"
```

Color lock type as `user`.

```
dbSetShapeColorLockType(shape "system")
```

Changes color lock type to `system`.

```
dbGetShapeColorLockType(shape)
=> "system"
```

Color lock type is `system`.

```
dbSetShapeColorLockType(shape nil)
```

Unlocks color of the locked shape. Shows that `dbSetShapeColorLockType` can be used to unlock color of the locked shape.

```
dbGetShapeColorLockType(shape)
=> nil
```

Shape color is locked.

```
dbIsShapeColorLocked(shape)
=> nil
```

Confirms that shape color is not locked

```
dbSetShapeColorLockType(shape "system")
```

Sets color lock type as `system` directly on the unlocked shape.

```
dbGetShapeColorLockType(shape)
```


© 2020 All Rights Reserved.

```
=> "system"
```

Color lock type is `system`.

```
dbIsShapeColorLocked(shape)
=> t
```

Confirms that shape color is locked.

© 2020 All Rights Reserved.

dbSetShapeColorLockType

```
dbSetShapeColorLockType(
     d_shapeId
     g_lockType
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the color lock type for the specified shape.

Arguments

d_shapeId	The database ID of a shape.
g_lockType	Color lock type to assign to the specified shape. Valid values: ■ `"user"` Sets coloring of the shape to locked, with lock type `"user"`. All shapes locked in the older versions of Virtuoso have these lock type. ■ `"system"` Sets coloring of the shape to locked, with lock type `"system"`. This lock type indicates that the shape was locked by an automated coloring tool. ■ `nil` Unlocks coloring of the shape and clears information about its lock type. Default value for existing locked shapes is `"user"`

Value Returned

`t`	Specified lock type is successfully set.
`nil`	Lock type is not set.

© 2020 All Rights Reserved.

Example

```
shape = css()
```

Returns database ID of the currently selected shape.

```
dbSetShapeColorLocked(shape nil)
```

Unlocks the shape color.

```
dbSetShapeColor(shape "mask1Color")
```

Changes shape color.

```
dbGetShapeColorLockType(shape)
=> nil
```

Shape color is unlocked.

```
dbSetShapeColorLocked(shape t)
```

Locks shape color.

```
dbGetShapeColorLockType(shape)
=> "user"
```

Color lock type is `user`.

```
dbSetShapeColorLockType(shape "system")
```

Changes color lock type to `system`.

```
dbGetShapeColorLockType(shape)
=> "system"
dbSetShapeColorLockType(shape nil)
```

Unlocks color of the locked shape. Shows that `dbSetShapeColorLockType` can be used to unlock color of the locked shape.

```
dbGetShapeColorLockType(shape)
=> nil
```

Shape color is unlocked.

```
dbIsShapeColorLocked(shape)
=> nil
```

Confirms that shape color is unlocked.

```
dbSetShapeColorLockType(shape "system")
```

Color lock type set directly on unlocked shape using `dbSetShapeColorLockType`.

```
dbGetShapeColorLockType(shape)
=> "system"
```


© 2020 All Rights Reserved.

Color lock type is `system`.

```
dbIsShapeColorLocked(shape)
=> t
```

Confirms that shape color is locked.

© 2020 All Rights Reserved.

dbGetViaLayerColorLockType

```
dbGetViaLayerColorLockType(
     d_viaId
     t_viaLayerType
     )
     => t_lockType / nil
```

Description

(ICADVM20.1 Only) Returns the color lock type assigned to the specified via layer.

Arguments

d_viaId	The database ID of a via layer.
t_viaLayerType	Specifies the via layer type. Valid values: `layer1`, `layer2`, `cutLayer`

Value Returned

t_lockType	Color lock type assigned to the specified via layer. Valid values: ■ `user` Coloring of the via layer is locked and the lock type "`system`" is not set. ■ `system` Coloring of the via layer is locked and the lock type "`system`" has been set using the dbSetViaLayerColorLockType SKILL function or some other Cadence application.
`nil`	Via layer is not locked.

© 2020 All Rights Reserved.

Example

```
via = css()
```

Returns database ID of the currently selected via.

```
dbSetViaColorInfo(via
                  list("shift1" nil)
                  nil
                  nil
                  )
```

Sets coloring of via `layer1` to unlocked `shift1`.

```
dbGetViaLayerLockType(via "layer1")
=> nil
dbSetViaColorStateLayerLocked(via "layer1" t)
```

Locks color of `layer1`.

```
dbGetViaLayerLockType(via "layer1")
=> "user"
```

Lock type for `layer1` is `user`.

```
dbSetViaLayerColorLockType(via "layer1" "system")
```

Changes color lock type to `system`.

```
dbGetViaLayerLockType(via "layer1")
=> "system"
```

Lock type for `layer1` is `system`.

```
dbSetViaLayerColorLockType(via "layer1" nil)
```

Lock via `layer1` using `dbSetViaLayerColorLockType`.

```
dbGetViaLayerLockType(via "layer1")
=> nil
```

Via `layer1` is unlocked.

```
dbIsViaColorStateLayerLocked(via "layer1")
=> nil
```

Confirms that vial layer1 is unlocked.

```
dbSetViaColorInfo(via
                  list("noShift" "system")
                  nil
                  nil)
```

Sets color lock type `system` directly on unlocked via `layer1`.

© 2020 All Rights Reserved.

```
dbGetViaLayerLockType(via "layer1")
=> "system"
```

Lock type for `layer1` is `system`.

```
dbIsViaColorStateLayerLocked(via "layer1")
=> t
```

Confirms that via `layer1` is locked.

© 2020 All Rights Reserved.

dbSetViaLayerColorLockType

```
dbSetViaLayerColorLockType(
     d_viaId
     t_viaLayer
     g_lockType
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the color lock type for the specified via layer.

Arguments

d_viaId	The database ID of a via layer.
t_viaLayerType	Specifies the via layer type. Valid values: `layer1`, `layer2`, `cutLayer`
g_lockType	Color lock type to assign to the specified via layer. Valid values: ■ `"user"` Sets coloring of the via layer to locked, with lock type `"user"`. All via layers locked in the older versions of Virtuoso have these lock type. ■ `"system"` Sets coloring of the via layer to locked, with lock type `"system"`. This lock type indicates that the via layer was locked by an automated coloring tool. ■ `nil` Unlocks coloring of the via layer and clears information about its lock type. Default value for existing locked via layers is `"user"`

Value Returned

`t`	Specified lock type is successfully set.
`nil`	Lock type is not set.

© 2020 All Rights Reserved.

Example

```
via = css()
```

Returns database ID of the currently selected via.

```
dbSetViaColorInfo(via
                  list("shift1" nil)
                  nil
                  nil
                  )
```

Sets coloring of via `layer1` to unlocked `shift1`.

```
dbGetViaLayerLockType(via "layer1")
=> nil
dbSetViaColorStateLayerLocked(via "layer1" t)
```

Locks color of `layer1`.

```
dbGetViaLayerLockType(via "layer1")
=> "user"
```

Lock type for `layer1` is `user`.

```
dbSetViaLayerColorLockType(via "layer1" "system")
```

Changes color lock type to `system`.

```
dbGetViaLayerLockType(via "layer1")
=> "system"
```

Lock type for `layer1` is `system`.

```
dbSetViaLayerColorLockType(via "layer1" nil)
```

Lock via `layer1` using `dbSetViaLayerColorLockType`.

```
dbGetViaLayerLockType(via "layer1")
=> nil
```

Via `layer1` is unlocked.

```
dbIsViaColorStateLayerLocked(via "layer1")
=> nil
```

Confirms that vial layer1 is unlocked.

```
dbSetViaColorInfo(via
                  list("noShift" "system")
                  nil
                  nil)
```

Sets color lock type `system` directly on unlocked via `layer1`.

© 2020 All Rights Reserved.

```
dbGetViaLayerLockType(via "layer1")
=> "system"
```

Lock type for `layer1` is `system`.

```
dbIsViaColorStateLayerLocked(via "layer1")
=> t
```

Confirms that via `layer1` is locked.

© 2020 All Rights Reserved.

dbCellViewHasUnknownColorLockChanges

```
dbCellViewHasUnknownColorLockChanges(
     d_cellViewId
     [ g_shapes g_vias g_HCLs ]
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Checks whether any edits were done to the specified type of objects in the older versions of Virtuoso after color lock type information was set for some of them.

Arguments

d_cellViewId	Specifies the cellview ID.
g_shapes	A Boolean specifying whether the function should check color lock type of the shapes in the specified cellview. The default is `t`.
g_vias	A Boolean specifying whether the function should check color lock type of the vias in the cellview. The default is `t`.
g_HCLs	A Boolean specifying whether the function should check color lock type of the HCLs in the cellview. The default is `t`.

Value Returned

`t`	Specified cellview has objects with unknown edits.
`nil`	Specified cellview has no objects with unknown edits.

Example

Consider that user1 creates a cellview with color lock type `"system"` using Virtuoso version ICADVM20.1 Base release.

```
cv = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "w")
r  = dbCreateRect(a "M1" list(0.0:0.0 0.1:0.1))
```


© 2020 All Rights Reserved.

```
dbSetShapeColor(r1 "mask1Color")
dbSetShapeColorLockType(or1 "system")
dbSave(cv)
```

Next, user2 adds one more shape to this cellview using any advanced nodes Virtuoso release older than ICADV12.3 ISR15 and sends the data back to user1.

```
cv = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "a")
r  = dbCreateRect(a "M2" list(0.2:0.0 0.3:0.1))
dbSave(cv)
```

User1 will see that there are unknown changes on object type `shapes`.

```
cv = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "a")
dbCellViewHasUnknownColorLockChanges(cv)
=> t

dbCellViewHasUnknownColorLockChanges(cv nil t t)
=> nil
```

The `dbCellViewHasUnknownColorLockChanges` function allows any edits on the given object type. Therefore even if user2 did not change any coloring in the cellview, user1 cannot see any changes made by user2.

In this scenario, it is recommended to reset all lock types in the cellview:

```
dbCellViewResetAllColorLockTypes(cv "system")
dbCellViewHasUnknownColorLockChanges(cv)
=> nil
```


© 2020 All Rights Reserved.

dbGetShapeEffectiveColorLockType

```
dbGetShapeEffectiveColorLockType(
     l_occShape
     )
     => t_colorLockType / nil
```

Description

(ICADVM20.1 Only) Returns the effective color lock type of the specified occurrence shape. It is either the color lock type of the topmost HCL, returned by dbGetHierColorOverride or the color lock type of the shape itself.

© 2020 All Rights Reserved.

Arguments

l_occShape `list(l_hierPath d_shapeId)`

where,

- *l_hierPath*
  is
  `list((d_refId | l_mosaicElem)…)`

  where,

  - *d_refId*
    The database ID of the non-mosaic instance or via.
  - *l_mosaicElem*
    is
    `list(d_mosaicId x_row x_col)`
    Represents a mosaic element in the hierarchy path.

    where,

    - *d_mosaicId*
      The database ID of the mosaic instance.
    - *x_row*
      Row in which *d_shapeId* occurs inside this mosaic instance.
    - *x_col*
      Column in which *d_shapeId* occurs inside this mosaic instance.
- *d_shapeId*
  The database ID of the shape.

© 2020 All Rights Reserved.

Value Returned

t_colorLockType	Effective color lock type of the specified occurrence shape.
	Valid values:
	`user`
	Coloring of the via layer is locked.
	`system`
	Coloring of the shape was locked by automated coloring tools.

`nil`	Color of the shape is unlocked.

Example

Create a hierarchy of three cellviews.

```
a = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "w")
b = dbOpenCellViewByType("lib" "b" "layout" "maskLayout" "w")
c = dbOpenCellViewByType("lib" "c" "layout" "maskLayout" "w")

ibc = dbCreateInst(b c "ibc" 0:0 "R0")
iab = dbCreateInst(a b "iab" 0:0 "R0")
```

Create rectangles at two lower levels and color them with HCLs.

```
r1 = dbCreateRect(c "M1" list(0.0:0.0 0.1:0.1))
r2 = dbCreateRect(c "M1" list(0.2:0.0 0.3:0.1))
r3 = dbCreateRect(b "M1" list(0.0:0.2 0.1:0.3))

or1 = list(list(ibc) r1)
dbSetOccShapeColor(or1 "mask1Color")
dbSetOccShapeColorLockType(or1 "user")

or2 = list(list(iab ibc) r2)
dbSetOccShapeColor(or2 "mask2Color")
dbSetOccShapeColorLockType(or2 "system")

or3 = list(list(iab) r3)
dbSetOccShapeColor(or3 "mask1Color")
```

Define a helper function to print `occShapes`.

```
defun(humanizeOccShape (occShape)
    when(occShape
```


© 2020 All Rights Reserved.

```
        list(
            mapcar(lambda((inst) inst->name) car(occShape))
            cadr(occShape)->bBox
        ) ; cons
    ) ; when
) ; defun
```

Query all occurrences of all `M1` shapes in the cellview `a` and print the information about their coloring, including lock type and override.

```
foreach(l dbColorShapeQuery2(a "M1" a->bBox)
    occShape = car(l)
    color    = cadr(l)
    lockType = dbGetShapeEffectiveColorLockType(occShape)
    isHCL    = cadddr(l)
    hcl      = when(isHCL dbGetHierColorOverride(occShape))

    printf("occ shape %L:\n  color=%L\n  lock type=%L\n  set by the override=%L\n"
           humanizeOccShape(occShape) color lockType humanizeOccShape(hcl))
) ; foreach
```

The output will be similar to the following:

```
occ shape (("iab") ((0.0 0.2) (0.1 0.3))):
  color="mask1Color"
  lock type=nil
  set by the override=(("iab") ((0.0 0.2) (0.1 0.3)))
occ shape (("iab" "ibc") ((0.0 0.0) (0.1 0.1))):
  color="mask1Color"
  lock type="user"
  set by the override=(("ibc") ((0.0 0.0) (0.1 0.1)))
occ shape (("iab" "ibc") ((0.2 0.0) (0.3 0.1))):
  color="mask2Color"
  lock type="system"
  set by the override=(("iab" "ibc") ((0.2 0.0) (0.3 0.1)))
```


© 2020 All Rights Reserved.

dbCellViewResetAllColorLockTypes

```
dbCellViewResetAllColorLockTypes(
     d_cellViewId
     t_value
     [g_shapes g_vias g_HCLs]
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Changes color lock type for all color-locked objects of the specified object types to the given value.

Arguments

d_cellViewId	Specifies the cellview ID.
t_value	Color lock type to reset the specified objects. Valid values: `"user"`, `"system"`
g_shapes	A Boolean specifying whether the function should reset color lock type of the shapes in the specified cellview. The default is `t`.
g_vias	A Boolean specifying whether the function should reset color lock type of the vias in the cellview. The default is `t`.
g_HCLs	A Boolean specifying whether the function should reset color lock type of the HCLs in the cellview. The default is `t`.

Value Returned

`t`	Color lock type is reset to the specified value for all color-locked objects of the specified object types.
`nil`	An error occurs.

© 2020 All Rights Reserved.

Example

Consider that user1 creates a cellview with color lock type `"system"` using Virtuoso version ICADVM20.1 Base release.

```
cv = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "w")
r  = dbCreateRect(a "M1" list(0.0:0.0 0.1:0.1))
dbSetShapeColor(r1 "mask1Color")
dbSetShapeColorLockType(or1 "system")
dbSave(cv)
```

Next, user2 adds one more shape to this cellview using any advanced nodes Virtuoso release older than ICADV12.3 ISR15 and sends the data back to user1.

```
cv = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "a")
r  = dbCreateRect(a "M2" list(0.2:0.0 0.3:0.1))
dbSave(cv)
```

User1 will see that there are unknown changes on object type `shapes`.

```
cv = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "a")
dbCellViewHasUnknownColorLockChanges(cv)
=> t

dbCellViewHasUnknownColorLockChanges(cv nil t t)
=> nil
```

The `dbCellViewHasUnknownColorLockChanges` function allows any edits on the given object type. Therefore even if user2 did not change any coloring in the cellview, user1 cannot see any changes made by user2.

In this scenario, it is recommended to reset all lock types in the cellview:

```
dbCellViewResetAllColorLockTypes(cv "system")

dbCellViewHasUnknownColorLockChanges(cv)
=> nil
```


© 2020 All Rights Reserved.

dbSetOccShapeColorLockType

```
dbSetOccShapeColorLockType(
     l_occShape
     g_colorLockType
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Changes the color lock type of the specified occurrence shape. You can set the color state only for occurrence shapes with the following types: Path, PathSeg, Polygon, and Rect.

Error occurs when you try to set the color state:

- of a non-colorable shape.
- if the color of the occurrence shape is gray because it is inconsistent to lock the coloring of an uncolored shape.

Note: You can use the dbGetShapeEffectiveColorLockType function to check the color lock type of a given occShape.

© 2020 All Rights Reserved.

Arguments

l_occShape `list(l_hierPath d_shapeId)`

where,

- *l_hierPath*
  is
  `list((d_refId | l_mosaicElem)…)`

  where,

  - *d_refId*
    The database ID of the non-mosaic instance or via.
  - *l_mosaicElem*
    is
    `list(d_mosaicId x_row x_col)`
    Represents a mosaic element in the hierarchy path.

    where,

    - *d_mosaicId*
      The database ID of the mosaic instance.
    - *x_row*
      Row in which *d_shapeId* occurs inside this mosaic instance.
    - *x_col*
      Column in which *d_shapeId* occurs inside this mosaic instance.
- *d_shapeId*
  The database ID of the shape.

© 2020 All Rights Reserved.

g_colorLockType	Color lock type to assign to the specified occurrence shape. Valid values: ■ `"user"` Coloring of the occurrence shape is locked. ■ `"system"` Coloring of the occurrence shape was locked by automated coloring tools. ■ `nil` Color of the occurrence shape is unlocked.

Value Returned

`t`	Specified lock type is successfully set.
`nil`	Lock type is not set.

Example

Create a hierarchy of three cellviews.

```
a = dbOpenCellViewByType("lib" "a" "layout" "maskLayout" "w")
b = dbOpenCellViewByType("lib" "b" "layout" "maskLayout" "w")
c = dbOpenCellViewByType("lib" "c" "layout" "maskLayout" "w")

ibc = dbCreateInst(b c "ibc" 0:0 "R0")
iab = dbCreateInst(a b "iab" 0:0 "R0")
```

Create rectangles at two lower levels and color them with HCLs.

```
r1 = dbCreateRect(c "M1" list(0.0:0.0 0.1:0.1))
r2 = dbCreateRect(c "M1" list(0.2:0.0 0.3:0.1))
r3 = dbCreateRect(b "M1" list(0.0:0.2 0.1:0.3))

or1 = list(list(ibc) r1)
dbSetOccShapeColor(or1 "mask1Color")
dbSetOccShapeColorLockType(or1 "user")

or2 = list(list(iab ibc) r2)
dbSetOccShapeColor(or2 "mask2Color")
dbSetOccShapeColorLockType(or2 "system")
```


© 2020 All Rights Reserved.

```
or3 = list(list(iab) r3)
dbSetOccShapeColor(or3 "mask1Color")
```

Define a helper function to print `occShapes`.

```
defun(humanizeOccShape (occShape)
    when(occShape
        list(
            mapcar(lambda((inst) inst->name) car(occShape))
            cadr(occShape)->bBox
        ) ; cons
    ) ; when
) ; defun
```

Query all occurrences of all `M1` shapes in the cellview `a` and print the information about their coloring, including lock type and override.

```
foreach(l dbColorShapeQuery2(a "M1" a->bBox)
    occShape = car(l)
    color    = cadr(l)
    lockType = dbGetShapeEffectiveColorLockType(occShape)
    isHCL    = cadddr(l)
    hcl      = when(isHCL dbGetHierColorOverride(occShape))

    printf("occ shape %L:\n  color=%L\n  lock type=%L\n  set by the override=%L\n"
           humanizeOccShape(occShape) color lockType humanizeOccShape(hcl))
) ; foreach
```

The expected output will be similar to the following:

```
occ shape (("iab") ((0.0 0.2) (0.1 0.3))):
  color="mask1Color"
  lock type=nil
  set by the override=(("iab") ((0.0 0.2) (0.1 0.3)))
occ shape (("iab" "ibc") ((0.0 0.0) (0.1 0.1))):
  color="mask1Color"
  lock type="user"
  set by the override=(("ibc") ((0.0 0.0) (0.1 0.1)))
occ shape (("iab" "ibc") ((0.2 0.0) (0.3 0.1))):
  color="mask2Color"
  lock type="system"
  set by the override=(("iab" "ibc") ((0.2 0.0) (0.3 0.1)))
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

14c

Width Spacing Pattern Database Access Functions (ICADVM20.1 Only)

This chapter includes the following topics:

- Width Spacing Pattern Functions
- Width Spacing Pattern Group Functions
- Width Spacing Snap Pattern Def Functions
- Related Snap Pattern Functions
- Pattern Region Functions
- Active Wire Types Functions
- Global Grid Offset Functions

© 2020 All Rights Reserved.

Width Spacing Pattern Functions

Supported functions include:

- dbCreateWidthSpacingPattern
- dbCreateWidthSpacingPatternWithColor
- dbFindWidthSpacingPattern
- dbGetWidthSpacingPatternAllowedRepeatMode
- dbGetWidthSpacingPatternDefaultRepeatMode
- dbGetWidthSpacingPatterns
- dbSetWidthSpacingPatternRepeatMode

© 2020 All Rights Reserved.

dbCreateWidthSpacingPattern

```
dbCreateWidthSpacingPattern(
     d_cellView
     t_name
     l_patternSpecs
     [ n_offset ]
     [ g_repeatOffset ]
     [ g_shiftColor ]
     [ t_allowedRepeatMode ]
     [ t_defaultRepeatMode ]
     )
     => d_widthSpacingPattern / nil
```

Description

(ICADVM20.1 Only) Creates a width spacing pattern in the specified cellview.

Arguments

d_cellView	ID of the cellview for which the width spacing pattern must be created.
t_name	Name of the width spacing pattern.
l_patternSpecs	`list(`*l_patternSpec*`)` List of track group specifications. where, ■ *l_patternSpec*: `list(`*l_specs* [*d_repeat* [*l_wireTypes* [*l_colorNames*] [*l_displayPacketNames*]]]) A track group specification. ❑ *l_specs*: `list(`*l_spec*`)` List of individual track specifications for the specified track group. ❍ *l_spec*: `list(`*g_width g_space* [*t_wireType* [*t_colorName*] [*t_displayPacketName*]]) A track specification. *g_width* Width of shapes on the specified track.

© 2020 All Rights Reserved.

g_space
The centerline distance of the next track from the current track, in the period direction.

t_wireType
The type of wire. This parameter can be specified only when *d_repeat* *l_wireTypes* is not specified. It is used to refer to a specific group of tracks, for example, routing a set of tracks by name. The default value is an empty string `""`.

t_colorName
Color of the specified track. By default, tracks are not colored. This parameter can be specified only when *d_repeat* *l_wireTypes* is not specified. Valid values are `"mask1Color"`, `"mask2Color"`, and `"mask3Color"`.

t_displayPacketName
Name of a display packet that can be specified to control how the track should be drawn on the screen. It can be specified only if *t_colorName* is specified.

- *d_repeat*
  The number of times the track group is repeated. A value of 1 creates the track group from the specified *l_specs* (1x). A value of 2 creates the track group from the specified *l_specs* that are repeated once (2x).

- *l_wireTypes*: `list([t_wireType …])`
  List of wire types associated with the specified track group (only if *d_repeat* is specified on the track group). Use an empty string (`""`) to specify no wire type for an individual track. If the number of tracks in the track group is greater than the number of wire types in this list, then the wire types list is repeated across the track group until all tracks are assigned a wire type, one-by-one.

© 2020 All Rights Reserved.

- *t_wireType*
  The type of wire.

For example, for a track group with 6 tracks:

- If *l_wireTypes* is `list("a" "b")`, then tracks 1, 3, and 5 are wire type **a**; tracks 2, 4, and 6 are wire type **b**.
- If *l_wireTypes* is `list("a" "a" "b")`, then tracks 1, 2, 4, and 5 are wire type **a**; tracks 3 and 6 are wire type **b**.

- *l_colorNames*: `list([`*t_colorName* …`])`
  List of colors that is applied to the specified track group (only if *d_repeat* is specified on the track group). Use `"grayColor"` to leave an individual track uncolored. If the number of tracks in the specified track group is greater than the number of colors in this list, then the color list is repeated across the track group until all tracks are assigned a color, one-by-one.

  - *t_colorName*
    Color of the specified track. Valid values are `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, and `"grayColor"`.

  For example, for a track group with 3 tracks and *l_colorNames* is `list("mask1Color" "mask2Color")`, then tracks 1 and 3 are `mask1Color`, and track 2 is `mask2Color`.

- *l_displayPacketNames*
  List of display packet names for the track group (only if *d_repeat* is specified).

  Default value: An empty list.

  l_displayPacketName: `list([`*t_displaypacketName* …`])`

  t_displayPacketName is the display packet name for the specified track.

© 2020 All Rights Reserved.

n_offset	Distance of the first track from the period track.
g_repeatOffset	Determines whether the offset is applied when a pattern repeats. When set to `t` the offset is repeated every time and when set to `nil` the offset is applied only once.
g_shiftColor	Determines whether the track colors are shifted when a pattern repeats. The parameter is optional and is used only when all tracks are individually colored. If no color is specified for a track, then it remains uncolored (or gray). The default value is `nil`. The tracks cannot be colored automatically.
t_allowedRepeatMode	Name of allowed repeat mode. Valid values: `any`, `none`, `steppedOnly`, and `flippedOnly`. The default value is "`any`".
t_defaultRepeatMode	Name of default repeat mode. Valid values are Valid values: `noRepeat`, `stepped`, `flippedStartsWithOdd`, and `flippedStartsWithEven`. The default value is "`noRepeat`".

Value Returned

d_widthSpacingPattern	The ID of the created width spacing pattern.
`nil`	Returned in case of failure.

Examples

Example 1

```
dbCreateWidthSpacingPattern(cv "wsp1" '((((0.04 0.08 "mywire1" "mask1Color") (0.09
0.12)))) 0.05 )
```

In the above example, a width spacing pattern `wsp1` is created in the cellview `cv` with two track specifications.

- For the first track,
  - `0.04` is the width of the shapes.

© 2020 All Rights Reserved.

- `0.08` is the distance of the next track from the current track, in the period direction.
- `mywire1` is the type of wire to use.
- `mask1Color` is the mask color.
- `0.05` is the distance of the first track from the period track.

- For the second track,
  - 0.09 is the width of the shapes.
  - 0.12 is the distance of the next track from the current track, in the period direction.

Example 2

```
dbCreateWidthSpacingPattern(cv "wsp2"
   '((((0.04 0.08 "mywire1" "mask1Color") (0.09 0.12)) 2))
   0.05
)
```

This is similar to the previous example. However, the two tracks are repeated (repeat value of 2). Therefore, the pattern consists of 4 tracks, as shown below.

© 2020 All Rights Reserved.

Example 3

```
dbCreateWidthSpacingPattern(cv "wsp1"
    '(
        (
            (
                (0.04 0.08)
            )
            4
            ("vdd1" "sig1")
            ("mask2Color" "mask3Color")
        )
        (
            (
                (0.04 0.08 nil "mask2Color")
                (0.09 0.012 "big")
            )
        )
    )
    0.007
    t
    nil
    "steppedOnly"
    "stepped"
)
```

In this example, a width spacing pattern `wsp1` is created in the cellview `cv` with two track groups, defining a total of six tracks.

- For the first track group:
  - The width of the track is `0.04`.

© 2020 All Rights Reserved.

- The distance of the next track from the current track, in the period direction is `0.08`.
- The track group is repeated `4` times resulting in a total of 4 tracks.
- The offset distance of the first track from the period track is `4`.
- The wire type of the first and third track is `vdd1`. The wire type of the second and fourth track is `sig1`.
- The track color of the first and third track is `mask2Color`. The track color of the second and fourth track is `mask3Color`.

- For the second track group:
  - For the first track in this group:
    - The width is `0.04`.
    - The distance of the next track from this track, in the period direction is `0.08`.
    - No wire type is specified.
    - The track color is `mask2Color`.
  - For the second track in this group and the sixth track in the pattern:
    - The width is `0.09`.
    - The distance of the next track from this track, in the period direction is `0.012`.
    - The wire type is `big`.
    - No track color is specified.
  - The second track group is not repeated, and the wire type and track color of each track are specified on individual tracks.
- The distance of the first track from the period grid is 0.007.
- The `repeatOffset` attribute is set to `t`. This means the height of the pattern includes the offset of the first track.
- The `shiftColor` attribute is set to `nil`. This means the track colors are not shifted when the pattern repeats.
- *`allowedRepeatMode`* is `steppedOnly`, which implies that the pattern can only be stepped, but not flipped. In conformance, *`defaultRepeatMode`* is `stepped`.

© 2020 All Rights Reserved.

dbCreateWidthSpacingPatternWithColor

```
dbCreateWidthSpacingPatternWithColor(
     d_cellView
     t_name
     l_patternSpecs
     t_startingColor
     [ n_offset ]
     [ g_repeatOffset ]
     [ t_allowedRepeatMode ]
     [ t_defaultRepeatMode ]
     )
     => d_widthSpacingPattern / nil
```

Description

(ICADVM20.1 Only) Creates a width spacing pattern in the specified cellview. This function requires you to specify the color for only the first track, referred to as the starting color. All other tracks are colored automatically by shifting colors.

Arguments

d_cellView	The ID of the cellview for which the width spacing pattern must be created.
t_name	Name of the width spacing pattern.

© 2020 All Rights Reserved.

l_patternSpecs

`list(`*l_patternSpec*`)`
List of track group specifications.

where,

- *l_patternSpec*: `list(`*l_specs* [*d_repeat* [*l_wireTypes*] [*l_displayPacketNames*]`)`
  A track group specification.
  - *l_specs*: `list(`*l_spec*`)`
    List of individual track specifications for the specified track group.
    - *l_spec*: `list(`*g_width g_space* [*t_wireType*] [*t_displayPacketName*]`)`
      A track specification.

      g_width
      Width of the shapes for the specified track.

      g_space
      The distance of the next track from the current track, in the period direction.

      t_wireType
      The type of wire. This parameter can be specified only when *d_repeat l_wireTypes* is not specified. It is used to refer to a specific group of tracks, for example, routing a set of tracks by name. The default value is an empty string `""`.

      t_displayPacketName
      The name of a display packet that can be specified to control how the track should be drawn on the screen.
  - *d_repeat*
    The number of times the track group is repeated. A value of 1 creates the track group from the specified *l_specs* (1x). A value of 2 creates the track group from the specified *l_specs* that are repeated once (2x).

© 2020 All Rights Reserved.

- *l_wireTypes*: list([*t_wireType* ...])
  List of wire types associated with the specified track group (only if *d_repeat* is specified on the track group). Use an empty string ("") to specify no wire type for an individual track. If the number of tracks in the track group is greater than the number of wire types in this list, then the wire types list is repeated across the track group until all tracks are assigned a wire type, one-by-one.

  - *t_wireType*
    The type of wire.

  For example, for a track group with 6 tracks:

  - If *l_wireTypes* is list("a" "b"), then tracks 1, 3, and 5 are wire type **a**; tracks 2, 4, and 6 are wire type **b**.
  - If *l_wireTypes* is list("a" "a" "b"), then tracks 1, 2, 4, and 5 are wire type **a**; tracks 3 and 6 are wire type **b**.

- *l_displayPacketNames*
  List of display packet names for the track group (only if *d_repeat* is specified).

  Default value: An empty list.

  l_displayPacketName: list([*t_displaypacketName* …])

  t_displayPacketName is the display packet name for the specified track.

t_startingColor — Color of the first track. All tracks are automatically colored by color shifting from the previous track color. Valid values are "mask1Color", "mask2Color", "mask3Color", and "grayColor". Use "grayColor" to make all tracks uncolored.

n_offset — The distance of the first track from the period track.

© 2020 All Rights Reserved.

g_repeatOffset	Determines whether the offset is applied when a pattern repeats. When set to `t` the offset is repeated every time and when set to `nil` the offset is applied only once.
t_allowedRepeatMode	Name of allowed repeat mode. Valid values are "`any`", "`none`", "`steppedOnly`" and "`flippedOnly`". The default value is "`any`".
t_defaultRepeatMode	Name of default repeat mode. Valid values are "`noRepeat`", "`stepped`", "`flippedStartsWithOdd`" and "`flippedStartsWithEven`". The default value is "`noRepeat`".

Value Returned

d_widthSpacingPattern	The ID of the created width spacing pattern.
`nil`	Returned in case of failure.

Examples

```
dbCreateWidthSpacingPatternWithColor(cv "wscp" '((((0.04 0.08 "mywire1"
"mask1Color") (0.09 0.012)))) "mask2Color" 0.05 t "steppedOnly" "stepped")
```

In the above example, a width spacing pattern `wsp` is created in the cellview `cv` with two track specifications.

- For the first track,
  - `0.04` is the width of the shapes.
  - `0.08` is the distance of the next track from the current track, in the period direction.
  - `mywire1` is the type of wire to use.
  - `0.05` is the distance of the first track from the period track.
- For the second track,
  - 0.09 is the width of the shapes.
  - 0.12 is the distance of the next track from the current track, in the period direction.
- The starting color is `mask2Color`.

© 2020 All Rights Reserved.

- The following figure shows the period when the number of masks on the layer is 2 (`mask1Color` and `mask2Color`).

- The allowed repeat mode is `steppedOnly`, meaning the pattern can only be stepped, but not flipped.
- The default repeat mode is `stepped`, which conforms to the allowed repeat mode of `steppedOnly`.

© 2020 All Rights Reserved.

dbFindWidthSpacingPattern

```
dbFindWidthSpacingPattern(
     d_cellView
     t_name
     )
     => d_widthSpacingPattern / nil
```

Description

(ICADVM20.1 Only) Searches for the width spacing pattern with the given name in the specified cellview and technology library.

Arguments

d_cellView	The ID of the cellview in which the search is performed.
t_name	Name of the width spacing pattern for which the search is performed.

Value Returned

d_widthSpacingPattern	The ID of the specified width spacing pattern, if found.
nil	Returned in case of failure.

Example

```
dbFindWidthSpacingPattern(cv "wsp1")
```


© 2020 All Rights Reserved.

dbGetWidthSpacingPatternAllowedRepeatMode

```
dbGetWidthSpacingPatternAllowedRepeatMode(
     d_wspId
     )
     => t_allowedRepeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the allowed repeat mode for the specified width spacing pattern.

Arguments

d_wspId	ID of a width spacing pattern.

Value Returned

t_allowedRepeatMode	Value of the allowed repeat mode.
nil	Returned in case of failure.

Example

```
dbGetWidthSpacingPatternAllowedRepeatMode(wspId)
```

Returns the allowed repeat mode for the width spacing pattern `wspId`.

© 2020 All Rights Reserved.

dbGetWidthSpacingPatternDefaultRepeatMode

```
dbGetWidthSpacingPatternDefaultRepeatMode(
     d_wspId
     )
     => t_defaultRepeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the default repeat mode of the specified width spacing pattern.

Arguments

d_wspId	ID of a width spacing pattern.

Value Returned

t_defaultRepeatMode	Value of the default repeat mode.
nil	Returned in case of failure.

Example

```
dbGetWidthSpacingPatternDefaultRepeatMode(wspId)
```

Returns default repeat mode of the width spacing pattern `wspId`.

© 2020 All Rights Reserved.

dbGetWidthSpacingPatterns

```
dbGetWidthSpacingPatterns(
     d_cellView
     )
     => l_widthSpacingPatterns / nil
```

Description

(ICADVM20.1 Only) Returns a list of all width spacing pattern IDs in the specified cellview.

Arguments

d_cellView	The ID of the cellview.

Value Returned

l_widthSpacingPatterns	The list of width spacing pattern IDs.
nil	Returned in case of failure.

Example

```
dbGetWidthSpacingPatterns(cv)
=> ("wsp1")
```


© 2020 All Rights Reserved.

dbSetWidthSpacingPatternRepeatMode

```
dbSetWidthSpacingPatternRepeatMode(
     d_wspId
     t_allowedRepeatMode
     t_defaultRepeatMode
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets a valid combination of allowed and default repeat modes for the specified width spacing pattern.

Arguments

d_wspId	ID of a width spacing pattern.
t_allowedRepeatMode	Name of the allowed repeat mode. Valid values: `any`, `none`, `steppedOnly`, `flippedOnly`
t_defaultRepeatMode	Name of the default repeat mode. Valid values: `noRepeat`, `stepped`, `flippedStartsWithOdd`, `flippedStartsWithEven`

Value Returned

`t`	A valid combination of allowed and default repeat modes was set successfully on the specified width spacing pattern.
`nil`	Returned in case of failure.

Example

```
dbSetWidthSpacingPatternRepeatMode(wspId "flippedOnly" "flippedStartsWithOdd ")
```

Sets the combination of allowed repeat mode as `flippedOnly` and default repeat mode as `flippedStartsWithOdd` for the width spacing pattern `wspId`.

© 2020 All Rights Reserved.

Width Spacing Pattern Group Functions

Supported functions include:

- dbCreateWidthSpacingPatternGroup
- dbGetWidthSpacingPatternGroups
- dbFindWidthSpacingPatternGroup

© 2020 All Rights Reserved.

dbCreateWidthSpacingPatternGroup

```
dbCreateWidthSpacingPatternGroup(
     d_cellView
     t_name
     l_patternNames
     )
     => d_widthSpacingPatternGroup / nil
```

Description

(ICADVM20.1 Only) Creates a group of width spacing patterns in the specified cellview.

Arguments

d_cellView	The ID of the cellview in which the width spacing pattern group is created.
t_name	Name of the width spacing pattern group.
l_patternNames	`list (t_patternName)` List of the width spacing patterns that must be included in the group.

Value Returned

d_widthSpacingPatternGroup	The ID of the created width spacing pattern group.
`nil`	Returned in case of failure.

Example

```
dbCreateWidthSpacingPatternGroup(cv "group1" '("wsp1" "wsp2"))
```


© 2020 All Rights Reserved.

dbGetWidthSpacingPatternGroups

```
dbGetWidthSpacingPatternGroups(
     d_cellView
     )
     => l_widthSpacingPatternGroups / nil
```

Description

(ICADVM20.1 Only) Returns a list of all width spacing pattern group IDs from the specified cellview.

Arguments

d_cellView	The ID of the cellview.

Value Returned

l_widthSpacingPatternGroups	List of width spacing pattern group IDs.
nil	Returned in case of failure.

Example

```
dbGetWidthSpacingPatternGroups(cv)
=> ("wsp1")
```


© 2020 All Rights Reserved.

dbFindWidthSpacingPatternGroup

```
dbFindWidthSpacingPatternGroup(
     d_cellView
     t_name
     )
     => d_widthSpacingPatternGroup / nil
```

Description

(ICADVM20.1 Only) Searches for the width spacing pattern group with the specified name in the given cellview and technology library.

Arguments

d_cellView	The ID of the cellview in which the search is performed.
t_name	Name of the width spacing pattern group for which the search is performed.

Value Returned

d_widthSpacingPatternGroup	ID of the specified width spacing pattern group, if found.
nil	Returned in case of failure.

Example

```
dbFindWidthSpacingPatternGroup(cv "wsp_group1")
```


© 2020 All Rights Reserved.

Width Spacing Snap Pattern Def Functions

Supported functions include:

- dbCreateWidthSpacingSnapPatternDef
- dbFindWidthSpacingSnapPatternDefByName
- dbDeleteWidthSpacingSnapPatternDef
- dbGetWidthSpacingSnapPatternDefsByLP

Note: For information on Active Wire Type functions related to widthSpacingSnapPatternDef object, see Active Wire Types Functions.

© 2020 All Rights Reserved.

dbCreateWidthSpacingSnapPatternDef

```
dbCreateWidthSpacingSnapPatternDef(
     d_cellViewId
     t_name
     (tx_layer [tx_purpose])
     t_direction
     n_period
     t_defaultActivePattern
     [ l_snappingLayers ]
     [ n_offset ]
     [ l_patternNames ]
     [ l_patternGroupNames ]
     [ g_gridType ]
     )
     => widthSpacingSnapPatternDefId / nil
```

Description

(ICADVM20.1 Only) Creates a widthSpacingSnapPatternDef object in the specified cellview.

Arguments

d_cellviewId	ID of the cellview in which the widthSpacingSnapPatternDef object is created.
t_name	Name of the widthSpacingSnapPatternDef object.
(tx_layer [tx_purpose])	The layer-purpose pair.
t_direction	The pattern direction. Valid values are `horizontal` and `vertical`.
g_period	The spacing between the coarse-grain period tracks (heart-beat period).
t_defaultActivePattern	The pattern used in the areas in which no regions have been drawn.

© 2020 All Rights Reserved.

l_snappingLayers

`list(l_snappingLayer)`
The list of snapping layer definitions, where:

- `l_snappingLayer: list(t_layer [l_purposeList])`
  The snapping layer definition, where:
  - `t_layer`
    The name of the snapping layer.
- `l_purposeList: list(t_purpose)`
  The list of purpose names.

g_offset

The distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis). Default value is `0`.

l_patternNames

`list(t_patternName)`
The list of the allowed width spacing pattern names.

l_patternGroupNames

`list(t_patternGroupName)`
The list of the allowed width spacing pattern group names.

g_gridType

The WSP grid to use for line-ends. The following values are valid:

- `upperLower`: Applies to both ends of the wires.
- `lower`: Applies to the lower line-ends of vertical wires and the left line-ends of horizontal wires.
- `upper`: Applies to the upper line-ends of vertical wires and the right line-ends of horizontal wires.
- `nil`: The default value, which is used when this argument is not specified. In this case, the line-end grid is not configured for the widthSpacingSnapPatternDef object.

© 2020 All Rights Reserved.

Value Returned

d_widthSpacingSnapPatternDefId	The ID of the widthSpacingSnapPatternDef object that was created.
nil	The widthSpacingSnapPatternDef object was not created.

Example

```
dbCreateWidthSpacingSnapPatternDef(
cellView
"wsspDef"
'("layer1"  "purpose1")
"vertical"
0.2
"wsp3"
'(("layer2" ("dummy1" "dummy2")) ("layer3" ("dummy3" "dummy4")))
0.1
'("wsp" "wsp2")
'("basePatterns" "wspg2"))
```

Creates a widthSpacingSnapPatternDef object `wsspDef` defined by the layer-purpose pair `layer1-purpose1` in the `vertical` direction with space `0.2` and offset `0.1`.

© 2020 All Rights Reserved.

dbFindWidthSpacingSnapPatternDefByName

```
dbFindWidthSpacingSnapPatternDefByName(
     d_cellViewId
     t_name
     )
     => d_widthSpacingSnapPatternDefId / nil
```

Description

(ICADVM20.1 Only) Searches for the given widthSpacingSnapPatternDef object in the specified cellview.

Arguments

d_cellviewId	The ID of the cellview in which the search is performed.
t_name	Name of the widthSpacingSnapPatternDef object for which the search is performed.

Value Returned

d_widthSpacingSnapPatternDefId	The ID of the widthSpacingSnapPatternDef object that was found.
`nil`	The widthSpacingSnapPatternDef object was not found.

Example

```
dbFindWidthSpacingSnapPatternDefByName(cv "wsspDef")
```

Searches for the widthSpacingSnapPatternDef object `wsspDef` in the cellview `cv`.

© 2020 All Rights Reserved.

dbDeleteWidthSpacingSnapPatternDef

```
dbDeleteWidthSpacingSnapPatternDef(
     d_dbWidthSpacingSnapPatternDefId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Deletes the specified widthSpacingSnapPatternDef object.

Arguments

d_dbWidthSpacingSnapPatternDefId

The ID of the widthSpacingSnapPatternDef object.

Value Returned

`t`	The object was successfully deleted.
`nil`	The object was not deleted.

Example

```
dbDeleteWidthSpacingSnapPatternDef(wsspDef)
```

Deletes the widthSpacingSnapPatternDef object `wsspDef`.

© 2020 All Rights Reserved.

dbGetWidthSpacingSnapPatternDefsByLP

```
dbGetWidthSpacingSnapPatternDefsByLP(
     d_cellViewId
     l_LP
     )
     => l_widthSpacingSnapPatternDefIDList / nil
```

Description

(ICADVM20.1 Only) Returns all the widthSpacingSnapPatternDefs on a layer-purpose pair.

Arguments

d_cellViewId	ID of a cellview.
l_LP	List of layer-purpose names. `list(t_layer t_purpose)`

Return value

l_widthSpacingSnapPatternDefIDList

List of widthSpacingSnapPatternDef IDs on the given layer-purpose pair.

`nil` Returned in case of a failure.

Examples

```
wSSPDef = dbGetWidthSpacingSnapPatternDefsByLP(cv list("Metal2" "localWSP"))
> (db:0x1872da1a db:0x1872da1c)
```

Returns the database IDs for all the widthSpacingSnapPatternDefs defined in the specified cellview `cv` for layer `Metal2` and purpose `localWSP`.

```
dbGetWidthSpacingSnapPatternDefsByLP(cv list("Metal2" "localWSP"))~>name
>("M2WSP" "M2WSP_double")
```

Returns all the widthSpacingSnapPatternDefs defined in the specified cellview `cv` for layer `Metal2` and purpose `localWSP`.

© 2020 All Rights Reserved.

Related Snap Pattern Functions

Supported functions include:

- dbCreateRelatedSnapPatterns
- dbGetRelatedSnapPatterns
- dbFindRelatedSnapPatterns

© 2020 All Rights Reserved.

dbCreateRelatedSnapPatterns

```
dbCreateRelatedSnapPatterns(
     d_cellView
     t_name
     l_relatedSnapPatterns
     [l_rowRegionSpecNames]
     [l_extraLPP]
     [f_regionSnapPitchHorizontal]
     [f_regionSnapPitchVertical]
     )
     => d_relatedSnapPatterns / nil
```

Description

(ICADVM20.1 Only) Creates a group of related snap patterns in the specified cellview.

Arguments

d_cellView	The ID of the cellview in which the related snap patterns are to be created.
t_name	Name of the group of the related snap patterns.
l_relatedSnapPatterns	`list(`*l_snapPatternDef*`)` List of the snap pattern definitions to include in the group. where, ■ *l_snapPatternDef*: `list(`*t_snapPatternDefName* [*l_widthSpacingPatternNames*] [*l_widthSpacingPatternGroupNames*]`)` A snap pattern definition. ❑ *t_snapPatternDefName*: Name of the snap pattern definition. ❑ *l_widthSpacingPatternNames*: `list(`*t_widthSpacingPatternName*`)` List of the width spacing pattern names allowed in this definition. These patterns can belong either to a tech database or to a design database.

© 2020 All Rights Reserved.

- *l_widthSpacingPatternGroupNames*: list (*t_widthSpacingPatternGroupName*)
  List of the width spacing pattern group names allowed in the definition. These pattern groups can belong either to a tech database or to a design database.

l_rowRegionSpecNames list(*t_rowRegionSpecName...*)
List of the names of RowRegionSpecs to include in the group.

l_extraLPP An additional layer-purpose pair. A shape on the LPP is created automatically when an instance of the related snap pattern is placed in the layout.

f_regionSnapPitchHorizontal

The pitch at which the horizontal region edges snap in the vertical direction. This value overrides the X snapping grid values.

The default value is 0.

f_regionSnapPitchVertical

The pitch at which the vertical region edges snap in the horizontal direction. This value overrides the Y snapping grid values.

The default value is 0.

Value Returned

d_relatedSnapPatterns The ID of the created group of related snap patterns.

nil Returned in case of failure.

Example

```
dbCreateRelatedSnapPatterns(cv "rsp_group1" '(("snap1" nil ("wsp_group1"))
("snap2" ("wsp1"))) nil '("Active" "drawing") 0.1 0.2 )
```


© 2020 All Rights Reserved.

dbGetRelatedSnapPatterns

```
dbGetRelatedSnapPatterns(
     d_cellView
     )
     => l_relatedSnapPatterns / nil
```

Description

(ICADVM20.1 Only) Returns a list of related snap pattern IDs from the specified cellview.

Arguments

d_cellView	The ID of the cellview.

Value Returned

l_relatedSnapPatterns	List of related snap pattern IDs.
nil	Returned in case of failure.

Example

```
dbGetRelatedSnapPatterns(cv)
=> ("rsp1")
```


© 2020 All Rights Reserved.

dbFindRelatedSnapPatterns

```
dbFindRelatedSnapPatterns(
     d_cellView
     t_name
     )
     => d_relatedSnapPatterns / nil
```

Description

(ICADVM20.1 Only) Searches for the group of related snap patterns with the specified name in the given cellview and technology library.

Arguments

d_cellView	The ID of the cellview in which the search is performed.
t_name	Name of the group of related snap patterns for which a search is to be performed.

Value Returned

d_relatedSnapPatterns	The ID of the specified related snap patterns group, if found.
nil	Returned in case of failure.

Example

```
dbFindRelatedSnapPatterns(cv "rsp1")
```


© 2020 All Rights Reserved.

Pattern Region Functions

Supported functions include:

- dbGetPatternRegionActivePattern
- dbSetPatternRegionActivePattern
- dbHasPatternRegionActivePattern
- dbGetPatternRegionAllowedPatterns
- dbSetPatternRegionAllowedPatterns
- dbGetPatternRegionAllowedPatternGroups
- dbSetPatternRegionAllowedPatternGroups
- dbGetPatternRegionCommittedPatterns
- dbGetPatternRegionCommittedPatternShiftColor
- dbGetPatternRegionCreateType
- dbGetPatternRegionShiftColor
- dbSetPatternRegionShiftColor
- dbHasPatternRegionInfo
- dbGetPatternRegionName
- dbSetPatternRegionName
- dbGetPatternRegionWidthSpacingSnapPatternDefName
- dbSetPatternRegionWidthSpacingSnapPatternDefName
- dbGetPatternRegionDefaultRepeatMode
- dbSetPatternRegionDefaultRepeatMode
- dbGetPatternRegionRepeatMode
- dbSetPatternRegionRepeatMode

Note: For information on Active Wire Type functions related to the pattern region object, see Active Wire Types Functions.

© 2020 All Rights Reserved.

dbGetPatternRegionActivePattern

```
dbGetPatternRegionActivePattern(
     d_shapeId
     )
     => t_patternName / nil
```

Description

(ICADVM20.1 Only) Returns the name of the currently set active pattern for the specified shape.

Arguments

d_shapeId	The ID of the shape.

Value Returned

t_patternName	The name of the active pattern.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionActivePattern(rect)
=> "wsp1"
dbGetPatternRegionActivePattern(rect1)
=> nil
```


© 2020 All Rights Reserved.

dbSetPatternRegionActivePattern

```
dbSetPatternRegionActivePattern(
     d_shapeId
     t_patternName
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the given active pattern for the specified shape.

Arguments

d_shapeId	The ID of the shape.
t_patternName	The name of the active pattern.

Value Returned

`t`	The active pattern was set successfully for the specified shape.
`nil`	Returned in case of failure.

Example

```
dbSetPatternRegionActivePattern(rect "wsp1")
=> t
dbGetPatternRegionActivePattern(rect)
=> "wsp1"
```


© 2020 All Rights Reserved.

dbHasPatternRegionActivePattern

```
dbHasPatternRegionActivePattern(
     d_shapeId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Determines whether the specified shape has an associated active pattern.

Arguments

d_shapeId	The ID of the shape.

Value Returned

t	The specified shape has an associated active pattern.
nil	Returned in case of failure.

Example

```
dbHasPatternRegionActivePattern(rect)
=> nil
dbSetPatternRegionActivePattern(rect "wsp1")
=> t
dbHasPatternRegionActivePattern(rect)
=> t
```


© 2020 All Rights Reserved.

dbGetPatternRegionAllowedPatterns

```
dbGetPatternRegionAllowedPatterns(
     d_shapeId
     )
     => l_patternNameList / nil
```

Description

(ICADVM20.1 Only) Returns a list of allowed pattern names for the specified shape.

Arguments

d_shapeId	The ID of the shape.

Value Returned

l_patternNameList	The list of allowed pattern names.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionAllowedPatterns(rect)
=> ("wsp1" "wsp2")
```


© 2020 All Rights Reserved.

dbSetPatternRegionAllowedPatterns

```
dbSetPatternRegionAllowedPatterns(
     d_shapeId
     l_patternNameList
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the given allowed patterns for the specified shape.

Arguments

d_shapeId	The ID of the shape.
l_patternNameList	The list of allowed patterns.

Value Returned

`t`	The allowed patterns were set successfully for the specified shape.
`nil`	Returned in case of failure.

Example

```
dbSetPatternRegionAllowedPatterns(rect '("wsp1" "wsp2"))
=> t
```


© 2020 All Rights Reserved.

dbGetPatternRegionAllowedPatternGroups

```
dbGetPatternRegionAllowedPatternGroups(
     d_shapeId
     )
     => l_groupNameList / nil
```

Description

(ICADVM20.1 Only) Returns a list of allowed pattern groups for the specified shape.

Arguments

d_shapeId	The ID of the shape.

Value Returned

l_groupNameList	The list of allowed pattern groups.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionAllowedPatternGroups(rect)
=> ("wspg1" "wspg2")
```


© 2020 All Rights Reserved.

dbSetPatternRegionAllowedPatternGroups

```
dbSetPatternRegionAllowedPatternGroups(
     d_shapeId
     l_groupNameList
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the given allowed pattern groups for the specified shape.

Arguments

d_shapeId	The ID of the shape.
l_groupNameList	The list of the allowed pattern groups.

Value Returned

`t`	The allowed pattern groups were set successfully for the specified shape.
`nil`	Returned in case of failure.

Example

```
dbSetPatternRegionAllowedPatternGroups(rect '("wspg1" "wspg2"))
=> t
```


© 2020 All Rights Reserved.

dbGetPatternRegionCommittedPatterns

```
dbGetPatternRegionCommittedPatterns(
     d_shapeId
     )
     => l_patternNameList / nil
```

Description

(ICADVM20.1 Only) Returns a list of committed pattern names for the specified shape.

Arguments

d_shapeId	The ID of the shape.

Value Returned

l_patternNameList	The list of committed pattern names.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionCommittedPatterns(rect)
=> ("wsp1")
```


© 2020 All Rights Reserved.

dbGetPatternRegionCommittedPatternShiftColor

```
dbGetPatternRegionCommittedPatternShiftColor(
     d_shapeId
     t_patternName
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Determines whether a specific committed pattern was committed with or without a color shift.

Arguments

d_shapeId	The ID of the shape.
t_patternName	The name of the committed pattern.

Value Returned

`t`	The specified pattern was committed with color shift.
`nil`	Returned in case of failure.

Example

```
dbGetPatternRegionCommittedPatternShiftColor(rect "wsp1")
=> nil
```


© 2020 All Rights Reserved.

dbGetPatternRegionCreateType

```
dbGetPatternRegionCreateType(
     d_shapeId
     )
     => t_createType / nil
```

Description

(ICADVM20.1 Only) Returns the create type of the specified shape. Possible values of create type are `systemCreated` and `userCreated`.

Arguments

d_shapeId	The ID of the shape.

Value Returned

t_createType	The create type.
`nil`	Returned in case of failure.

Example

```
dbGetPatternRegionCreateType(rect)
=> "userCreated"
```


© 2020 All Rights Reserved.

dbGetPatternRegionShiftColor

```
dbGetPatternRegionShiftColor(
     d_shapeId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Determines whether shift color is enabled for the specified shape.

Arguments

d_shapeId	The ID of the shape.

Value Returned

`t`	The shift color is enabled.
`nil`	Returned in case of failure.

Example

```
dbGetPatternRegionShiftColor(rect)
=> nil
```


© 2020 All Rights Reserved.

dbSetPatternRegionShiftColor

```
dbSetPatternRegionShiftColor(
     d_shapeId
     g_shiftColor
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Determines whether shift color is enabled for the specified shape.

Arguments

d_shapeId	The ID of the shape.
g_shiftColor	The shift color boolean flag.

Value Returned

`t`	Shift color is successfully set for the specified shape.
`nil`	Returned in case of failure.

Example

```
dbGetPatternRegionShiftColor(rect)
=> nil
```


© 2020 All Rights Reserved.

dbHasPatternRegionInfo

```
dbHasPatternRegionInfo(
     d_shapeId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Determines whether the specified shape contains pattern region information.

Arguments

d_shapeId	The ID of the shape.

Value Returned

t	The specified shape contains pattern region information.
nil	Returned in case of failure.

Example

```
dbHasPatternRegionInfo(rect)
=> nil
dbSetPatternRegionAllowedPatterns(rect '("wsp1" "wsp2"))
=> t
dbHasPatternRegionInfo(rect)
=> t
```


© 2020 All Rights Reserved.

dbGetPatternRegionName

```
dbGetPatternRegionName(
     d_shapeId
     )
     => t_name / nil
```

Description

(ICADVM20.1 Only) Returns the name of the pattern region set for the specified shape.

Arguments

d_shapeId	The ID of the shape.

Value Returned

t_name	The name of the pattern region.
nil	Returned in case of failure.

Example

```
rect = dbCreateRect(cv '("Metal1" "drawing") '((0 0) (10 10)))
> db:0x2c4be89e
dbHasPatternRegionInfo(rect)
> nil
dbGetPatternRegionName(rect)
> *WARNING* (DB-220061): dbGetPatternRegionName: This shape ID has no pattern
region info.
> nil
```

The shape `rect` does not contain pattern region information.

```
dbSetPatternRegionAllowedPatterns(rect '("1X"))
> t
dbHasPatternRegionInfo(rect)
> t
```

Pattern region information is assigned to the shape `rect`.

```
dbGetPatternRegionName(rect)
```


© 2020 All Rights Reserved.

```
> ""
```

The default name of a pattern region is an empty string.

```
dbSetPatternRegionName(rect "rect")
> t
dbGetPatternRegionName(rect)
> "rect"
```

The pattern region `"rect"` has been set for the specified shape.

© 2020 All Rights Reserved.

dbSetPatternRegionName

```
dbSetPatternRegionName(
     d_shapeId
     t_name
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the specified pattern region for the specified shape.

Arguments

d_shapeId	The ID of the shape.
t_name	The name of the pattern region.

Value Returned

`t`	The specified pattern region is set.
`nil`	Returned in case of failure.

Example

```
rect = dbCreateRect(cv '("Metal1" "drawing") '((0 0) (10 10)))
> db:0x2c4be89e
dbHasPatternRegionInfo(rect)
> nil
dbGetPatternRegionName(rect)
> *WARNING* (DB-220061): dbGetPatternRegionName: This shape ID has no pattern
region info.
> nil
dbSetPatternRegionName(rect "rect")
> *WARNING* (DB-220061): dbSetPatternRegionName: This shape ID has no pattern
region info.
> nil
```

The shape `rect` does not contain pattern region information.

```
dbSetPatternRegionAllowedPatterns(rect '("1X"))
> t
```


© 2020 All Rights Reserved.

```
dbHasPatternRegionInfo(rect)
> t
```

Pattern region information is assigned to the shape `rect`.

```
dbGetPatternRegionName(rect)
> ""
```

The default name of a pattern region is an empty string.

```
dbSetPatternRegionName(rect "rect")
> t
dbGetPatternRegionName(rect)
> "rect"
```

The pattern region `rect` has been set for the specified shape.

© 2020 All Rights Reserved.

dbGetPatternRegionWidthSpacingSnapPatternDefName

```
dbGetPatternRegionWidthSpacingSnapPatternDefName(
     d_dbShapeId
     )
     => t_defName / nil
```

Description

(ICADVM20.1 Only) Returns the widthSpacingSnapPatternDef name from the specified pattern region.

Argument

d_dbShapeId	Database ID of a pattern region.

Return values

t_defName	Returned if a widthSpacingSnapPatternDef name was set successfully on the pattern region.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionWidthSpacingSnapPatternDefName(region)
> ""
```

Returns `""` because widthSpacingSnapPatternDef name is not set on the pattern region `region`.

```
dbSetPatternRegionWidthSpacingSnapPatternDefName(region "wSSPDef")
> t
```

WidthSpacingSnapPatternDef name `wSSPDef` is set on the pattern region `region`.

```
dbGetPatternRegionWidthSpacingSnapPatternDefName(region)
> "wSSPDef"
```

Returns `wSSPDef` as the widthSpacingSnapPatternDef name for the pattern region `region`.

© 2020 All Rights Reserved.

dbSetPatternRegionWidthSpacingSnapPatternDefName

```
dbSetPatternRegionWidthSpacingSnapPatternDefName(
     d_dbShapeId
     t_defName
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the widthSpacingSnapPatternDef name on the specified pattern region. The widthSpacingSnapPatternDef must already exist.

Arguments

d_dbShapeId	ID of a pattern region.
t_defName	Name of the widthSpacingSnapPatternDef to be stored for the pattern region.

Return value

`t`	Returned if the widthSpacingSnapPatternDef name has been set.
nil	Returned in case of failure.

Example

```
dbSetPatternRegionWidthSpacingSnapPatternDefName(region
"newWSSPDef")
```

Sets the widthSpacingSnapPatternDef name `newWSSPDef` on the pattern region `region`.

© 2020 All Rights Reserved.

dbGetPatternRegionDefaultRepeatMode

```
dbGetPatternRegionDefaultRepeatMode(
     d_patternRegion
     )
     => t_repeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the default repeat mode for the specified pattern region.

Arguments

d_patternRegion	ID of the pattern region for which the repeat mode is set.

Value Returned

t_repeatMode	The default repeat mode for the specified pattern region.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionDefaultRepeatMode(patternRegionId)
```

Returns the default repeat mode for the pattern region `patternRegionId`.

© 2020 All Rights Reserved.

dbSetPatternRegionDefaultRepeatMode

```
dbSetPatternRegionDefaultRepeatMode(
     d_patternRegion
     t_repeatMode
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets default repeat mode on the specified pattern region.

Arguments

d_patternRegion	ID of the pattern region for which the repeat mode is to be set.
t_repeatMode	Name of the repeat mode to be set. Valid values: `noRepeat`, `stepped`, `flippedStartsWithOdd`, `flippedStartsWithEven`

Value Returned

`t`	Specified repeat mode is successfully set on the pattern region.
`nil`	Returned in case of failure.

Example

```
dbSetPatternRegionDefaultRepeatMode(patternRegionId "stepped")
```

Sets the default repeat mode as `stepped` for the pattern region `patternRegionId`.

© 2020 All Rights Reserved.

dbGetPatternRegionRepeatMode

```
dbGetPatternRegionRepeatMode(
     d_patternRegion
     t_patternName
     )
     => t_repeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the repeat mode for the specified pattern region.

Arguments

d_patternRegion	ID of the pattern region for which the repeat mode is set.
t_patternName	Name of a width spacing pattern.

Return Value

t_repeatMode	The repeat mode for the specified pattern region.
`nil`	Returned in case of failure.

Example

```
dbGetPatternRegionRepeatMode(patternRegionId "wsp")
```

Returns the repeat mode for the width spacing pattern `wsp` in the pattern region `patternRegionId`.

© 2020 All Rights Reserved.

dbSetPatternRegionRepeatMode

```
dbSetPatternRegionRepeatMode(
     d_patternRegion
     t_patternName
     t_repeatMode
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the specified repeat mode on the pattern region for the specified width spacing pattern.

Arguments

d_patternRegion	ID of the pattern region on which the repeat mode is to be set.
t_patternName	Name of a width spacing pattern.
t_repeatMode	Name of the repeat mode to be set. Valid values: `noRepeat`, `stepped`, `flippedStartsWithOdd`, `flippedStartsWithEven`

Value Returned

`t`	Specified repeat mode is successfully set on the pattern region for the specified width spacing pattern.
`nil`	Returned in case of failure.

Example

```
dbSetPatternRegionRepeatMode(patternRegionId "wsp" "stepped")
```

In this example, repeat mode `stepped` is successfully set on the pattern region `patternRegionId` for the width spacing pattern `wsp`.

© 2020 All Rights Reserved.

Active Wire Types Functions

Width Spacing Patterns include wire types that can be defined per track pattern. Active Wire Types functions enable users and applications to specify tracks, which are active or not filtered by wire type, using one of the following:

- per-design database basis
- per-widthSpacingSnapPatternDef basis
- per-pattern region basis

These functions also provide the capability to clear the specifications. For example, it is required to be able to specify that no tracks are allowed and all are filtered on a pattern region.

Supported functions include:

Per-Design Database Basis Functions

- dbGetCellViewActiveWireTypes
- dbSetCellViewActiveWireTypes
- dbClearAllActiveWireTypes

Per-widthSpacingSnapPatternDef Basis Functions

- dbGetCellViewActiveWireTypesForWidthSpacingSnapPatternDef
- dbSetCellViewActiveWireTypesForWidthSpacingSnapPatternDef

Per-Pattern Region Basis Functions

- dbGetPatternRegionActiveWireTypes
- dbSetPatternRegionActiveWireTypes
- dbSetPatternRegionNoActiveWireTypes
- dbHasPatternRegionNoActiveWireTypes

© 2020 All Rights Reserved.

Per-Design Database Basis Functions

dbGetCellViewActiveWireTypes

```
dbGetCellViewActiveWireTypes(
     d_dbCellViewId
     )
     => l_activeWireTypeNames / nil
```

Description

(ICADVM20.1 Only) Returns the active wire types on the specified cellview.

Arguments

d_dbCellViewId	ID of the cellview where the active wire types are to be returned.

Value Returned

l_activeWireTypeNames	List of active wire types.
nil	Returned in case of failure.

Example

```
dbGetCellViewActiveWireTypes(cv)
```

Lists the active wire types in the cellview `cv`.

© 2020 All Rights Reserved.

dbSetCellViewActiveWireTypes

```
dbSetCellViewActiveWireTypes(
     d_dbCellViewId
     l_activeWireTypeNames
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the active wire types on the specified cellview.

Arguments

d_dbCellViewId	ID of the cellview where the active wire types is set.
l_activeWireTypeNames	`list(t_activeWireTypeNames)` List of active wire types to be set on the cellview.

Value Returned

`t`	The active wire types was set successfully for the specified cellview.
`nil`	Returned in case of failure.

Example

```
dbSetCellViewActiveWireTypes(cv list("wire1" "wire2" "wire3"))
```

Opens a cellview `cv`, and then, calls the `dbSetCellViewActiveWireTypes` function with a list of the active wire types `wire1`, `wire2` and `wire3` to be set on the cellview `cv`.

© 2020 All Rights Reserved.

dbClearAllActiveWireTypes

```
dbClearAllActiveWireTypes (
     d_dbCellViewId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Clears all the active wire type specifications on the design and for the widthSpacingSnapPatternDef. Additionally, clears all the region-specific active wire types.

Arguments

d_dbCellViewId	ID of the cellview where the active wire types are to be cleared.

Value Returned

`t`	The active wire specifications are successfully filtered.
`nil`	Returned in case of failure.

Example

```
dbClearAllActiveWireTypes(cv)
```

Clears the active wire types in the cellview `cv`.

© 2020 All Rights Reserved.

Per-widthSpacingSnapPatternDef Basis Functions

dbGetCellViewActiveWireTypesForWidthSpacingSnapPatternDef

```
dbGetCellViewActiveWireTypesForWidthSpacingSnapPatternDef (
     d_dbCellViewId
     t_widthSpacingSnapPatternDefName
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns the active wire types on the specified cellview for a given widthSpacingSnapPatternDef.

Arguments

d_dbCellViewId	ID of the cellview where the active wire types are to be returned.
t_widthSpacingSnapPatternDefName	Name of the widthSpacingSnapPatternDef in the cellview where the active wire types are to be returned.

Value Returned

l_activeWireTypeNames	List of active wire types.
nil	Returned in case of failure.

Example

```
dbGetCellViewActiveWireTypesForWidthSpacingSnapPatternDef(cv "wSSPD")
```

Lists the active wire types on the cellview `cv` for the widthSpacingSnapPatternDef `wSSPD`.

© 2020 All Rights Reserved.

dbSetCellViewActiveWireTypesForWidthSpacingSnapPatternDef

```
dbSetCellViewActiveWireTypesForWidthSpacingSnapPatternDef(
     d_dbCellViewId
     t_widthSpacingSnapPatternDefName
     l_activeWireTypeNames
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the active wire types on the specified cellview for a given widthSpacingSnapPatternDef.

Arguments

d_dbCellViewId	ID of the cellview where the active wire types are to be set.
t_widthSpacingSnapPatternDefName	Name of the widthSpacingSnapPatternDef in the cellview where the active wire types are to be set.
l_activeWireTypeNames	`list(l_activeWireTypeName)` List of active wire types.

Value Returned

`t`	The active wire types was set successfully on the specified cellview for the given widthSpacingSnapPatternDef.
`nil`	Returned in case of failure.

Example

```
dbSetCellViewActiveWireTypesForWidthSpacingSnapPatternDef(cv "wSSPD" list("wire1"
"wire2" "wire3"))
```

Opens a cellview `cv` with a widthSpacingSnapPatternDef named `wSSPD` and the list of active wire types `wire1`, `wire2`, and `wire3` on it.

© 2020 All Rights Reserved.

Per-Pattern Region Basis Functions

dbGetPatternRegionActiveWireTypes

```
dbGetPatternRegionActiveWireTypes (
     d_PatternRegion
     )
     => l_activeWireTypeNames / nil
```

Description

(ICADVM20.1 Only) Returns the active wire types set on the specified pattern region.

Arguments

d_PatternRegion	ID of the pattern region where the active wire types are returned.

Value Returned

l_activeWireTypeNames	List of active wire types.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionActiveWireTypes(shape)
```

Lists the active wire types set on the pattern region `shape`.

© 2020 All Rights Reserved.

dbSetPatternRegionActiveWireTypes

```
dbSetPatternRegionActiveWireTypes(
     d_PatternRegion
     l_activeWireTypeNames
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the active wire types on the specified pattern region.

Arguments

d_PatternRegion	ID of the pattern region where the active wire types are to be set.
l_activeWireTypeNames	`list(t_activeWireTypeNames)` List of active wire types to be set on the pattern region.

Value Returned

`t`	The active wire types are set on the specified pattern region.
`nil`	Returned in case of failure.

Example

```
dbSetPatternRegionActiveWireTypes(shape list("wire1" "wire2" "wire3"))
```

Sets the active wire types `wire1`, `wire2` and `wire3` on the pattern region `shape`.

© 2020 All Rights Reserved.

dbSetPatternRegionNoActiveWireTypes

```
dbSetPatternRegionNoActiveWireTypes (
     d_PatternRegion
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets no active wire types on the specified pattern region.

Arguments

d_PatternRegion	ID of the pattern region where the no active wire types are to be set.

Value Returned

`t`	No active wire types is set on the specified pattern region.
`nil`	Returned in case of failure.

Example

```
dbSetPatternRegionNoActiveWireTypes(shape)
```

Sets no active wire types on the pattern region `shape`.

© 2020 All Rights Reserved.

dbHasPatternRegionNoActiveWireTypes

```
dbHasPatternRegionNoActiveWireTypes (
     d_PatternRegion
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Returns true if no active wire types is set on the specified pattern region.

Arguments

d_PatternRegion	ID of the PatternRegion where no active wire types are set.

Value Returned

`t`	Returned in case of success.
`nil`	Returned in case of failure.

Example

```
dbHasPatternRegionNoActiveWireTypes(shape)
```

Returns true if no active wire types is set on the specified pattern region `shape`.

© 2020 All Rights Reserved.

Global Grid Offset Functions

The supported functions include:

- dbSetGlobalGridOffsetReferenceType
- dbGetGlobalGridOffsetReferenceType
- dbGetGlobalGridDefaultRepeatMode
- dbSetGlobalGridDefaultRepeatMode
- dbGetGlobalGridRepeatMode
- dbSetGlobalGridRepeatMode

© 2020 All Rights Reserved.

dbSetGlobalGridOffsetReferenceType

```
dbSetGlobalGridOffsetReferenceType (
     d_dbCellViewId
     t_referenceType
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the global grid offset reference type on a cellview for the snapPatternDef and widthSpacingSnapPatternDef global grids. The global grid offset reference type can be `boundary` or `origin`.

- **Boundary**: The global grid offset is relative to the lower (left) edge of the cellview boundary for global grids with vertical (horizontal) direction. The cellview boundary is determined by a Snap Boundary, or, if no Snap Boundary exists, by a PRBoundary. If the cellview boundary is moved, the global grid will move accordingly.
- **Origin**: The global grid offset is relative to the x axis (y axis) of the cellview for global grids with vertical (horizontal) direction. If the cellview boundary is moved, the global grid will remain unchanged.

Note: The initial value for the cellview is determined by the `snapPatternGlobalGridAnchoring` environment variable. By default, `snapPatternGlobalGridAnchoring` is set to `t`, which corresponds to `boundary`.

Arguments

d_dbCellViewId	ID of the cellview used to store the global grid offset reference type.
t_referenceType	The value of the global grid offset reference type. Valid values: `boundary` or `origin`.

Value Returned

`t`	The global grid offset reference type is set on the specified cellview.
`nil`	Returned in case of failure.

Example

© 2020 All Rights Reserved.

```
cv = dbOpenCellViewByType("design" "top" "layout")
```

Opens a cellview by using the `dbOpenCellViewByType` SKILL command that returns the cellview ID `cv`.

```
dbSetGlobalGridOffsetReferenceType(cv "origin")
```

Passes `cv` and `origin` to lock the global grid offset reference at the origin.

© 2020 All Rights Reserved.

dbGetGlobalGridOffsetReferenceType

```
dbGetGlobalGridOffsetReferenceType(
     d_dbCellViewId
     )
     => t_referenceType
```

Description

(ICADVM20.1 Only) Returns the global grid offset reference. The value returned can either be `origin` or `boundary`.

Arguments

d_dbCellViewId	ID of the cellview used to obtain the global grid offset reference type.

Value Returned

t_referenceType	The value of the global grid offset reference type. Valid values: `boundary` or `origin`.

Example

```
cv = dbOpenCellViewByType("design" "top" "layout")
```

Opens a cellview by using the `dbOpenCellViewByType` SKILL command that returns the cellview ID `cv`.

```
dbGetGlobalGridOffsetReferenceType(cv)
```

Passes the `cv` to `getGlobalGridOffsetReferenceType` to obtain the global grid reference type.

© 2020 All Rights Reserved.

dbGetGlobalGridDefaultRepeatMode

```
dbGetGlobalGridDefaultRepeatMode(
     d_cellViewId
     t_wsspDefName
     )
     => t_repeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the default repeat mode for the specified widthSpacingSnapPatternDef global grid in the specified cellview.

Arguments

d_cellViewId	ID of a cellview.
t_wsspDefName	Name of a widthSpacingSnapPatternDef object.

Value Returned

t_repeatMode	Value of the default repeat mode.
nil	Returned in case of failure.

Example

```
dbGetGlobalGridDefaultRepeatMode(cellViewId "wsspDefName")
```

Returns the default repeat mode for the `wsspDefName` global grid in the cellview `cellViewId`.

© 2020 All Rights Reserved.

dbSetGlobalGridDefaultRepeatMode

```
dbSetGlobalGridDefaultRepeatMode(
     d_cellViewId
     t_wsspDefName
     t_repeatMode
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the specified repeat mode as the default on a cellview for the specified widthSpacingSnapPatternDef global grid.

Arguments

d_cellViewId	ID of a cellview
t_wsspDefName	Name of a widthSpacingSnapPatternDef object.
t_repeatMode	Name of the repeat mode to be set. Valid values: `noRepeat`, `stepped`, `flippedStartsWithOdd`, `flippedStartsWithEven`

Value Returned

`t`	The default repeat mode is successfully set for the specified widthSpacingSnapPatternDef global grid.
`nil`	Returned in case of failure.

Example

```
dbSetGlobalGridDefaultRepeatMode(cellViewId "wsspDefName" "stepped")
```

The value `stepped` is set as the default repeat mode for the `wsspDefName` global grid in the cellview `cellViewId`.

© 2020 All Rights Reserved.

dbGetGlobalGridRepeatMode

```
dbGetGlobalGridRepeatMode(
     d_cellViewId
     t_wsspDefName
     t_patternName
     )
     => t_repeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the repeat mode for the specified widthSpacingSnapPatternDef global grid in the specified cellview for the given width spacing pattern.

Arguments

d_cellViewId	ID of a cellview.
t_wsspDefName	Name of a widthSpacingSnapPatternDef object.
t_patternName	Name of a width spacing pattern.

Value Returned

t_repeatMode	Value of a valid repeat mode.
nil	Returned in case of failure.

Example

```
dbGetGlobalGridRepeatMode(cellViewId "wsspDefName" "wspName")
```

Returns the repeat mode for the `wsspDefName` global grid in the cellview `cellViewId` for the width spacing pattern `wspName`.

© 2020 All Rights Reserved.

dbSetGlobalGridRepeatMode

```
dbSetGlobalGridRepeatMode(
     d_cellViewId
     t_wsspDefName
     t_patternName
     t_repeatMode
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the specified repeat mode for the widthSpacingSnapPatternDef global grid in the specified cellview for the given width spacing pattern.

Arguments

d_cellViewId	ID of a cellview.
t_wsspDefName	Name of the widthSpacingSnapPatternDef.
t_patternName	Name of the width spacing pattern.
t_repeatMode	Name of the repeat mode to be set. Valid values: `noRepeat`, `stepped`, `flippedStartsWithOdd`, `flippedStartsWithEven`

Value Returned

`t`	Returned in case of success.
`nil`	Returned in case of failure.

Example

```
dbSetGlobalGridRepeatMode(cellViewId "wsspDefName" "wspName" "stepped")
```

The repeat mode `stepped` is set for the `wsspDefName` global grid in the cellview `cellViewId` for the width spacing pattern `wspName`.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

15

Hierarchical Design Functions

This chapter describes the SKILL functions to hierarchical designs.

- Hierarchy Database Functions
- Occurrence Property Functions
- Parent/Child Database Functions

© 2020 All Rights Reserved.

Hierarchy Database Functions

This section describes the following SKILL functions:

- hdbAddConfigToBag
- hdbBind
- hdbCloseConfigsInBag
- hdbCreateConfigBag
- hdbCreatePathVectorStack
- hdbExplainBinding
- hdbExplainSignature
- hdbExplainStop
- hdbGetPathStackStr
- hdbGetPathStr
- hdbGetPathVectorStackTop
- hdbIsAtStopPoint
- hdbIsConfig
- hdbIsOpenConfig
- hdbReOpenNewConfig
- hdbPopPathVect
- hdbPushPathVect
- hdbSaveACopy
- hdbSaveAs
- hdbSave
- hdbClose
- hdbOpen
- hdbDestroyPathVector
- hdbGetCellName

© 2020 All Rights Reserved.

- hdbGetLibName
- hdbGetTopCellName
- hdbGetTopLibName
- hdbGetTopViewName
- hdbGetViewName
- hdbPopCell
- hdbPushCell
- hdbSetObjBindRule
- hdbSetDefaultLibListString
- hdbSetDefaultViewListString
- hdbSetDefaultStopListString
- hdbSetTopCellViewName
- hdbBindBit
- hdbBitHasRules
- hdbHasIterInstBitRules
- hdbIsIterInst
- hdbIterInstHasBitRules
- hdbPushBitCell

hdbAddConfigToBag

```
hdbAddConfigToBag(
     h_configBagId
     h_configId
     )
     => t / nil
```

Description

Adds a configuration handle to a configuration bag.

© 2020 All Rights Reserved.

Arguments

h_configBagId	The ID of the configuration bag.
h_configId	A configuration ID, something returned by `hdbOpen`.

Value Returned

`t`	The configuration bag exists and a handle has been added to it.
`nil`	The *h_configBagId* was invalid.

Example

```
hdbAddConfigToBag( configBag config )
```

Reference

hdbCreateConfigBag
hdbCloseConfigsInBag

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbBind

```
hdbBind(
     h_pathVector
     t_masterLibName
     t_masterCellName
     t_masterViewName
     t_instName
     g_fixedLib
     g_fixedCell
     g_fixedView
     [ g_wantSignature ]
     [ g_wantWhy ]
     )
     => ( t_boundLibId t_boundCellId t_boundView h_bindSignature h_bindWhy )
```

Description

Binds an instance to a master.

Arguments

h_pathVector	The path vector ID returned by `hdbCreatePathVector`.
t_masterLibName	The master library name for the instances.
t_masterCellName	The master cell name for the instances.
t_masterViewName	The master view name for the instances.
t_instName	The *instName* for the instance.
g_fixedLib	The indicator of whether *t_masterLibName* can be changed (`t`) or not (`nil`) by `hdbBind`.
g_fixedCell	The indicator of whether *t_masterCellName* can be changed (`t`) or not (`nil`) by `hdbBind`.
g_fixedView	The indicator of whether *t_masterViewName* can be changed (`t`) or not (`nil`) by `hdbBind`.
g_wantSignature	The indicator of whether a *h_bindSignature* object is returned (`t`) or not (`nil`).
g_wantWhy	The indicator of whether a *BindWhy* object is returned (`t`) or not (`nil`).

© 2020 All Rights Reserved.

Value Returned

t_boundLib	The library name of the bound instance.
t_boundCell	The cell name of the bound instance.
t_boundView	The view name of the bound instance.
h_bindSignatureId	The ID of an object that can be used to explain the uniqueness of a binding.
h_bindWhyId	The ID of an object that can be used to explain why a binding was made the way it was.

Example

```
bindRtn = hdbBind(
      pathVect
      "lib1" "cell1"
      "view1" "inst1"
      nil nil nil
      t t)
```

Reference

hdbExplainBinding
hdbExplainSignature

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbCloseConfigsInBag

```
hdbCloseConfigsInBag(
     h_configBagId
     )
     => t / nil
```

Description

Closes all of the configurations held in a configuration bag.

Arguments

h_ConfigBagId	A configuration bag ID; something returned by `hdbCreateConfigBag`.

Value Returned

`t`	All of the configurations held in the bag were closed successfully.
`nil`	An error was encountered while trying to close all the configs in the bag.

Example

```
hdbCloseConfigsInBag( configBag )
```

Reference

hdbCloseConfigsInBag
hdbAddConfigToBag
hdbCreateConfigBag

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbCreateConfigBag

```
hdbCreateConfigBag(
     )
     => h_configBagId / nil
```

Description

Creates a configuration bag to hold configurations.

Arguments

None

Value Returned

h_configBagId	The ID of a bag to put configurations in.
nil	A configuration bag was not created.

Example

```
configBag = hdbCreateConfigBag()
```

Reference

hdbCloseConfigsInBag
hdbAddConfigToBag

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbCreatePathVectorStack

```
hdbCreatePathVectorStack(
     )
     => h_pathVectorStackId / nil
```

Description

Creates a stack to hold path vectors when traversing into sub-configurations.

Arguments

None

Value Returned

h_pathVectorStackId	The ID of the created path vector stack.
nil	A path vector stack could not be created.

Example

```
pathVectStack = hdbCreatePathVectorStack()
```

Reference

hdbGetPathStackStr
hdbGetPathVectorStackTop
Attribute Types
hdbPushPathVect

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbExplainBinding

```
hdbExplainBinding(
     t_indentation
     t_masterLibName
     t_masterCellName
     t_masterViewName
     t_instName
     g_fixedLib
     g_fixedCell
     g_fixedView
     t_boundLib
     t_boundCell
     t_boundView
     h_bindSignatureId
     h_bindWhyId
     )
     => t_string
```

Description

Explains why a `hdbBind` binding was made the way it was.

Arguments

t_indentation	The indentation string that will precede each line.
t_masterLibName	The master library name for the instance.
t_masterCellName	The master cell name for the instance.
t_masterViewName	The master view name for the instance.
t_instName	The *instName* for the instance.
g_fixedLib	The library is bound.
g_fixedCell	The cell is bound.
g_fixedView	The view is bound.
t_boundLib	The library name bound by `hdbBind`.
t_boundCell	The cell name bound by `hdbBind`.
t_boundView	The view name bound by `hdbBind`.
h_bindSignatureId	The *bindSignature* object returned by `hdbBind`.
h_bindWhyId	The *bindWhy* object returned by `hdbBind`

© 2020 All Rights Reserved.

Value Returned

t_string	A string that explains the binding.

Example

```
whyBoundStr = hdbExplainBinding(
      "    " "lib1" "cell1"
      "view1" "inst1"
      nil nil nil
      "lib2" "cell2" "view2"
      bindSig bindWhy)
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbExplainSignature

```
hdbExplainSignature(
     t_indentation
     h_bindSignatureId
     )
     => t_string
```

Description

Explains the contents of a binding signature.

Arguments

t_indentation	The indentation string that precedes each line.
h_bindSignatureId	A *bindSignature* object; something returned by `hdbBind`.

Value Returned

t_string	A string that explains the contents of the binding signature.

Example

```
bindSigStr = hdbExplainSignature("    " bindSignature)
```

Reference

hdbBind

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbExplainStop

```
hdbExplainStop(
     t_indentation
     h_pathVectorId
     )
     => t_string
```

Description

Explains why a stop was declared.

Arguments

t_indentation	The indentation string that precedes each line.
h_pathVectorId	A path vector ID; something returned by `hdbCreatePathVector`.

Value Returned

t_string	A string that explains why a stop was declared.

Example

```
whyStopStr = hdbExplainStop("    " pathVector)
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetPathStackStr

```
hdbGetPathStackStr(
     h_pathVectorStackId
     )
     => t_string
```

Description

Creates a string listing all the path vectors. Creates something similar to a fully rooted path for a design binding that contains sub-configurations.

Arguments

h_pathVectorStackId

A path vector ID; something returned by `hdbCreatePathVectorStack`.

Value Returned

t_string All the path vectors in a path vector stack.

Example

```
whereAreWe = hdbGetPathStackStr( pathVectorStack )
```

Reference

hdbGetPathVectorStackTop
hdbCreatePathVectorStack
Attribute Types
hdbPushPathVect

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetPathStr

```
hdbGetPathStr(
     h_pathVectorId
     )
     => t_string
```

Description

Creates a string representation of a path vector.

Arguments

h_pathVectorId	A path vector ID; something returned by `hdbCreatePathVector`.

Value Returned

t_string	The path vector.

Example

```
path = hdbGetPathStr( pathVector )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetPathVectorStackTop

```
hdbGetPathVectorStackTop(
     h_pathVectorStackId
     )
     => h_pathVectorId
```

Description

Gets the top of a path vector stack.

Arguments

h_pathVectorStackId

A path vector stack ID; something returned by `hdbCreatePathVectorStack`.

Value Returned

h_pathVectorId The top *pathVectorId*.

Example

```
pathVect = hdbGetPathVectorStackTop( pathVectorStack )
```

Reference

hdbGetPathStackStr
hdbCreatePathVectorStack
Attribute Types
hdbPushPathVect

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbIsAtStopPoint

```
hdbIsAtStopPoint(
     h_pathVectorId
     )
     => t / nil
```

Description

Checks if the point just pushed into with `hdbPushCell` is a stop point.

Arguments

h_pathVectorId	A path vector ID; something returned by `hdbCreatePathVector`.

Value Returned

`t`	The point just pushed into with `hdbPushCell` is a stop point.
`nil`	The point just pushed into with `hdbPushCell` is not a stop point.

Example

```
stop = hdbIsAtStopPoint( pathVector )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbIsConfig

```
hdbIsConfig(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Checks if a given library/cell/view is a configuration.

Arguments

t_libName	The library name.
t_cellName	The cell name.
t_viewName	The view name.

Value Returned

t	A given library/cell/view is a configuration.
nil	A given library/cell/view is not a configuration.

Example

```
isConfig = hdbIsConfig("lib1" "cell1" "view1")
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbIsOpenConfig

```
hdbIsOpenConfig(
     h_configID
     )
     => t / nil
```

Description

Checks if the configuration ID you specify is valid. The configuration ID is valid if the configuration is still open.

Arguments

h_configID	A configuration ID, returned by hdbOpen.

Value Returned

`t`	The configuration ID is valid.
`nil`	The configuration ID is not valid.

Reference

hdbOpen

© 2020 All Rights Reserved.

hdbReOpenNewConfig

```
hdbReOpenNewConfig(
     h_configID
     )
     => t / nil
```

Description

Increments the reference count of an in-memory only configuration (that is, a new configuration that has not yet been written to disk) and returns `t`. If the configuration does not exist in memory or if it already exists on disk, this function returns `nil` and the reference count is not incremented.

Note: The reference count of a configuration keeps track of the number of times the configuration has been opened. `hdbOpen` and this function increment the reference count, while `hdbClose` decrements it.

Arguments

h_configId	A configuration ID, returned by `hdbOpen`.

Value Returned

`t`	The reference count is incremented.
`nil`	The reference count is not incremented because the configuration does not exist in memory or it already exists on disk.

Example

New in-memory configuration:

```
myConfig=hdbOpen("" "" "" "w")
hdbReOpenNewConfig(myConfig)
=> t
```

Configuration that exists on disk:

```
config=hdbOpen("mylib" "mycell" "myview" "config" "w")
hdbReOpenNewConfig(config)
=> nil
```


© 2020 All Rights Reserved.

hdbPopPathVect

```
hdbPopPathVect(
     h_pathVectorStackId
     )
     => h_pathVectorId / nil
```

Description

Pops a path vector off the top of a path vector stack.

Arguments

h_pathVectorStackId

A path vector stack ID; something returned by `hdbCreatePathVectorStack`.

Value Returned

h_pathVectorId	The top of the path vector stack.
`nil`	The stack is empty.

Example

```
pathVectorTop = hdbPopPathVect( pathVectorStack )
```

Reference

hdbGetPathStackStr
hdbCreatePathVectorStack
hdbGetPathVectorStackTop
hdbPushPathVect

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

hdbPushPathVect

```
hdbPushPathVect(
     h_pathVectorStackId
     h_pathVectorId
     )
     => t / nil
```

Description

Adds a path vector on to the top of a path vector stack.

Arguments

h_pathVectorStackId	A path vector stack ID; something returned by `hdbCreatePathVectorStack`.
h_pathVectorId	A path vector ID; something returned by `hdbCreatePathVector`.

Value Returned

`t`	The path vector is added on to the top of a path vector stack.
`nil`	No path vector was added on to the top of a path vector stack.

Example

```
hdbPushPathVect( pathVectorStack pathVector )
```

Reference

hdbGetPathStackStr
hdbCreatePathVectorStack
hdbGetPathVectorStackTop
Attribute Types

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbSaveACopy

```
hdbSaveACopy(
     h_configId
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Makes a copy of a configuration and saves it to a library/cell/view. Unlike hdbSaveAs, this function does not modify the location of the configuration associated with the specified configId.

Example:

Arguments

h_configId	A configuration ID; something returned by `hdbOpen`.
t_libName	A library name.
t_cellName	A cell name.
t_viewName	A view name.

Value Returned

`t`	The configuration is saved to library/cell/view.
`nil`	The configuration is not saved to library/cell/view.

Example

```
cv=hdbOpen("LIB1" "cell1" "config" "r")
hdbSaveACopy(cv "LIB1" "cell1" "config_new")
```

Makes a copy of the configuration `cv` and saves it to `LIB1/cell1/config_new`.

© 2020 All Rights Reserved.

hdbSaveAs

```
hdbSaveAs(
     h_configId
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Saves a configuration to a library/cell/view.

Arguments

h_configId	A configuration ID; something returned by `hdbOpen`.
t_libName	A library name.
t_cellName	A cell name.
t_viewName	A view name.

Value Returned

`t`	The configuration is saved to library/cell/view.
`nil`	The configuration is not saved to library/cell/view.

Example

```
ok = hdbSaveAs( config "lib1" "cell1" "view1" )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbSave

```
hdbSave(
     h_configId
     )
     => t / nil
```

Description

Saves a configuration opened with `hdbOpen`.

Arguments

h_configId	A configuration ID returned from a call to hdbOpen.

Value Returned

`t`	The configuration is saved.
`nil`	The save operation failed.

Example

```
hdbSave( hdbConfig )
```


© 2020 All Rights Reserved.

hdbClose

```
hdbClose(
     h_configId
     )
     => t / nil
```

Description

Closes a configuration. In all cases where you are using the hdb interface to traverse (bind, elaborate) a design use `hdbCloseConfigsInBag` to close open configurations instead of this function.

Arguments

h_configId	The ID of a configuration.

Value Returned

`t`	Closes the configuration.
`nil`	The close operation failed.

Example

```
closeStatus = hdbClose( hdbConfig )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbOpen

```
hdbOpen(
     t_libName
     t_cellName
     t_viewName
     t_mode
     [ t_nameSpace ]
     )
     => h_configId / nil
```

Description

Opens a configuration for read or edit.

© 2020 All Rights Reserved.

Arguments

t_libName	The library name for the configuration you want to open.
t_cellName	The cell name for the configuration you want to open.
t_viewName	The view name for the configuration you want to open.
t_mode	String value of `r`, `w`, or `a`. ■ `"r"` Opens a configuration for read by reading in an existing configuration file. ■ `"w"` Creates a new (empty) configuration for edit. When the configuration is saved, a new configuration file is created. If a configuration file already existed it is ignored. ■ `"a"` Opens a configuration for edit. If a configuration file exists, it is modified at the time of save. If the configuration file doesn't exist, a new one is created.
t_nameSpace	The name space you want to use for library/cell/view and instance names when using the hdb interface.

Value Returned

h_configId	The ID of the configuration opened.
`nil`	The open operation failed.

Example

```
hdbConfig = hdbOpen( "lib1" "cell1" "view1" "r" "CDB" )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbCreatePathVector

```
hdbCreatePathVector(
     h_configId
     )
     => h_pathVectorId / nil
```

Description

Creates a path vector to use while traversing (elaborating) a design hierarchy.

Arguments

h_configId	The ID of a configuration obtained with `hdbOpen`.

Value Returned

h_pathVectorId	The path vector for the configuration specified by a configuration ID.
`nil`	The operation failed.

Example

```
pathVector = hdbCreatePathVector( config )
```

Reference

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

hdbDestroyPathVector

```
hdbDestroyPathVector(
     h_pathVectorId
     )
     => t / nil
```

Description

Destroys a path vector.

Arguments

h_pathVectorId	The path vector obtained with `hdbCreatePathVector`.

Value Returned

`t`	The path vector has been destroyed.
`nil`	The path vector has not been destroyed.

Example

```
status = hdbDestroyPathVector( pathVector )
```

Reference

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

hdbGetCellName

```
hdbGetCellName(
     h_configId
     )
     => t_cellName / nil
```

Description

Gets the cell name of a configuration specified by *h_configId*.

Arguments

h_configId	The ID of a configuration obtained with `hdbOpen`.

Value Returned

t_cellName	The cell name of the configuration specified by *h_configId*.
`nil`	Did not get the cell name of the configuration specified by *h_configId*.

Example

```
configCellName = hdbGetCellName( config )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetLibName

```
hdbGetLibName(
     h_configId
     )
     => libName / nil
```

Description

Gets the library name of a configuration specified by *h_configId*.

Arguments

h_configId	The ID of a configuration obtained with `hdbOpen`.

Value Returned

libName	The library name of the configuration specified by *h_configId*.
`nil`	Did not get the library name of the configuration specified by *h_configId*.

Example

```
configLibName = hdbGetCellName( config )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetTopCellName

```
hdbGetTopCellName(
     h_configId
     )
     => t_topCellName / nil
```

Description

Gets the name of the top cell of a configuration specified by *h_configId*.

Arguments

h_configId	The ID of a configuration obtained with `hdbOpen`.

Value Returned

t_topCellName	The top cell name of the configuration specified by *h_configId*.
`nil`	Did not get the top cell name of the configuration specified by *h_configId*.

Example

```
topCell = hdbGetTopCellName( config )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetTopLibName

```
hdbGetTopLibName(
     h_configId
     )
     => t_topLibName / nil
```

Description

Gets the name of the top library of a configuration specified by *h_configId*.

Arguments

h_configId	The ID of a configuration obtained with `hdbOpen`.

Value Returned

t_topLibName	The top library name of the configuration specified by *h_configId*.
`nil`	Did not get the top library name of the configuration specified by *h_configId*.

Example

```
topLib = hdbGetTopLibName( config )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetTopViewName

```
hdbGetTopViewName(
     h_configId
     )
     => t_topViewName / nil
```

Description

Gets the name of the top view pointed to by *h_hdbConfig*.

Arguments

h_configId	The ID of a configuration obtained with `hdbOpen`.

Value Returned

t_topViewName	The top view name of the configuration specified by *h_configId*.
`nil`	Did not get the top view name of the configuration specified by *h_configId*.

Example

```
topView = hdbGetTopViewName( config )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbGetViewName

```
hdbGetViewName(
     h_configId
     )
     => t_viewName / nil
```

Description

Gets the view name of a configuration specified by *h_configId*.

Arguments

h_configId	The ID of a configuration obtained with `hdbOpen`.

Value Returned

t_viewName	The view name of the configuration specified by *h_configId*.
`nil`	Did not get the view name of the configuration specified by *h_configId*.

Example

```
configViewName = hdbGetViewName( config )
```

Reference

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

hdbPopCell

```
hdbPopCell(
     h_pathVectorId
     )
     => t / nil
```

Description

Pops an entry off a path vector.

Arguments

h_pathVectorId	A path vector ID; something obtained from `hdbCreatePathVector`.

Value Returned

`t`	A cell has been popped from a path vector.
`nil`	A cell was not popped from a path vector.

Example

```
status = hdbPopCell( pathVector )
```

Reference

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

hdbPushCell

```
hdbPushCell(
     h_pathVectorId
     t_instanceName
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Push a cell onto a path vector.

Arguments

h_pathVectorId	A path vector ID; something obtained from `hdbCreatePathVector`.
t_instanceName	An instance name.
t_libName	A library name.
t_cellName	A cell name.
t_viewName	A view name.

Value Returned

`t`	A cell has been pushed on to a path vector.
`nil`	A cell has not been pushed on to a path vector.

Example

```
status = hdbPushCell( pathVector "inst1" "lib1" "cell1" "view1" )
```

Reference

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

hdbSetObjBindRule

```
hdbSetObjBindRule(
     h_configId
     g_specifier
     g_rule
     )
     => t / nil
```

Description

Sets or deletes a binding rule on a cell, instance, or occurrence (path). You can set the following binding rules: library list, view list, binding, or stop point.

© 2020 All Rights Reserved.

Arguments

h_configId

The ID of the configuration that is opened in write or append mode; the return value of `hdbOpen`.

g_specifier

A list of lists specifying the cell, instance, or occurrence on which to set the binding rule.

To set a binding rule on a cell, specify a list that contains a single list of four elements:

```
((lib cell nil nil))
```

Here,

- Value of *lib* is a string or `nil` to indicate that the library list should be used to locate the specified cell.
- *cell* is a required string.

Example:

```
(("lib1" "cell1" nil nil))
```

Sets a binding rule on the cell `lib1.cell1`.

To set a binding rule on an instance, specify a list that contains a single list of four elements:

```
((lib cell view instance))
```

Here,

- Value of *lib* is a string or `nil` to indicate that the library list should be used to locate the specified cell.
- *cell* is a required string
- Value of *view* is a string or `nil` to indicate that the view should be determined using the view list.
- *instance* is a required string.

Example:

```
(("lib1" "cell1" "view1" "inst1"))
```

Sets a binding rule on instance `(lib1.cell1.view1).inst1`.

© 2020 All Rights Reserved.

To set a binding rule on an occurrence, specify a list of lists, each of which must have a library, cell, view, and instance. The list specifies the path to the occurrence, beginning with the top design.

```
((lib cell view inst) (lib cell view inst)
...)
```

g_rule One of the following rules:

Rule	**Specified as ...**
Library list	`('hdbcLibListRule ("lib1" "lib2" "lib3" ...))`
View list	`('hdbcViewListRule ("view1" "view2" "view3" ...))`
Binding	`('hdbcBindingRule ("lib "cell" "view"))`

where *lib*, *cell*, or *view* can also be `nil`. For example:

```
('hdbcBindingRule ("lib1 "cell1" "view1"))
('hdbcBindingRule (nil "cell1" "view1"))
('hdbcBindingRule (nil nil "view1"))
```

Note: When setting a binding rule on an occurrence (path), all fields must be specified in the rule. For example,

```
rule = list ('hdbcBindingRule list ("worklib"
"subsub" "schematic2"))
```

Stop list	`('hdbcStopRule t)`

To delete a rule, specify `nil` as the second element of the list. For example: `('hdbcStopRule nil)`

© 2020 All Rights Reserved.

Value Returned

`t`	The binding is set or deleted.
`nil`	There was an error.

Examples

```
hdbSetObjBindRule(myConfigID list(list("lib1" "cell1" nil nil)) list('hdbcStopRule
t))
```

To set a binding rule on an occurrence, the specifier for the path list would look like:

```
spec= list(list ("worklib" "top" "schematic" "IO") list("worklib" "sub"
"schematic" "IO")
```

and the rule for it would look like:

```
rule= list('hdbcBindingRule list ("worklib" "subsub" "schematic2")
```


© 2020 All Rights Reserved.

hdbSetDefaultLibListString

```
hdbSetDefaultLibListString(
     h_configID
     t_liblist
     )
     => t / nil
```

Description

Sets the global library list in the configuration specified by the *h_configId*

Arguments

h_configId	Specifies a configuration ID returned by `hdbOpen`
t_libList	Specifies the library list that needs to be set

Value Returned

`t`	If the library list has been set
`nil`	If the library list has not been set

Example

```
status = hdbSetDefaultLibListString( config "lib1 lib2 lib3" )
```


© 2020 All Rights Reserved.

hdbSetDefaultViewListString

```
hdbSetDefaultViewListString(
     h_configID
     t_viewList
     )
     => t / nil
```

Description

Sets the global view list in the configuration specified by *h_configId*

Arguments

h_configId	Specifies a configuration ID returned by `hdbOpen`
t_viewList	Specifies the view list that needs to be set

Value Returned

`t`	If the view list has been set
`nil`	If the view list has not been set

Example

```
status = hdbSetDefaultViewListString( config "view1 view2 view3" )
```


© 2020 All Rights Reserved.

hdbSetDefaultStopListString

```
hdbSetDefaultStopListString(
     h_configID
     t_stopList
     )
     => t / nil
```

Description

Sets the global stop view list in the configuration specified by *h_configId*

Arguments

h_configId	Specifies a configuration ID returned by `hdbOpen`
t_stopList	Specifies the stop view list that needs to be set

Value Returned

`t`	If the stop view list has been set
`nil`	If the stop view list has not been set

Example

```
status = hdbSetDefaultStopListString( config "stop1 stop2 stop3" )
```


© 2020 All Rights Reserved.

hdbSetTopCellViewName

```
hdbSetTopCellViewName(
     h_configID
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Sets the top of the design hierarchy to `library.cell:view` in the configuration specified by *`h_configId`*

Arguments

`h_configId`	Specifies a configuration ID returned by `hdbOpen`
`t_libName`	Specifies a library name
`t_cellName`	Specifies a cell name
`t_viewName`	Specifies a view name

Value Returned

`t`	If the top of the design hierarchy has been set
`nil`	If the top of the design hierarchy has not been set

Example

```
status = hdbSetTopCellViewName( config "lib1" "cell1" "view1" )
```


© 2020 All Rights Reserved.

hdbBindBit

```
hdbBindBit(
     h_pathVector
     t_masterLibName
     t_masterCellName
     t_masterViewName
     t_baseInstName
     x_bit
     x_leftRangeIndex
     x_rightRangeIndex
     g_fixedLib
     g_fixedCell
     g_fixedView
     [ g_wantSignature ]
     [ g_wantWhy ]
     )
     => ( t_boundLib t_boundCell t_boundView h_bindSignatureId h_bindWhyId )
```

Description

Binds a single bit of an iterated instance.

Arguments

h_pathVector	The path vector ID returned by `hdbCreatePathVector`.
t_masterLibName	The master library name of the instance.
t_masterCellName	The master cell name of the instance.
t_masterViewName	The master view name of the instance.
t_baseInstName	The base part of the iterated instance name. For example, if the iterated instance name is `I1<0:3>`, the base instance name is `I1`.
x_bit	The index number of the bit of the iterated instance being bound.
x_leftRangeIndex	The left range index of the iterated instance.
x_rightRangeIndex	The right range index of the iterated instance.
g_fixedLib	Indicates whether *t_masterLibName* can be changed (`t`) or not (`nil`).
g_fixedCell	Indicates whether *t_masterCellName* can be changed (`t`) or not (`nil`).

© 2020 All Rights Reserved.

g_fixedView	Indicates whether *t_masterViewName* can be changed (t) or not (nil).
g_wantSignature	Indicates whether an *h_bindSignature* object is returned (t) or not (nil).
g_wantWhy	Indicates whether an *h_bindWhy* object is returned (t) or not (nil).

Value Returned

t_boundLib	The library name of the bound instance.
t_boundCell	The cell name of the bound instance.
t_boundView	The view name of the bound instance.
h_bindSignatureId	The ID of an object that indicates the uniqueness of a binding.
h_bindWhyId	The ID of an object that indicates why a binding was made the way it was.

Example

Binds bit 1 of the iterated instance I1<0:3>.

```
bindRtn = hdbBindBit( pathVect "lib1" "cell1" "view1" "I1" 1 0 3 nil nil nil t t )
```


© 2020 All Rights Reserved.

hdbBitHasRules

```
hdbBitHasRules(
     h_hdbPathVector
     t_baseInstName
     x_bit
     )
     => t / nil
```

Description

Determines whether a bit-specific configuration rule applies to the specified iterated instance bit.

Arguments

h_hdbPathVector	The path vector ID returned by hdbCreatePathVector.
t_baseInstName	The base part of the iterated instance name. For example, if the iterated instance name is `I1<0:3>`, the base instance name is `I1`.
x_bit	The index number of the bit of the iterated instance.

Value Returned

`t`	A bit-specific configuration rule applies to the specified iterated instance bit.
`nil`	A bit-specific configuration rule does not apply to the specified iterated instance bit.

Example

Checks whether bit `1` of an iterated instance with base name `I1` at the specified path has bit rules.

```
hasRules = hdbBitHasRules( pathVect "I1" 1 )
```


© 2020 All Rights Reserved.

hdbHasIterInstBitRules

```
hdbHasIterInstBitRules(
     h_configId
     )
     => t /nil
```

Description

Determines whether a configuration contains rules that apply to iterated instance bits.

Argument

h_configId	A configuration ID, returned by hdbOpen.

Value Returned

`t`	The configuration contains rules that apply to iterated instance bits.
`nil`	The configuration does not contain rules that apply to iterated instance bits.

Example

An application that does not support iterated instance bit rules can detect configurations that contain these rules and take appropriate action.

```
when( hdbHasIterInstBitRules(configId)
    warn("The specified config has iterated instance bit rules that are \n
         not supported by this application.\n")
)
```


© 2020 All Rights Reserved.

hdbIsIterInst

```
hdbIsIterInst(
     t_instName
     [ t_nameSpace ]
     )
     => (t_baseInstName t_leftIndex t_rightIndex) / nil
```

Description

Determines whether an instance name is of an iterated instance. If it is of an iterated instance, the base instance name and range indexes are returned.

Arguments

t_instName	The *instName* for the instance.
t_nameSpace	The name space you want to use for library/cell/view and instance names when using the Hierarchy Editor interface.

Value Returned

t_baseInstName	The base part of the iterated instance name. For example, if the iterated instance name is `I1<0:3>`, the base instance name is `I1`.
x_leftRangeIndex	The left range index of the iterated instance.
x_rightRangeIndex	The right range index of the iterated instance.
`nil`	The instance name is not an iterated instance name.

Example

Determines whether an instance name is of a scalar instance or of an iterated instance and retrieves information about the iterated instance from its instance name.

```
iterInstInfo = hdbIsIterInst(instName)
if( null(iterInstInfo)
then
    printf("%s is a scalar instance\n" instName)
else
    baseInstName = car(iterInstInfo)
    left = cadr(iterInstInfo)
    right = caddr(iterInstInfo)
```


© 2020 All Rights Reserved.

```
    printf("%s is an iterated instance with range %x : %x\n"
            baseInstName left right)
)
```


© 2020 All Rights Reserved.

hdbIterInstHasBitRules

```
hdbIterInstHasBitRules(
     h_pathVectorId
     t_baseInstName
     x_leftRangeIndex
     x_rightRangeIndex
     )
     => t / nil
```

Description

Determines whether an iterated instance has bit-specific configuration rules. If it does, you might want to check whether the binding for individual bits of the iterated instance are different from the iterated instance binding.

Arguments

h_pathVectorId	The path vector ID as returned by hdbCreatePathVector.
t_baseInstName	The base part of the iterated instance name. For example, if the iterated instance name is `I1<0:3>`, the base instance name is `I1`.
x_leftRangeIndex	The left range index of the iterated instance.
x_rightRangeIndex	The right range index of the iterated instance.

Value Returned

`t`	The iterated instance has bit-specific configuration rules.
`nil`	There are no bit-specific configuration rules for the iterated instance.

Example

Determine whether the iterated instance I1<0:3> has bit specific configuration rules.

```
hasBitRules = hdbIterInstHasBitRules( pathVect "I1" 0 3 )
```


© 2020 All Rights Reserved.

hdbPushBitCell

```
hdbPushBitCell(
     h_pathVectorId
     t_baseInstName
     x_bit
     x_leftRangeIndex
     x_rightRangeIndex
     t_libName
     t_cellName
     t_viewName
     [ h_bindWhy ]
     )
     => t / nil
```

Description

Pushes the cell of an iterated instance bit on to a path vector.

Arguments

h_pathVectorId	A path vector ID as returned by hdbCreatePathVector.
t_baseInstName	The base part of the iterated instance name. For example, if the iterated instance name is `I1<0:3>`, the base instance name is `I1`.
x_bit	The number of the bit of the iterated instance being bound.
x_leftRangeIndex	The left range index of the iterated instance.
x_rightRangeIndex	The right range index of the iterated instance.
t_libName	The library name of the iterated instance bit.
t_cellName	The cell name of the iterated instance bit.
t_viewName	The view name of the iterated instance bit.
h_bindWhy	The ID of an object that explains why the binding was made the way it was.

Value Returned

`t`	A cell was pushed on to the specified path vector.
`nil`	A cell was not pushed on to the specified path vector.

© 2020 All Rights Reserved.

Example

Pushes a cell for bit `1` of the iterated instance `I1<0:3>` on to a path vector.

```
rtn = hdbPushBitCell(pathVect "I1" 1 0 3 "lib1" "cell1" "view1" )
```


© 2020 All Rights Reserved.

Occurrence Property Functions

This section describes the following SKILL functions that are used to work with occurrence properties of designs.

- occpClose
- occpCreateProp
- occpGetBestProp
- occpGetBitInstProp
- occpGetCellProp
- occpGetDefaultProp
- occpGetInstProp
- occpGetPathProp
- occpOpen
- occpRmBitInstProp
- occpRmCellProp
- occpRmDefaultProp
- occpRmPathProp
- occpRmInstProp
- occpSave
- occpSetBitInstProp
- occpSetCellProp
- occpSetDefaultProp
- occpSetInstProp
- occpSetPathProp

© 2020 All Rights Reserved.

occpClose

```
occpClose(
     o_occpHandle
     )
     => t / nil
```

Description

Closes a properties file without saving unsaved changes.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.

Value Returned

`t`	The properties file was closed.
`nil`	The properties file was not closed or does not exist.

Example

```
when( MyOccpHandle = occpOpen("lib1" "cell1" "cfgView1" "r")
    when( MyVariantProp = occpGetCellProp(MyOccpHandle "lib1" "cell1" "variant")
        printf("Variant is %L\n"  MyVariantProp->value))
    occpClose(MyOccpHandle))
```


© 2020 All Rights Reserved.

occpCreateProp

```
occpCreateProp(
     t_name
     g_value @key (nonInheritable nil) (noEval nil)
     )
     => r_occpProp / nil
```

Description

Creates an occurrence property structure. It is returned from the `occpGet<Type>Prop` family of functions and provided as an argument to the `occpSet<Type>Prop` family of functions.

Arguments

t_name	The name of the property.
g_value	The value of the property.
`nonInheritable`	Specifies that the property value is non-inheritable. The default value of `nil` indicates that the property value is inheritable.
`noEval`	Specifies that the property value is not subject to evaluation. The default value of `nil` indicates that the property value is subject to evaluation.

Value Returned

r_occpProp	The property name structure, as returned by `occpCreateProp`.
`nil`	The occurrence property structure could not be created.

Example

```
> MyProp = occpCreateProp("variant" "lowPower" ?noEval t)
> printf("name=%s value=%L nonInheritable=%L noEval=%L\n"
    MyProp->name  MyProp->value  MyProp->nonInheritable  MyProp->noEval)
name=variant value="lowPower" nonInheritable=nil noEval=t
t
```


© 2020 All Rights Reserved.

occpGetBestProp

```
occpGetBestProp(
     o_occpHandle
     h_pathVectorId
     t_propName
     )
     => r_occpProp / nil
```

Description

Gets the property information with the highest precedence for the specified occurrence path. The information is based on the occurrence path, the iterated instance, the instance, the cell, and the default properties with the specified name.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.
h_pathVectorId	The occurrence path for the property as returned from a call to `hdbCreatePathVector`.
t_propName	The name of the property.

Value Returned

r_occpProp	The property name structure, as returned by `occpCreateProp`.
`nil`	The property information does not exist or could not be retrieved.

Example

```
MyHdbHandle = hdbOpen("lib1" "cell1" "cfgView1" "r" "CDBA")
when( MyHdbHandle
    MyOccpHandle = occpOpen("lib1" "cell1" "cfgView1" "r" "CDBA")
    when( MyOccpHandle
        MyPathVec = hdbCreatePathVector(MyHdbHandle)
        hdbPushCell(MyPathVec "I42"
            hdbGetTopLibName(MyHdbHandle)
            hdbGetTopCellName(MyHdbHandle)
            hdbGetTopViewName(MyHdbHandle) )
        MyPathProp = occpGetBestProp(MyOccpHandle MyPathVec "variant")
        when( MyPathProp
            printf("I42 variant is %L\n" MyPathProp->value) )
        occpClose(MyOccpHandle) ))
```


© 2020 All Rights Reserved.

occpGetBitInstProp

```
occpGetBitInstProp(
     o_occpHandle
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_baseInstName
     x_bit
     t_propName
     )
     => r_occpProp / nil
```

Description

Gets the property information for an iterated instance.

© 2020 All Rights Reserved.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property. The default value of `nil` indicates no specific cell name.
t_viewName	The view name for the property. The default value of `nil` indicates no specific view name.
t_baseInstName	The base instance name for the iterated instance property.
x_bit	The integer bit value of the base instance name for the iterated instance property.
t_propName	The name of the property.

Value Returned

r_occpProp	The property name structure, as returned by `occpCreateProp`.
`nil`	The property information does not exist or could not be retrieved.

Example

```
when( or(
    occpGetBitInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" 2 "variant")
    occpGetBitInstProp(MyOccpHandle nil "cell1" "view1" "I42" 2 "variant")
    occpGetBitInstProp(MyOccpHandle nil nil "view1" "I42" 2 "variant")
    occpGetBitInstProp(MyOccpHandle nil nil nil "I42" 2 "variant") )
    printf("Iterated instance[2] has variant property\n") )
```


© 2020 All Rights Reserved.

occpGetCellProp

```
occpGetCellProp(
     o_occpHandle
     t_libName | nil
     t_cellName
     t_propName
     )
     => r_occpProp / nil
```

Description

Gets the property information for a cell.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property.
t_propName	The name of the property.

Value Returned

r_occpProp	The property name structure, as returned by `occpCreateProp`.
`nil`	The property information does not exist or could not be retrieved.

Example

```
when( or(
    occpGetCellProp(MyOccpHandle "lib1" "cell1" "variant")
    occpGetCellProp(MyOccpHandle nil "cell1" "variant")
    printf("Cell has variant property\n") )
```


© 2020 All Rights Reserved.

occpGetDefaultProp

```
occpGetDefaultProp(
     o_occpHandle
     t_propName
     )
     => r_occpProp / nil
```

Description

Gets default property information from the specified properties file.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.
t_propName	The name of the property.

Value Returned

r_occpProp	The property name structure, as returned by `occpCreateProp`.
`nil`	The property information does not exist or could not be retrieved.

Example

```
MyProp = occpGetDefaultProp(MyOccpHandle "variant")
when( MyProp
    printf("value is %L\n" MyProp->value))
```


© 2020 All Rights Reserved.

occpGetInstProp

```
occpGetInstProp(
     o_occpHandle
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_instName
     t_propName
     )
     => r_occpProp / nil
```

Description

Gets the property information for an instance.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property. The default value of `nil` indicates no specific cell name.
t_viewName	The view name for the property. The default value of `nil` indicates no specific view name.
t_instName	The instance name for the property.
t_propName	The name of the property.

Value Returned

r_occpProp	The property name structure, as returned by `occpCreateProp`.
`nil`	The property information does not exist or could not be retrieved.

© 2020 All Rights Reserved.

Example

```
when( or( occpGetInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" "variant")
    occpGetInstProp(MyOccpHandle nil "cell1" "view1" "I42" "variant")
    occpGetInstProp(MyOccpHandle nil nil "view1" "I42" "variant")
    occpGetInstProp(MyOccpHandle nil nil nil "I42" "variant") )
    printf("Instance has variant property\n") )
```


© 2020 All Rights Reserved.

occpGetPathProp

```
occpGetPathProp(
     o_occpHandle
     h_pathVectorId
     t_propName
     )
     => r_occpProp / nil
```

Description

Gets property information for the specified occurrence path.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.
h_pathVectorId	The occurrence path for the property as returned from a call to `hdbCreatePathVector`.
t_propName	The name of the property.

Value Returned

r_occpProp	The property name structure, as returned by `occpCreateProp`.
`nil`	The property information does not exist or could not be retrieved.

Example

```
MyHdbHandle = hdbOpen("lib1" "cell1" "cfgView1" "r" "CDBA")
when( MyHdbHandle
    MyOccpHandle = occpOpen("lib1" "cell1" "cfgView1" "r" "CDBA")
    when( MyOccpHandle
        MyPathVec = hdbCreatePathVector(MyHdbHandle)
        hdbPushCell(MyPathVec "I42"
            hdbGetTopLibName(MyHdbHandle)
            hdbGetTopCellName(MyHdbHandle)
            hdbGetTopViewName(MyHdbHandle) )
            hdbPushCell(MyPathVec "J84" "lib2" "cell2" "view2")
        MyPathProp = occpGetPathProp(MyOccpHandle MyPathVec "variant")
        when( MyPathProp
        printf("%s variant is %L\n" hdbGetPathStr(MyPathVec)
        MyPathProp->value) )
        hdbDestroyPathVector(MyPathVec)
        occpClose(MyOccpHandle) ))
```


© 2020 All Rights Reserved.

occpOpen

```
occpOpen(
     t_libName
     t_cellName
     t_viewName
     t_mode [t_nameSpace [x_iterInstSupport]]
     )
     => o_occpHandle / nil
```

Description

Opens a properties file in the specified configuration view for read or edit. When editing a properties file, include the `hdbOpen` call before `occpOpen` and close the properties file before closing the HDB handle.

© 2020 All Rights Reserved.

Arguments

t_libName	The library name for the configuration.
t_cellName	The cell name for the configuration.
t_viewName	The view name for the configuration.
t_mode	Specifies one of these modes: ■ `r`: Opens an existing properties file for reading. ■ `w`: Creates a new properties file and opens it for editing. If a properties file already exists, it is opened with previous content discarded. ■ `a`: Opens an existing properties file for editing. If it does not exist, creates a new file and opens it for editing.
t_nameSpace	The name space to use for library, cell, view, and instance names when using the occurrence properties SKILL interface. If it is not specified, the CDBA name space is used.
x_iterInstSupport	Indicates if the application supports iterated instance properties. The default value of `0` indicates that the application does not support iterated instances.

Value Returned

o_occpHandle	The handle for the opened properties file.
`nil`	The properties file was not opened or does not exist.

Example

```
unless( MyOccpHandle = occpOpen("lib1" "cell1" "cfgView1" "r" "CDBA")
    printf("Could not open properties file (readOnly)\n"))
```


© 2020 All Rights Reserved.

occpRmBitInstProp

```
occpRmBitInstProp(
     o_occpHandle
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_baseInstName
     x_bit
     t_propName
     )
     => t / nil
```

Description

Removes a property for an iterated instance.

© 2020 All Rights Reserved.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property. The default value of `nil` indicates no specific cell name.
t_viewName	The view name for the property. The default value of `nil` indicates no specific view name.
t_baseInstName	The base instance name for the iterated instance property.
x_bit	The integer bit value of the base instance name for the iterated instance property.
t_propName	The name of the property.

Value Returned

`t`	The property was removed.
`nil`	The property does not exist or could not be removed.

Example

```
when( occpGetBitInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" 2 "variant")
    occpRmBitInstProp(MyOccpHandle nil "cell1" "view1" "I42" 2 "variant") )
```


© 2020 All Rights Reserved.

occpRmCellProp

```
occpRmCellProp(
     o_occpHandle
     t_libName | nil
     t_cellName
     t_propName
     )
     => t / nil
```

Description

Removes a cell property.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property.
t_propName	The name of the property.

Value Returned

`t`	The cell property was removed.
`nil`	The cell property does not exist or could not be removed.

Example

```
when( occpGetCellProp(MyOccpHandle "lib1" "cell1" "variant")
    occpRmCellProp(MyOccpHandle nil "cell1" "variant") )
```


© 2020 All Rights Reserved.

occpRmDefaultProp

```
occpRmDefaultProp(
     o_occpHandle
     t_propName
     )
     => t / nil
```

Description

Removes a default property.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
t_propName	The name of the property.

Value Returned

`t`	The property was removed.
`nil`	The property does not exist or could not be removed.

Example

```
occpRmDefaultProp(MyOccpHandle "variant")
```


© 2020 All Rights Reserved.

occpRmPathProp

```
occpRmPathProp(
     o_occpHandle
     h_pathVectorId
     t_propName
     )
     => t / nil
```

Description

Removes an occurrence path property.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
h_pathVectorId	The occurrence path for the property as returned from a call to `hdbCreatePathVector`.
t_propName	The name of the property.

Value Returned

`t`	The property was removed.
`nil`	The property does not exist or could not be removed.

Example

```
occpRmDefaultProp(MyOccpHandle "variant")
```


© 2020 All Rights Reserved.

occpRmInstProp

```
occpRmInstProp(
     o_occpHandle
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_instName
     t_propName
     )
     => t / nil
```

Description

Removes an instance property.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property. The default value of `nil` indicates no specific cell name.
t_viewName	The view name for the property. The default value of `nil` indicates no specific view name.
t_instName	The instance name of the property.
t_propName	The name of the property.

Value Returned

`t`	The property was removed.
`nil`	The property does not exist or could not be removed.

© 2020 All Rights Reserved.

Example

```
MyHasLibProp =
    occpGetInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" "variant")
when( MyHasLibProp
    MyNoLibProp = occpGetInstProp(MyOccpHandle nil "cell1" "view1"
        "I42" "variant")
    when( MyNoLibProp && MyHasLibProp->value == MyNoLibProp->value
        occpRmInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" "variant") ))
```


© 2020 All Rights Reserved.

occpSave

```
occpSave(
     o_occpHandle
     )
     => t / nil
```

Description

Saves a properties file opened with `occpOpen` and keeps it open. Only saved updates are reflected when other tools access the file.

Arguments

o_occpHandle	The properties file handle returned from a call to `occpOpen`.

Value Returned

`t`	The properties file was saved.
`nil`	The properties file was not saved. The reasons could be that the file was not found, it was not open, or it did not have any changes to save.

Example

```
MyHdbHandle = hdbOpen("lib1" "cell1" "cfgView1" "a" "CDBA")
when( MyHdbHandle
    MyOccpHandle = occpOpen("lib1" "cell1" "cfgView1" "a" "CDBA")
    when( MyOccpHandle
        occpSetCellProp(MyOccpHandle "lib1" "cell1" occpCreateProp("variant"
            "lowPower"))
        occpSave(MyOccpHandle)
        occpClose(MyOccpHandle) )
    (hdbClose MyHdbHandle) )
```


© 2020 All Rights Reserved.

occpSetBitInstProp

```
occpSetBitInstProp(
     o_occpHandle
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_baseInstName
     x_bit
     r_occpProp
     )
     => t / nil
```

Description

Sets the property information for an iterated instance. The new setting overwrites any existing property setting for the iterated instance.

© 2020 All Rights Reserved.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property. It must be specified if a library name is specified. The default value of `nil` indicates no specific cell name.
t_viewName	The view name for the property. It must be specified if a cell name is specified. The default value of `nil` indicates no specific view name.
t_baseInstName	The base instance name for the iterated instance property.
x_bit	The integer bit value of the base instance name for the iterated instance property.
t_propName	The property name structure as returned by `occpCreateProp`.

Value Returned

`t`	The property was set.
`nil`	The property could not be set.

Example

```
MyProp = occpGetBitInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" 2 "variant")
when( MyProp
    occpSetBitInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" 3 MyProp)
    occpSetBitInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" 4 MyProp) )
```


© 2020 All Rights Reserved.

occpSetCellProp

```
occpSetCellProp(
     o_occpHandle
     t_libName | nil
     t_cellName
     r_occpProp
     )
     => t / nil
```

Description

Sets the property information for a cell. The new setting overwrites any existing property setting for the cell.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property.
r_occpProp	The property name structure, as returned by `occpCreateProp`.

Value Returned

`t`	The property was set.
`nil`	The property could not be set.

Example

```
unless( or( occpGetCellProp(MyOccpHandle "lib1" "cell1" "variant")
    occpGetCellProp(MyOccpHandle nil    "cell1" "variant") )
    occpSetCellProp(MyOccpHandle "lib1" "cell1" occpCreateProp("variant"
        "lowPower")) )
```


© 2020 All Rights Reserved.

occpSetDefaultProp

```
occpSetDefaultProp(
     o_occpHandle
     r_occpProp
     )
     => t / nil
```

Description

Sets the default property information. The new setting overwrites any existing default property setting.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
r_occpProp	The property name structure, as returned by `occpCreateProp`.

Value Returned

`t`	The property was set.
`nil`	The property could not be set.

Example

```
occpSetDefaultProp(MyOccpHandle occpCreateProp("variant" "lowPower"))
```


© 2020 All Rights Reserved.

occpSetInstProp

```
occpSetInstProp(
     o_occpHandle
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_instName
     r_occpProp
     )
     => t / nil
```

Description

Sets the property information for an instance. The new setting overwrites any existing property setting for the instance.

© 2020 All Rights Reserved.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
t_libName	The library name for the property. The default value of `nil` indicates no specific library name.
t_cellName	The cell name for the property. It must be specified if a library name is specified. The default value of `nil` indicates no specific cell name.
t_viewName	The view name for the property. It must be specified if a cell name is specified. The default value of `nil` indicates no specific view name.
t_instName	The instance name for the instance property.
r_occpProp	The property name structure, as returned by `occpCreateProp`.

Value Returned

`t`	The property was set.
`nil`	The property could not be set.

Example

```
unless( occpGetInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" "variant")
    occpSetInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42"
        occpCreateProp("variant" "lowPower")) )
```


© 2020 All Rights Reserved.

occpSetPathProp

```
occpSetPathProp(
     o_occpHandle
     h_pathVectorId
     r_occpProp
     )
     => t / nil
```

Description

Sets the occurrence path property information. The new setting overwrites any existing occurrence path property setting.

Arguments

o_occpHandle	The properties file handle returned by `occpOpen` when used with the `a` or `w` mode.
h_pathVectorId	The occurrence path for the property as returned from a call to `hdbCreatePathVector`.
r_occpProp	The property name structure, as returned by `occpCreateProp`.

Value Returned

`t`	The property was set.
`nil`	The property could not be set.

© 2020 All Rights Reserved.

Example

```
MyHdbHandle = hdbOpen("lib1" "cell1" "cfgView1" "a" "CDBA")
when( MyHdbHandle
    MyOccpHandle = occpOpen("lib1" "cell1" "cfgView1" "a" "CDBA")
    when( MyOccpHandle
        MyPathVec = hdbCreatePathVector(MyHdbHandle)
        hdbPushCell(MyPathVec "I42"
            hdbGetTopLibName(MyHdbHandle)
            hdbGetTopCellName(MyHdbHandle)
            hdbGetTopViewName(MyHdbHandle) )
        hdbPushCell(MyPathVec "J84" "lib2" "cell2" "view2")
        occpSetPathProp(MyOccpHandle MyPathVec occpCreateProp("variant"
            "lowCost"))
        hdbDestroyPathVector(MyPathVec)
        occpSave(MyOccpHandle)
        occpClose(MyOccpHandle) ))
```


© 2020 All Rights Reserved.

Parent/Child Database Functions

This section describes the following SKILL functions:

- pcdbClose
- pcdbGetInstGen
- pcdbGetInstMasterGen
- pcdbGetIgnoredInstMasterGen
- pcdbInstMasterCell
- pcdbInstMasterCellFixed
- pcdbInstMasterLib
- pcdbInstMasterLibFixed
- pcdbInstMasterView
- pcdbInstMasterViewFixed
- pcdbInstName
- pcdbInstWidth
- pcdbNextInst
- pcdbNextInstMaster
- pcdbOpen
- pcdbViewHasPcdbInfo
- pcdbSave

pcdbClose

```
pcdbClose(
     h_pcdbHandle
     )
     => t / nil
```

Description

Closes a pcdb handle opened with `pcdbOpen`.

© 2020 All Rights Reserved.

Arguments

h_pcdbHandle	A pcdb handle returned from a call to `pcdbOpen`.

Value Returned

`t`	The pcdb handle is closed.
`nil`	The close operation failed.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbOpen

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbGetInstGen

```
pcdbGetInstGen(
     h_master
     )
     => h_instGen / nil
```

Description

Creates an instance generator that can be used to get the instances for a given master.

Arguments

h_master	A master returned from a call to `pcdbNextInstMaster`.

Value Returned

h_instGen	An instance generator.
`nil`	Did not get an instance generator.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbNextInst

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbGetInstMasterGen

```
pcdbGetInstMasterGen(
     h_pcdbHandle
     )
     => h_masterGen / nil
```

Description

Creates a generator that can be used to get the masters of a view.

Arguments

h_pcdbHandle	A pcdb handle returned from a call to `pcdbOpen`.

Value Returned

b_masterGen	A generator.
`nil`	Did not get a generator.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbOpen

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbGetIgnoredInstMasterGen

```
pcdbGetIgnoredInstMasterGen(
     h_pcdbHandle
     )
     => h_masterGen / nil
```

Description

Creates a generator that can be used to get a list of ignored instance masters. An ignored instance master is one that is not a sub-design.

Arguments

h_pcdbHandle	A pcdb handle returned from a call to `pcdbOpen`.

Value Returned

b_masterGen	A generator.
`nil`	Did not create a generator.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbOpen

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

pcdbInstMasterCell

```
pcdbInstMasterCell(
     h_master
     )
     => t_cellName / nil
```

Description

Gets the cell name for a master.

Arguments

h_master	A master returned from a call to `pcdbNextInstMaster`.

Value Returned

t_cellName	A string which is the master's cell name.
`nil`	Did not get a cell name for a master.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbInstMasterLib
pcdbInstMasterView
pcdbInstMasterLibFixed
pcdbInstMasterCellFixed
pcdbInstMasterViewFixed

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

pcdbInstMasterCellFixed

```
pcdbInstMasterCellFixed(
     h_master
     )
     => t / nil
```

Description

Gets whether a master's cell name is fixed.

Arguments

h_master	A master returned from a call to `pcdbNextInstMaster`.

Value Returned

`t`	The master cell is fixed.
`nil`	The master cell is not fixed.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbInstMasterLib
pcdbInstMasterCell
pcdbInstMasterView
pcdbInstMasterLibFixed
pcdbInstMasterViewFixed

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbInstMasterLib

```
pcdbInstMasterLib(
     h_master
     )
     => t_libName / nil
```

Description

Gets the library name for a master.

Arguments

h_master	A master returned from a call to `pcdbNextInstMaster`.

Value Returned

t_libName	A string which is the master's library name.
`nil`	Did not get a library name for a master.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbInstMasterCell
pcdbInstMasterView
pcdbInstMasterLibFixed
pcdbInstMasterCellFixed
pcdbInstMasterViewFixed

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

pcdbInstMasterLibFixed

```
pcdbInstMasterLibFixed(
     h_master
     )
     => t / nil
```

Description

Gets whether a master's library name is fixed.

Arguments

h_master	A master returned from a call to `pcdbNextInstMaster`.

Value Returned

`t`	The master's library name is fixed.
`nil`	The master's library name is not fixed.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbInstMasterLib
pcdbInstMasterCell
pcdbInstMasterView
pcdbInstMasterCellFixed
pcdbInstMasterViewFixed

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbInstMasterView

```
pcdbInstMasterView(
     h_master
     )
     => t_viewName / nil
```

Description

Gets the view name for a master.

Arguments

h_master	A master returned from a call to `pcdbNextInstMaster`.

Value Returned

t_viewName	A string which is the master's view name.
`nil`	Did not get a view name for a master.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbInstMasterLib
pcdbInstMasterCell
pcdbInstMasterLibFixed
pcdbInstMasterCellFixed
pcdbInstMasterViewFixed

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

pcdbInstMasterViewFixed

```
pcdbInstMasterViewFixed(
     h_master
     )
     => t / nil
```

Description

Gets whether a master's view name is fixed.

Arguments

h_master	A master returned from a call to `pcdbNextInstMaster`.

Value Returned

`t`	The master's view name is fixed.
`nil`	The master's view name is not fixed.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInstMaster
pcdbInstMasterLib
pcdbInstMasterCell
pcdbInstMasterView
pcdbInstMasterLibFixed
pcdbInstMasterCellFixed

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbInstName

```
pcdbInstName(
     h_instance
     )
     => t_instName / nil
```

Description

Gets the instance name for an instance.

Arguments

h_instance	An instance returned from a call to `pcdbNextInst`.

Value Returned

t_instName	A string which is the instance name.
`nil`	Did not get an instance name for an instance.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInst
pcdbInstWidth

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbInstWidth

```
pcdbInstWidth(
     h_instance
     )
     => x_width / nil
```

Description

Gets the instance width for an instance.

Arguments

h_instance	An instance returned from a call to `pcdbNextInst`.

Value Returned

x_width	The instance width.
`nil`	Did not get an instance width for an instance.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbNextInst
pcdbInstName

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbNextInst

```
pcdbNextInst(
     h_instGen
     )
     => h_instance / nil
```

Description

Gets the next instance from an instance generator.

Arguments

h_instGen	An instance generator returned from a call to `pcdbNextInst`.

Value Returned

h_instance	An instance.
`nil`	Did not get an instance from an instance generator.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbGetInstGen
pcdbInstName
pcdbInstWidth

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

pcdbNextInstMaster

```
pcdbNextInstMaster(
     h_masterGen
     )
     => h_master / nil
```

Description

Gets the next master from a master generator.

Arguments

h_instGen	A master generator returned from a call to `pcdbNextInst`.

Value Returned

h_master	A master.
`nil`	Did not get a master from a master generator.

Example

An example for the use of the pcdb functions is at the end of this section.

Reference

pcdbGetInstMasterGen
pcdbInstMasterLib
pcdbInstMasterCell
pcdbInstMasterView
pcdbInstMasterLibFixed
pcdbInstMasterCellFixed
pcdbInstMasterViewFixed

An example for the use of this function can be found at

your_install_dir`/tools/hdb/examples/hdbTraverse.il`

© 2020 All Rights Reserved.

pcdbOpen

```
pcdbOpen(
     t_libName
     t_cellName
     t_viewName
     t_mode
     [ ?nameSpace t_nameSpace ]
     [ ?dontOpenNextLevel g_dontOpenNextLevel ]
     )
     => h_pcdbHandle / nil
```

Description

Opens a configuration and gets a handle that will allow access to the parent/child data for a view.

© 2020 All Rights Reserved.

Arguments

t_libName	The library name for the configuration you want to open.
t_libName	The cell name for the configuration you want to open.
t_viewName	The view name for the configuration you want to open.
t_mode	A string. Must be `r` for read.
?nameSpace *t_nameSpace*	The name space for library, cell, view, and instance names when using the pcdb SKILL interface. If it is not specified, the default name space, CDBA, is used.
?dontOpenNextLevel *g_dontOpenNextLevel*	By default, or if the value is `nil`, when opening an Open Access view, the next level of the design is opened to determine whether the instances associated with the part should be ignored. Parts with the `nlAction` property set to `ignore` are excluded from the handle. When set to a value other than `nil`, this argument indicates that while opening an Open Access view, the next level of the design is not opened to determine whether the instances associated with the part should be ignored.

Value Returned

h_pcdbHandle	The pcdb handle of the configuration opened.
`nil`	The open operation failed.

Example

Changes the name space from the default value, `CDBA`, to `Verilog`.

```
pcdbOpen("lib" "cell" "schematic" "r" ?nameSpace "Verilog")
```

Includes ignored masters.

```
pcdbOpen("lib" "cell" "schematic" "r" ?dontOpenNextLevel t)
```

Note: An example for the use of the pcdb functions is at the end of this section.

© 2020 All Rights Reserved.

Reference

pcdbGetInstMasterGen
pcdbClose

An example for the use of this function can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

© 2020 All Rights Reserved.

pcdbViewHasPcdbInfo

```
pcdbViewHasPcdbInfo(
     t_libName
     t_cellName
     t_viewname
     [ t_nameSpace ]
     )
     => t / nil
```

Description

Determines whether the specified view has pcdb information.
If *t_nameSpace* is not provided, *t_libName*, *t_cellName*, and *t_viewName* are assumed to be in the name space.

Arguments

t_libName	The library name.
t_cellName	The cell name.
t_viewName	The view name.
t_nameSpace	The name space.

Value Returned

`t`	The view has pcdb information.
`nil`	The view does not have pcdb information.

Example

```
pcdbViewHasPcdbInfo("lib1 "cell1" "view1" "Verilog")
```


© 2020 All Rights Reserved.

pcdbSave

```
pcdbSave(
     h_pcdbHandle
     )
     => t / nil
```

Description

Save a pcdb handle opened with `pcdbOpen`. This function is currently not useful. It would require other functions that would modify a pcdb handle that do not currently exist.

Arguments

h_pcdbHandle	A pcdb handle returned from a call to pcdbOpen.

Value Returned

`t`	The pcdb object is saved.
`nil`	The save operation failed.

Reference

pcdbOpen

An example for the use of pcdb functions can be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

Example for all pcdb functions

```
; Open the pc.db file (Get a pcdb "handle")
pcdb = pcdbOpen("lib" "cell" "schematic" "r")
; get a "generator" that can be used to get the "masters"
masterGen = pcdbGetInstMasterGen(pcdb)
; get the first master
master = pcdbNextInstMaster(masterGen)
; loop through all the masters
while( master != nil
    ; get the lib, cell and view name of this master
    lib = pcdbInstMasterLib(master)
    cell = pcdbInstMasterCell(master)
```


© 2020 All Rights Reserved.

```
    view = pcdbInstMasterView(master)
    ; Get whether lib, cell, and view are "fixed".
    ; "fixed" means that any elaboration (say using a configuration)
    ; should NOT do anything that would replace the master name with a
    ; different name supplied via the elaboration process (e.g a configuration)
    libfixed = pcdbInstMasterLibFixed(master)
    cellfixed = pcdbInstMasterCellFixed(master)
    viewfixed = pcdbInstMasterViewFixed(master)
    ; For this demo code, just print the information.
    printf("Master: %s %s %s %L %L %L\n"
        lib cell view libfixed cellfixed viewfixed)
    ; get a "generator" that can be used to get the instances of this
    ; master
    instGen = pcdbGetInstGen(master)
    ; get the first instance
    inst = pcdbNextInst(instGen)
    ; loop through all the instances
    while( inst != nil
        ; get the instance name and width
        instName = pcdbInstName(inst)
        instWidth = pcdbInstWidth(inst)
        ; For this demo code, just print the information
        printf("    Inst: %s width: %d\n" instName instWidth)
        inst = pcdbNextInst(instGen)
    )
    master = pcdbNextInstMaster(masterGen)
)
; free up resources associated with this pcdb "handle"
pcdbClose(pcdb
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

16

Virtuoso RF Functions (ICADVM20.1 Virtuoso MultiTech Framework Only)

This chapter describes the following Virtuoso RF (VRF) database SKILL functions:

- Enable VRF and MultiTech Framework Functions
  - dbIsMultiTechEnabled
  - dbIsVRFEnabled
  - dbIsVRFInfraEnabled

  Note: These functions will be deprecated in a future release.
- Curved Shape Functions
  - dbCreateCurvedAreaBoundary
  - dbCreateCurvedPath
  - dbCreateCurvedPolygon
  - dbCreateMultipleCurvedPolygons
  - dbIsCurved
  - dbGetCurvedAreaBoundaryPoints
  - dbGetCurvedPolygonPoints
  - dbSetCurvedAreaBoundaryPoints
  - dbGetCurvedPolygonPoints
  - dbSetCurvedAreaBoundaryPoints
  - dbSetCurvedPolygonPoints
  - dbCreateRegionBoundary
  - dbGetRegionBoundaryLayer

© 2020 All Rights Reserved.

- dbGetRegionBoundaryLayerType
- dbGetRegionBoundaryName
- dbIsRegionBoundary

■ **VRF Package Infrastructure Functions**

- pkxAddFuncDefPin
- pkxCreateCompDef
- pkxCreateCompDefPin
- pkxCreateFuncDef
- pkxCreateFuncDefPin
- pkxDeleteCompDef
- pkxFindCompDef
- pkxFindFuncDefPin
- pkxFindFuncDef
- pkxFindFuncDefPin
- pkxGetCompDefPins
- pkxGetCompDefs
- pkxGetFuncDefPins
- pkxGetFuncDefs
- pkxGetProp
- pkxGetProps
- pkxSetProp

© 2020 All Rights Reserved.

Enable VRF and MultiTech Framework Functions

dbIsMultiTechEnabled

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

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Confirms whether the Virtuoso Multi Tech Framework functionality is enabled in the current session.

Note: This function will be deprecated in a future release.

Arguments

None.

Value Returned

`t`	Returns `t` when all of the following are true: ■ Virtuoso Multi Tech Framework is available ■ Virtuoso Multi Tech Framework license has been checked out ■ An advanced node and methodologists release, such as ICADVM20.1 is being used
`nil`	Virtuoso Multi Technology Solution framework is not enabled or you are not using an advanced node and methodologies release such as ICADVM20.1.

Example

```
dbIsMultiTechEnabled()
=> t
```


© 2020 All Rights Reserved.

dbIsVRFEnabled

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

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Confirms whether the VRF-related functionality is enabled in the current session.

Note: This function will be deprecated in a future release.

Arguments

None.

Value Returned

`t`	VRF-related functionality is enabled.
`nil`	VRF-related functionality is not enabled.

Example

```
when(dbIsVRFEnabled()
  pts = dbGetCurvedPolygonPoints(css())
)
```


© 2020 All Rights Reserved.

dbIsVRFInfraEnabled

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

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Confirms whether the VRF infrastructure sub-system is enabled in the current session. This includes the basic database SKILL functions required to work with the curved shapes, Allegro translators, and other related features.

Note: This function will be deprecated in a future release.

Arguments

None.

Value Returned

`t`	VRF infrastructure is enabled.
`nil`	VRF infrastructure is not enabled.

Example

```
when(dbIsVRFInfraEnabled()
  pts = dbGetCurvedPolygonPoints(css())
)
```


© 2020 All Rights Reserved.

Curved Shape Functions

dbCreateCurvedAreaBoundary

```
dbCreateCurvedAreaBoundary(
     d_cellViewId
     l_complexCurvedPoints
     )
     => d_curvedAreaId / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a curved area boundary in the specified cellview, using the specified points.

Arguments

d_cellViewId	Database ID of a cellview to hold the new object.
l_complexCurvedPoints	A list representing closed complex curve.

Value Returned

d_curvedAreaId	Database ID of a curved area object.
nil	The operation fails.

Example

```
curvedPoints = list(list(0 0) list(200 0) list(200 100 "arc" "ccw") list(200 200)
list(0 200))
areaB = dbCreateCurvedAreaBoundary(cv curvedPoints)
```


© 2020 All Rights Reserved.

dbCreateCurvedPath

```
dbCreateCurvedPath(
     d_cellViewId
     txl_layerPurpose
     l_curvedPoints
     x_width
     )
     => d_curvedPathId
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a curved path that has a straight centerline and round edges at all segments.

Arguments

d_cellViewId	Database ID of a cellview.
txl_layerPurpose	Name, number, or a list of layers or layer-purpose pairs. For more information, see Type txl_layerPurpose.
l_curvedPoints	List of points used to describe the curved path. *Important* Currently, only straight centerlines are supported for curved paths. Curved information in the centerline may cause unexpected results.
x_width	Width of the curved path.

Value Returned

d_curvedPathId	Database ID of the curved path that is created.

Example

```
cp3 = list(list(300 100) list(500 100) list(500 200) list(600 200) list(600 0))
path1 = dbCreateCurvedPath(cv list("DIE_PAD" "drawing") cp3)
```

A curved path is created based on the specified list of points and layer-purpose as `"DIE_PAD" "drawing"`.

© 2020 All Rights Reserved.

dbCreateCurvedPolygon

```
dbCreateCurvedPolygon(
     d_CellViewId
     txl_layerPurpose
     l_curvedPoints
     )
     => d_curvedPolygonId
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a curved polygon.

Arguments

d_cellViewId	Database ID of a cellview.
txl_layerPurpose	Name, number, or a list of layers or layer-purpose pairs. For more information, see Type txl_layerPurpose.
l_curvedPoints	List of points used to describe the curved polygon.

Value Returned

d_curvedPolygonId	Database ID of the curved polygon that is created.

Example

```
cp1 = list(list(0 0) list(200 0) list(200 100 "arc" "ccw") list(200 200) list(0
200))
dbCreateCurvedPolygon(cv list("DIE_PAD" "drawing") cp1)
```

A curved polygon is created based on the specified list of points and layer-purpose as `"DIE_PAD" "drawing"`.

© 2020 All Rights Reserved.

dbCreateMultipleCurvedPolygons

```
dbCreateMultipleCurvedPolygons(
     d_cellViewId
     txl_layerPurpose
     l_curvedPoints)
     => list(d_curvedPolygonIds) / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates multiple curved polygons in the specified cellview on the specified layer or layer-purpose pair using the specified points.

Arguments

d_cellViewId	Database ID of a cellview.
txl_layerPurpose	Name, number, or a list of layers or layer-purpose pairs. For more information, see Type txl_layerPurpose.
l_curvedPoints	List of points used to describe the curved polygon.

Value Returned

list(*d_curvedPolygonIds*)	List of IDs of created curved polygons.
nil	An error occurred.

Example

Creates multiple curved polygons in the cellview `cv`, with the layer and curved points as `layer` and `curvedPoints`.

```
polygons = dbCreateMultipleCurvedPolygons(cv layer curvedPoints)
```


© 2020 All Rights Reserved.

dbIsCurved

```
dbIsCurved(
     d_shapeId
     )
     => t / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Checks whether the specified shape is a curved shape.

Arguments

d_shapeId	Database ID of a shape.

Return value:

`t`	The specified shape is a curved shape.
`nil`	The specified shape is not a curved shape.

Example

```
cp2 = list(list(0 0) list(200 0) list(200 200) list(100 200 "arc" "ccw") list(0
200))
=> ((0 0)
     (200 0)
     (200 200)
    (100 200 "arc" "ccw")
    (0 200)
)

poly = dbCreateCurvedPolygon(cvId lpp cp2)
=> t
dbIsCurved(poly)
=> t
```

Confirms that the specified shape `poly` is a curved shape.

© 2020 All Rights Reserved.

dbGetCurvedAreaBoundaryPoints

```
dbGetCurvedAreaBoundaryPoints(
     d_curvedAreaBoundaryId
     )
     => l_complexCurvedPoints / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns curved area boundary points for the specified curved area boundary object.

Arguments

d_curvedAreaBoundaryId

Database ID of a curved area boundary object.

Value Returned

l_complexCurvedPoints

List representing curved points.

`nil` The operation failed.

Example

```
points = dbGetCurvedAreaBoundaryPoints(areaB)
```


© 2020 All Rights Reserved.

dbGetCurvedPolygonPoints

```
dbGetCurvedPolygonPoints(
     d_curvedPolygonId
     )
     => l_curvedPoints
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Retrieves the curved point list for the specified curved polygon.

Arguments

d_curvedPolygonId	Database ID of a curved polygon.

Value Returned

l_curvedPoints	List of curved points for the specified curved polygon.

Example

```
cp2 = list(list(0 0) list(200 0) list(200 200) list(100 200 "arc" "ccw") list(0
200))
=> ((0 0)
     (200 0)
     (200 200)
    (100 200 "arc" "ccw")
    (0 200)
)
dbSetCurvedPolygonPoints(poly1 cp2)
=> t
dbGetCurvedPolygonPoints(poly1)
=> ((0.0 0.0)
    (200.0 0.0)
    (200.0 200.0)
    (100.0 200.0 "arc" "ccw")
    (0.0 200.0)
   )
```

Retrieves the list of points for the polygon `poly1`.

© 2020 All Rights Reserved.

dbSetCurvedAreaBoundaryPoints

```
dbSetCurvedAreaBoundaryPoints
     d_curvedAreaBoundaryId
     l_complexCurvedPoints
     )
     => t / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Sets curved points for the specified curved area boundary object.

Arguments

d_curvedAreaBoundaryId

Database ID of a curved area boundary object.

l_complexCurvedPoints

List representing a complex closed curve.

Value Returned

`t`	Curved points were set for the specified curved area object.
`nil`	The operation failed.

Example

```
points = list(list(list(200.0 200.0 "arcCentralAngle" 180.0) list(0.0 200.0) list(0
0) list(200.0 0.0)) nil)
```


© 2020 All Rights Reserved.

dbSetCurvedPolygonPoints

```
dbSetCurvedPolygonPoints(
     d_polygonId
     l_curvedPoints
     )
     => t / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Sets the given curved point list for the specified polygon. If the polygon is not curved, it is converted to curved and the points are set.

Arguments

d_curvedPolygonId	Database ID of a regular or a curved polygon.
l_curvedPoints	List of curved points to be set on the specified curved polygon.

Value Returned

`t`	The given curved point list is set for the specified curved polygon.
`nil`	The operation fails.

Example

```
cp2 = list(list(0 0) list(200 0) list(200 200) list(100 200 "arc" "ccw") list(0
200))
=> ((0 0)
     (200 0)
     (200 200)
    (100 200 "arc" "ccw")
    (0 200)
    )

dbSetCurvedPolygonPoints(poly1 cp2)
=> t
; Sets the list of points for the polygon poly1.
```


© 2020 All Rights Reserved.

dbCreateRegionBoundary

```
dbCreateRegionBoundary(
     d_cellViewId
     l_curvedPoints
     t_regionName
     [ t_layerType ]
     )
     => d_regionBoundaryId / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a region boundary.

A region boundary is a named curved area boundary that is used to specify areas where some constraints apply. A region boundary also has a layerType. If the layerType is `"layer"`, a layer must be specified.

© 2020 All Rights Reserved.

Arguments

d_cellViewId	Database ID of a cellview.
l_curvedPoints	List of curved points to be set on the specified curved area boundary.
t_regionName	Name of region boundary.
t_layerType	Type of layer. Valid values: `"all"`, `"outsideLayers"`, `"insidePlaneLayers"`, `"insideSignalLayers"`, `"layer"`
tx_layer	Layer name or number. **Note:** This argument is only specified when the t_layerType is `"layer"`.

Value Returned

d_regionBoundaryId	Database ID of the region boundary that is created.
`nil`	An error occurred.

Example

```
points = list(list(0 0) list(200 0) list(200 100 "arc" "ccw") list(200 200) list(0
200))
regB1 = dbCreateRegionBoundary(cv points "rbName1" "all")
regB2 = dbCreateRegionBoundary(cv points "rbName2" "layer" "Metal1")
```


© 2020 All Rights Reserved.

dbGetRegionBoundaryLayer

```
dbGetRegionBoundaryLayer(
     d_regionBoundaryID
     )
     => tx_regionLayer / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns the layer name or number of the specified region boundary.

Note: This function requires layerType to be `"layer"`. For more information, see dbGetRegionBoundaryLayerType

Arguments

d_regionBoundaryID

Database ID of a region boundary.

Value Returned

tx_regionLayer	Layer name, if found, or number of the region boundary layer.
`nil`	An error occurred or the layer type is not `"layer"`.

Example

```
regionLayer = dbGetRegionBoundaryLayer( regbId )
=> "METAL4"
```


© 2020 All Rights Reserved.

dbGetRegionBoundaryLayerType

```
dbGetRegionBoundaryLayerType(
     d_regionBoundaryId
     )
     => t_regionLayerType / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns the region boundary layer type.

Arguments

d_regionBoundaryId

Database ID of a region boundary.

Value Returned

t_regionLayerType	Layer type of a region boundary. Valid values: `"all"`, `"layer"`, `"outsideLayers"`, `"insidePlaneLayers"`, `"insideSignalLayers"`
`nil`	An error occurred.

Example

```
layerType = dbGetRegionBoundaryLayerType( regbId )
=> "outsideLayers"
```


© 2020 All Rights Reserved.

dbGetRegionBoundaryName

```
dbGetRegionBoundaryName(
     d_regionBoundaryId
     )
     => t_regionName / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns region boundary name.

Arguments

d_regionBoundaryId

Database ID of a region boundary.

Value Returned

t_regionName	Name of the region boundary.
nil	An error occured.

Example

```
name = dbGetRegionBoundaryName( regbId )
=> "regionBoundary1"
```


© 2020 All Rights Reserved.

dbIsRegionBoundary

```
dbIsRegionBoundary(
     d_areaBoundaryId
     )
     => t / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Confirms if the specified area boundary is a region boundary.

ArgumentsV

d_areaBoundaryId	Database ID of an area boundary.

Value Returned

`t`	The specified area boundary is a region boundary.
`nil`	The specified area boundary is not a region boundary.

Example

```
isRegionBoundary = dbIsRegionBoundary( areaId )
=> t
```


© 2020 All Rights Reserved.

VRF Package Infrastructure Functions

The SKILL functions in this section are used to represent Allegro component definition (`compDef`) objects in Virtuoso.

A `compDef` object has the following attributes:

- **Device type**

  A unique string that identifies the component definition object.

- **Component class**

  Valid component classes are `IC`, `IO`, `DISCRETE`, `MECHANICAL`, `PLATING_BAR`, `DRIVER_CELL`, `INTERPOSER`, `IC_PACKAGE`, and `IO_PACKAGE`.

- **Package**

  A string identifying the package the compDef uses.

In a `compDef` package, a *component* is an instance that is identified by a unique string. The `compDef` package has a:

- List of physical pins that are identified by names, which are usually integers. Each pin is represented as a `compDefPin` object.
- List of functions, represented as `funcDef`. These functions have input and output pins that are represented as `funcDefPin` objects.

The `compDefPin` objects a mapped to a `funcDefPin` objects.

© 2020 All Rights Reserved.

pkxAddFuncDefPin

```
pkxAddFuncDefPin(
     pkx_compDefPinId
     pkx_funcDefPinId
     )
     => t / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Adds the specified `funcDefPin` object to the specified `compDefPin` object.

Arguments

`pkx_compDefPinId`	ID of a `compDefPin` object.
`pkx_funcDefPinId`	ID of a `funcDefPin` object.

Value Returned

`t`	The specified `funcDefPin` is added to `compDefPin`.
`nil`	The operation failed.

Example

Adds `funcDefPin` to `compDefPin`.

```
pkxAddFuncDefPin(compDefPin funcDefPin)
```


© 2020 All Rights Reserved.

pkxCreateCompDef

```
pkxCreateCompDef(
     d_bagId
     t_deviceType
     t_class
     t_package
     )
     => pkx_compDefId | nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a `compDef` object in the specified database and returns its ID.

Arguments

d_bagId	ID of the database containing the object.
t_deviceType	Name of the device type that identifies the `compDef` object. For example, `dtype`.
t_class	Placement class of the `compDef` object to be created. For example, `IO`. For more information, see Component class.
t_package	Name of the matching package symbol used by the `compDef` object.

Value Returned

pkx_compDefId	ID of the created `compDef` object.
`nil`	The operation failed.

Example

Creates a `compDef` object in the database `bag` with the device type as `dtype`, class as `IO`, and package as `package`.

```
compDef = pkxCreateCompDef(bag "dtype" "IO" "package")
```


© 2020 All Rights Reserved.

pkxCreateCompDefPin

```
pkxCreateCompDefPin(
     pkx_compDefId
     t_number
     )
     => pkx_compDefPinId | nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a `compDefPin` object and returns its ID.

Arguments

pkx_compDefId	ID of a `compDef` object.
t_number	Name of the pin, represented by a combination of a string and a number. For example, `I10`.

Value Returned

pkx_compDefPinId	ID of the created `compDefPin` object.
`nil`	The operation failed.

Example

Creates `compDefPin` on the `compDef` object with the pin number as `I1`.

```
compDefPin = pkxCreateCompDefPin(compDef "I1")
```


© 2020 All Rights Reserved.

pkxCreateFuncDef

```
pkxCreateFuncDef(
     pkx_compDefId
     t_slot
     t_type
     )
     => pkx_funcDefId / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a `funcDef` object in the specified `compDef` and returns its ID.

Arguments

pkx_compDefId	ID of a `compDef` object.
t_slot	Name of the function slot. For example, `G1`.
t_type	Type of function. For example, `A`.

Value Returned

pkx_funcDefId	ID of the created `funcDef` object.
`nil`	The operation failed.

Example

Creates the `funcDef` object in the `compDef` object with the function slot as `G1` and type as `type1`.

```
funcDef = pkxCreateFuncDef(compDef "G1" "type1")
```


© 2020 All Rights Reserved.

pkxCreateFuncDefPin

```
pkxCreateFuncDefPin(
     pkx_funcDefId
     t_number
     t_use
     x_swapCode
     )
     => pkx_funcDefPinId / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a `funcDefPin` object in the specified `funcDef` object and returns its ID.

Arguments

pkx_funcDefId	ID of a `funcDef` object.
t_number	Name of the pin, represented by a combination of a string and a number. For example, `I10`.
t_use	Value indicating the logical use of the `funcDefPin` object. Valid values: `IN`, `OUT`, `BI`, `TRI`, `OCA`, `OCL`, `POWER`, `GROUND`, `NC`, and `UNSPEC`.
x_swapCode	Code indicating whether the `funcDefPin` object can be swapped with another pin.

Value Returned

pkx_funcDefPinId	ID of the created `funcDefPin` object.
`nil`	The `funcDefPin` object was not created.

Example

Creates the `funcDefPin` object in the `funcDef` object and returns its ID.

```
funcDefPin = pkxCreateFuncDefPin(funcDef "A" "IN" 1)
```


© 2020 All Rights Reserved.

pkxDeleteCompDef

```
pkxDeleteCompDef(
     pkx_compDefId
     )
     => t / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Deletes the specified `compDef` object.

Arguments

`pkx_compDefId`	ID of a `compDef` object.

Value Returned

`t`	The specified `compDef` object was deleted.
`nil`	The specified `compDef` object was not deleted.

Example

The `compDefId` object is deleted.

```
pkxDeleteCompDef(compDefId)
```


© 2020 All Rights Reserved.

pkxFindCompDef

```
pkxFindCompDef(
     d_bagId
     t_deviceType
     )
     => pkx_compDefId / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Searches for a `compDef` object in the specified database with the specified device type and returns its ID if it exists.

Arguments

d_bagId	ID of the database containing the object.
t_deviceType	Type of device that identifies the component definition. For example, `dtype`.

Value Returned

pkx_compDefId	ID of the `compDef` object with the specified attributes.
`nil`	The specified `compDef` object was not found.

Example

Searches for the `compDef` object in the database `bag`, with the device type `dtype`.

```
compDef = pkxFindCompDef(bag "dtype")
```


© 2020 All Rights Reserved.

pkxFindCompDefPin

```
pkxFindCompDefPin(
     pkx_compDefId
     t_number
     )
     => pkx_compDefPinId | nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Searches for a `compDefPin` object in the specified `compDef` with the specified attributes, and returns its ID if it exists.

Arguments

pkx_compDefId	ID of a `compDef` object.
t_number	Name of the pin, represented by a combination of a string and a number. For example, `I10`.

Value Returned

pkx_compDefPinId	ID of the `compDefPin` object in the specified `compDef`.
`nil`	The specified `compDefPin` object does not exist in the specified `compDef` object or the operation failed.

Example

Searches for the `compDef` object with the pin number `I1`.

```
pkxFindCompDefPin(compDef "I1")
```


© 2020 All Rights Reserved.

pkxFindFuncDef

```
pkxFindFuncDef(
     pkx_compDefId
     t_slot
     t_type
     )
     => pkx_funcDefId / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Searches for a `funcDef` object in the specified `compDef` with the specified attributes, and returns its ID if it exists.

Arguments

pkx_compDefId	ID of a `compDef` object.
t_slot	Name of the function slot. For example, `G1`.
t_type	Type of function. For example, `A`.

Value Returned

pkx_funcDefId	ID of the `funcDef` object in the specified `compDef` object.
`nil`	The specified `funcDef` object does not exist in the specified `compDef` object or the operation failed.

Example

Searches for the `funcDef` object in the `compDef` object with the function slot `G1` and the function type `type1`.

```
funcDef = pkxFindFuncDef(compDef "G1" "type1")
```


© 2020 All Rights Reserved.

pkxFindFuncDefPin

```
pkxFindFuncDefPin(
     pkx_funcDefId | pkx_compDefPinId
     t_name
     t_use
     s_swapCode
     )
     => pkx_funcDefPinId | nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Searches for a `funcDefPin` object with the specified attributes, either in a `funcDef` or in a `compDefPin`, and returns its ID if it exists.

Arguments

pkx_funcDefId | *pkx_compDefPinId*	ID of a `funcDef` or `compDefPin`.
t_name	Name of the `funcDefPin` object.
t_use	Value indicating the logical use of the `funcDefPin` object. Valid values: `IN`, `OUT`, `BI`, `TRI`, `OCA`, `OCL`, `POWER`, `GROUND`, `NC`, and `UNSPEC`.
s_swapCode	Code indicating whether the `funcDefPin` object can be swapped with another pin.

Value Returned

pkx_funcDefPinId	ID of the `funcDefPin` pin object that was found in the `funcDef` or `compDefPin` object.
`nil`	The `funcDefPin` pin object was not found or the operation failed.

Example

Searches for the `funcDefPin` object `pin3` with name `A`, use `IN`, and swapcode `1` in the `funcDef` object.

```
pin3= pkxFindFuncDefPin(funcDef "A" "IN" 1)
```


© 2020 All Rights Reserved.

Searches for the `compDefPin` object `pin3` with name `A`, use `IN`, and swapcode `1` in the `funcDef` object.

```
pin3 = pkxFindFuncDefPin(compDefPin "A" "IN" 1)
```


© 2020 All Rights Reserved.

pkxGetCompDefPins

```
pkxGetCompDefPins(
     d_bagId
     )
     => list(pkx_compDefPinId) | nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns a list of `compDefPin` object IDs in the specified database.

Arguments

d_bagId	ID of the database containing the `compDefPin` objects.

Value Returned

`list(`*pkx_compDefPinId*`)`	List of `compDefPin` objects.
`nil`	Cannot find any `compDefPin` objects or the operation failed.

Example

Returns a list of `compDefPin` objects in the database `bag`.

```
pins = pkxGetCompDefPins(bag)
```


© 2020 All Rights Reserved.

pkxGetCompDefs

```
pkxGetCompDefs(
     d_bagId
     )
     => list(pkx_compDefId) / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns a list of `compDef` object IDs in the specified database.

Arguments

d_bagId	ID of the database containing the `compDef` objects.

Value Returned

`list(`*pkx_compDefId*`)`	List of `compDefIDs` in the specified database.
`nil`	Cannot find any `compDef` objects or the operation failed.

Example

Returns a list of `compDef` objects in the database `bag`.

```
pkxGetCompDefs(bag)
```


© 2020 All Rights Reserved.

pkxGetFuncDefPins

```
pkxGetFuncDefPins(
     d_bagId
     )
     => list(pkx_funcDefPinId) | nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns a list of `funcDefPin` object IDs in the specified database.

Arguments

d_bagId	ID of the database containing the objects.

Value Returned

`list(`*pkx_funcDefPinId*`)`	List of `funcDefPinIds` found in the specified database.
`nil`	Cannot find any `funcDefPin` objects or the operation failed.

Example

Returns a list of `funcDefPin` objects in the database `bag`.

```
pkxGetFuncDefPins(bag)
```


© 2020 All Rights Reserved.

pkxGetFuncDefs

```
pkxGetFuncDefs(
     d_bagId
     )
     => list(pkx_funcDefId) / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns a list of `funcDef` object IDs in the specified database.

Arguments

d_bagId	ID of the database containing the `funcDef` object.

Value Returned

`list(`*pkx_funcDefId*`)`	List of `funcDef` objects found in the specified database.
`nil`	Cannot find any `funcDef` objects or the operation failed.

Example

Returns a list of `funcDef` objects in the database `bag`.

```
pkxGetFuncDefs(bag)
```


© 2020 All Rights Reserved.

pkxGetProp

```
pkxGetProp(
     pkx_Obj
     t_propName
     )
     => pkx_Prop / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns the property of the specified package object and name.

Arguments

pkx_Obj	ID of the specified package object.
t_propName	Name of the property.

Value Returned

pkx_Prop	Property of the specified package object.
nil	The operation failed.

Example

Retrieves the property for the package object `compDef` with the property name `propList`.

```
prop = pkxGetProp(compDefPin "propList")
```


© 2020 All Rights Reserved.

pkxGetProps

```
pkxGetProps(
     pkx_Obj
     t_propName
     )
     => list(pkx_properties) / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns a list of properties for the specified package object.

Arguments

pkx_Obj	ID of a package object.
t_propName	Name of the property.

Value Reurned

`list(pkx_properties)`	List of properties.
`nil`	The operation failed.

Example

Retrieves the list of properties for the package object `compDef`.

```
compDefProps = pkxGetProps(compDef)
```


© 2020 All Rights Reserved.

pkxSetProp

```
pkxSetProp(
     pkx_Obj
     t_propName
     S_propType
     g_propValue
     )
     => pkx_property / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Sets the specified properties on the specified package object.

Arguments

pkx_Obj	ID of a package object.
t_propName	Name of the property.
S_propType	Type of the property.
g_propValue	Value of the property.

Value Returned

pkx_prop	The specified properties are set on the specified object.
`nil`	The operation failed.

Example

Sets the property name `"propInt"`, the property type `'int`, and the property value `100` on the package object `compDef`.

```
prop = pkxSetProp(compDef "propInt" 'int 100)
```

Sets the property name `"propString"`, the property type `"string"`, and the property value `"string prop value"` on the package object `compDef`.

```
prop = pkxSetProp(compDef "propString" "string" "string prop value")
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

17

Photonics Functions (ICADVM20.1 Photonics Only)

This chapter describes the following photonics functions:

- Photonics Database SKILL functions
- Curvy Core Functions
- Photonics Flow SKILL functions

Note: You need the `Virtuoso_Photonics_Option` license with Virtuoso_MultiTech_Framework (95022) to use the functionality discussed in this section. For more information, see *License Checkout for Virtuoso Photonics Platform* in License Checkout for Virtuoso Photonics Platform of *Virtuoso Software Licensing and Configuration User Guide.*

© 2020 All Rights Reserved.

Photonics Database SKILL functions

This section covers the following photonics database SKILL functions:

- dbCellViewHasPhotonicPinFig
- dbGetPhotonicPinFigAngle
- dbGetPhotonicPinFigRadius
- dbGetPhotonicPinFigWidth
- dbIsPhotonicsEnabled
- dbSetPhotonicPinFigAngle
- dbSetPhotonicPinFigRadius
- dbSetPhotonicPinFigWidth
- dbUnsetPhotonicPinFigAngle
- dbUnsetPhotonicPinFigRadius
- dbUnsetPhotonicPinFigWidth

© 2020 All Rights Reserved.

dbCellViewHasPhotonicPinFig

```
dbCellViewHasPhotonicPinFig(
     d_cvId
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified cellview has a photonic pinFig.

Arguments

d_cvId	Database ID of a cellview.

Return Value

`t`	Specified cellview has a photonic pinFig.
`nil`	Specified cellview does not have a photonic pinFig.

Example

```
dbCellViewHasPhotonicPinFig(cvId)
=> t
```

The specified cellview contains a photonic pinFig.

© 2020 All Rights Reserved.

dbGetPhotonicPinFigAngle

```
dbGetPhotonicPinFigAngle(
     d_pinFigId
     )
     => n_angle / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the attribute value of the angle for the specified photonics pinFig.

Arguments

d_pinFigId	Database ID of a photonics pinFig.

Value Returned

n_angle	Attribute value of the angle for the specified photonics pinFig.
`nil`	The pinFig is not a photonic pin or the attribute is not set.

Example

```
dbGetPhotonicPinFigAngle(pinFigId)
=> 15.5
```

Attribute value of the angle on a photonic pinFig `pinFigId` is `15.5`.

© 2020 All Rights Reserved.

dbGetPhotonicPinFigRadius

```
dbGetPhotonicPinFigRadius(
     d_pinFigId
     )
     => n_radius / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the attribute value of the radius for the specified photonics pinFig.

Arguments

d_pinFigId	Database ID of a photonics pinFig.

Value Returned

n_radius	Attribute value of the radius for the specified photonics pinFig.
`nil`	The pinFig is not a photonic pin or the attribute is not set.

Example

```
dbGetPhotonicPinFigRadius(pinFigId)
=> 5
```

Attribute value of the radius on a photonic pinFig `pinFigId` is `5`.

© 2020 All Rights Reserved.

dbGetPhotonicPinFigWidth

```
dbGetPhotonicPinFigWidth(
     d_pinFigId
     )
     => n_width / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the attribute value of the width for the specified photonics pinFig.

Arguments

d_pinFigId	Database ID of a photonics pinFig.

Value Returned

n_width	Attribute value of the width for the specified photonics pinFig.
`nil`	The pinFig is not a photonic pin or the attribute is not set.

Example

```
dbGetPhotonicPinFigWidth(pinFigId)
=> 1
```

Attribute value of the width on a photonic pinFig `pinFigId` is `1`.

© 2020 All Rights Reserved.

dbIsPhotonicsEnabled

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

Description

(ICADVM20.1 Photonics Only) Confirms whether the photonics-related functionality is enabled.

Arguments

None

Value Returned

`t`	Photonics-related functionality is enabled.
`nil`	Photonics-related functionality is not enabled.

Example

After launching Virtuoso, use the `dbIsPhotonicsEnabled()` to confirm if the photonics-related functionality is enabled

```
;Cellview "cv" has photonic pinFig but the function returns nil if photonics is disabled
dbCellViewHasPhotonicPinFig(cv)
=> nil
```


© 2020 All Rights Reserved.

dbSetPhotonicPinFigAngle

```
dbSetPhotonicPinFigAngle(
     d_pinFigId
     n_value
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Sets the given attribute value as the angle of the specified photonic pinFig.

Arguments

d_pinFigId	Database ID of a photonics pinFig.
n_value	Angle attribute value to be set.

Value Returned

`t`	Specified attribute value is set as the angle of the photonics pinFig.
`nil`	Angle attribute value is not set.

Example

```
dbSetPhotonicPinFigAngle(pinFigId 15.5)
=> t
```

`15.5` is set as the angle attribute value on the photonic pinFig `pinFigId`.

© 2020 All Rights Reserved.

dbSetPhotonicPinFigRadius

```
dbSetPhotonicPinFigRadius(
     d_pinFigId
     n_value
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Sets the given attribute value as the radius of the specified photonic pinFig.

Arguments

d_pinFigId	Database ID of a photonics pinFig.
n_value	Radius attribute value to be set.

Value Returned

`t`	Specified attribute value is set as the radius of the photonics pinFig.
`nil`	Radius attribute value is not set.

Example

```
dbSetPhotonicPinFigRadius(pinFigId 5)
=> t
```

`5` is set as the radius attribute value on the photonic pinFig `pinFigId`.

© 2020 All Rights Reserved.

dbSetPhotonicPinFigWidth

```
dbSetPhotonicPinFigWidth(
     d_pinFigId
     n_value
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Sets the given attribute value as the width of the specified photonic pinFig.

Arguments

d_pinFigId	Database ID of a photonics pinFig.
n_value	Width attribute value to be set.

Value Returned

`t`	Specified attribute value is set as width of the photonics pinFig.
`nil`	Width attribute value is not set.

Example

```
dbSetPhotonicPinFigWidth(pinFigId 5)
=> t
```

`5` is set as the width attribute value on the photonic pinFig `pinFigId`.

© 2020 All Rights Reserved.

dbUnsetPhotonicPinFigAngle

```
dbUnsetPhotonicPinFigAngle(
     pinFigId
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Resets the attribute value of the angle for the specified photonics pinFig.

Arguments

pinFigId	Database ID of a photonics pinFig.

Value Returned

`t`	Angle attribute value of the specified photonic pinFig is reset.
`nil`	Angle attribute value is not reset.

Example

```
dbUnsetPhotonicPinFigAngle(pinFigId)
=> t
```

Angle attribute value of the photonic pinFig `pinFigId` is reset.

© 2020 All Rights Reserved.

dbUnsetPhotonicPinFigRadius

```
dbUnsetPhotonicPinFigRadius(
     pinFigId
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Resets the attribute value of the radius for the specified photonics pinFig.

Arguments

pinFigId	Database ID of a photonics pinFig.

Value Returned

`t`	Radius attribute value of the specified photonic pinFig is reset.
`nil`	Radius attribute value is not reset.

Example

```
dbUnsetPhotonicPinFigRadius(pinFigId)
=> t
```

Radius attribute value of the photonic pinFig `pinFigId` is reset.

© 2020 All Rights Reserved.

dbUnsetPhotonicPinFigWidth

```
dbUnsetPhotonicPinFigWidth(
     pinFigId
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Resets the attribute value of the width for the specified photonics pinFig.

Arguments

pinFigId	Database ID of a photonics pinFig.

Value Returned

`t`	Width attribute value of the specified photonic pinFig is reset.
`nil`	Width attribute value is not reset.

Example

```
dbUnsetPhotonicPinFigWidth(pinFigId)
=> t
```

Width attribute value of the photonic pinFig `pinFigId` is reset.

© 2020 All Rights Reserved.

Curvy Core Functions

This section describes the following curvy core SKILL functions.

- Function to Control the Curvy Core Feature
  - ccIsEnabled
- Functions to Create Curvy Core Mathematical Objects
  - ccCreateCurve
  - ccCreateFacet
  - ccCreateLineSegment
  - ccCreatePolyCurve
- Functions to Create ccShapes
  - ccCreateLine
  - ccCreatePath
  - ccCreatePie
  - ccCreatePolygon
  - ccCreateRect
  - ccCreateSurface
  - ccCreateSurfaceFromPath
- Functions for ccShape Geometries
  - ccCopyFig
  - ccConvertIntoCurvy
  - ccGenFigs
  - ccGenOffsetFig
  - ccGetFromFig
  - ccMoveFig
  - ccRemoveFigs
- Functions for Curvy Core Mathematical Objects

© 2020 All Rights Reserved.

- ccOffsetPolyCurve
- ccRenameFacet

- Functions to Query Object Attributes for Curvy Core Mathematical Objects
  - ccGetCurvePoint
  - ccGetCurveDerivative
  - ccGetPolyCurveMinRadius
  - ccIsLinearCurve
- Functions to Query Object Attributes for ccShapes
  - ccGetAll
  - ccGetPathBoundary
  - ccGetPathCenterLine
  - ccGetPathEdge
  - ccFindSurfaceFacet
- Functions for Transformations
  - ccLayerAnd
  - ccLayerAndNot
  - ccLayerOr
  - ccReversePolyCurve
  - ccSmoothenFig
  - ccSmoothenPolyCurve
  - ccTransformFacet
  - ccTransformPolyCurve
- Functions for Connectors
  - ccBendConnector
  - ccCreateConnectorWidthExpressionList
  - ccCurveConnector
  - ccSpiralConnector

© 2020 All Rights Reserved.

- ccTromboneConnector
- ccWaypointConnector
- ccWideConnector

© 2020 All Rights Reserved.

Function to Control the Curvy Core Feature

ccIsEnabled

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

Description

(ICADVM20.1 Photonics Only) Checks if the CurvyCore engine is enabled.

Arguments

None

Value Returned

`t`	The CurvyCore engine is enabled.
`nil`	The CurvyCore engine is not enabled.

Examples

```
ccIsEnabled()
=> nil
ccCreateCurve("t" "1" 0:1)
=> *Error* (CC-200007): CurvyCore component is not initialized.
```


© 2020 All Rights Reserved.

Functions to Create Curvy Core Mathematical Objects

ccCreateCurve

```
ccCreateCurve(
     t_x
     t_y
     l_interval
     )
     => o_curve / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccCurve object using the specified parametric functions and interval.

Note: The function will return an error if you try to create a zero length curve.

Arguments

t_x	Mathematical expression determining the X-coordinate of a parametric function. Supported operations are: +, -, :, *, ^ (power), sin, cos, tan, sinh, cosh, tanh, atan, exp, log, sqrt, cfresnel, sfresnel, pow.
t_y	Mathematical expression determining the Y-coordinate of a parametric function. Supported operations are: +, -, :, *, ^ (power), sin, cos, tan, sinh, cosh, tanh, atan, exp, log, sqrt, cfresnel, sfresnel, pow.
l_interval	`list(`*n_begin:n_end*`)` An interval for the input variable determined by its start *n_begin* and end *n_end*.

© 2020 All Rights Reserved.

Value Returned

o_curve	ID of the resulting ccCurve.
nil	An error occurred.

Examples

```
ccCreateCurve("t" "1" 0:1)
=> cc@0x269c83e0
```

```
ccCreateCurve("sin(t)" "s+t" 0:1)
=> *Error*  (CC-200053): Inconsistent variable name in string expression 's+t': s, t
```

```
ccCreateCurve("cos(t)" "sin(t)" -3.14:3.14)
=> cc@0x269c83f0
```


© 2020 All Rights Reserved.

ccCreateFacet

```
ccCreateFacet(
     l_origin
     [ ?angle n_angle ]
     [ ?radius n_radius ]
     [ ?offsets l_offsets ]
     [ ?name t_name ]
     )
     => o_facet / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccFacet object. A facet is a special type of straight segment that is specified by the origin point, offsets, curvature, and rotation angle. It can be used to implement a smooth stitching between two ccShape objects.

© 2020
All Rights Reserved.

Arguments

l_origin	`list(`*n_x n_y*`)`
	Origin points of the facet specified by *n_x* and *n_y* on the X-axis and Y-axis, respectively.
`?angle` *n_angle*	Angle between the facet and the positive X-axis.
	The default is `0`.
`?radius` *n_radius*	Curvature radius of the origin point.
	The default is `0`.
`?offsets` *l_offsets*	
	`list(`*n_lhs n_rhs*`)`
	Offsets from the origin point to the endpoints of the facet.
	The default is `(0:0)`.
`?name` *t_name*	Name of the facet.
	The default is `""`.

Value Returned

o_facet	ID of the resulting ccFacet
`nil`	An error occurred.

Example

```
ccCreateFacet(-2:0 ?offsets 2:2 ?angle 90)
> cc@0x269c83e0
```


© 2020 All Rights Reserved.

ccCreateLineSegment

```
ccCreateLineSegment(
     l_pt1
     l_pt2
     ) => o_curve / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccCurve object that represents a straight line segment connecting two specified points.

The parameter range of the segment is:

`t=[0,1]`

where, `t=0` is the first point and `t=1` is the second point.

Arguments

l_pt1	`list(`*l_x l_y*`)` The start point of the line segment.
l_pt2	`list(`*l_x l_y*`)` The end point of the line segment.

Value Returned

o_curve	ID of the resulting ccCurve.
`nil`	An error occurred.

Example

```
ccCreateLineSegment(-1:-1 1:1)
> cc@0x269c83f0
```


© 2020 All Rights Reserved.

ccCreatePolyCurve

```
ccCreatePolyCurve(
     l_segments
     [ ?close { t | nil } ]
     [ ?smoothness S_smoothness ]
     )
     => o_polyCurve / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccPolyCurve object from a list of specified polyCurve segments.

Arguments

l_segments

l_segment | (*l_segments...*)

A single curvy segment or a list of segments, specified by a ccCurve object and a transformation applied to this curve in the following format.

(*o_curve l_transform*)

where:

- *o_curve* is the ID of a ccCurve object.
- *l_transform* is a transformation object in the following format:

  ((*n_offsetX n_offsetY*) *n_angle*)

  where:

  - *n_angle* is the rotation angle.
  - *n_offsetX* and *n_offsetY* determine the offsets of the X-axis and Y-axis in this format (0,0).

?close

Specifies whether the resulting curve is closed (t) or open (nil).

The default is nil.

© 2020 All Rights Reserved.

`?smoothness` *`S_smoothness`*

Level of smoothness of the resulting curve.

Valid values:

- `G0` specifies that the resulting curve is continuous
- `G1` specifies that the derivative of the resulting curve is continuous.
- `G2` specifies that the second order derivative of the resulting curve is continuous.

Value Returned

`o_polyCurve`	ID of the resulting ccPolyCurve.
`nil`	An error occurred.

Examples

```
ccCreatePolyCurve(list(list(ccCreateCurve("cos(t)" "sin(t)" -
                  3.14159265:3.14159265) list(0:0 0)))
                  ?close t
                  )
=> cc@0x269c83f0

ccCreatePolyCurve(list(list(ccCreateCurve("t" "1" 0:1) list((0:0) 0))
                  list(ccCreateCurve("t" "1" 2:3) list((0:0) 0)))
                  ?smoothness 'G0
                  )
=> *Error* (CC-200014): Cannot create PolyCurve: required smoothness is G0, but
smoothness between segments 0 and 1 is discontinuous: point (1, 1) != (2, 1).
```


© 2020 All Rights Reserved.

Functions to Create ccShapes

ccCreateLine

```
ccCreateLine(
     d_cellViewId
     l_lpp
     g_polyCurveSegments
     )
     => o_line / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccLine object at the specified layer-purpose pair of the given cellview from either a ccPolyCurve object or a list of polyCurve segments forming a polyCurve. When list of polyCurve segments is specified the resulting polyCurve should be of class G2, otherwise, the function terminates with an error.

Arguments

d_cellViewId	Database ID of cellview.
l_lpp	*tx_layer* | (*tx_layer tx_purpose*) A layer or a layer-purpose pair.
g_polyCurveSegmen	*o_polyCurve* | (*l_polyCurveSegment*...) ID of an originating ccPolyCurve (*o_polyCurve*) or a list of polyCurve segments (*l_polyCurveSegment*) as specified in ccCreatePolyCurve.

Value Returned

o_line	ID of the resulting ccLine.
nil	An error occurred.

© 2020 All Rights Reserved.

Examples

```
ccCreateLine(cv "waveguide"
               ccCreatePolyCurve(list(list(ccCreateCurve("cos(t)" "sin(t)"
                -3.14159265:3.14159265)list(0:0 0)))))
=> cc@0x269c83f0
```

```
ccCreateLine(cv "waveguide" list(list(ccCreateCurve("t" "1" 0:1) list(0:0 0))
                                 list(ccCreateCurve("t" "1" 2:3) list(0:0 0))))
\e *Error* (CC-200014): Cannot create PolyCurve: required smoothness is G2, but
smoothness between segments 0 and 1 is discontinuous: point (1, 1) != (2, 1).
```


© 2020 All Rights Reserved.

ccCreatePath

```
ccCreatePath(
     d_cellviewId
     l_lpp
     l_segments
     [?defaultWidth g_defWidth]
     [?smoothness   S_smoothness]
     )
     => o_path / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccPath object on the specified layer-purpose pair from one or a list of path segments.

Arguments

d_cellViewId	Database ID of a cellview.
l_lpp	*tx_layer* | (*tx_layer tx_purpose*) A layer or a layer-purpose pair.

© 2020 All Rights Reserved.

l_segments *l_segment*|(*l_segments...*)

A single curvy segment or a list of segments, specified by a ccCurve object and a transformation applied to this curve in the following format.

(*o_curve l_transform* [*g_width*])

where:

- *o_curve* is the ID of a ccCurve object.
- *l_transform* is a transformation object in the following format:

  ((*n_offsetX n_offsetY*) *n_angle*)

  where:

  - *n_angle* is the rotation angle.
  - *n_offsetX* and *n_offsetY* determine the offsets of the X-axis and Y-axis in this format (0,0).
- *g_width* is the width of the specified segment. This width can be either of the following:
  - nil, which means that *defaultWidth* value is used for this segment.
  - A number, or an expression depending on the centerline-parameter of the segment *t*.

?defaultWidth *g_defWidth*

The default width value that can be set to a numeric value or nil.

The default is nil, in which case the width values should be individually defined in *l_segments*. An error occurs if either segment widths or ?defaultWidth is not set.

© 2020 All Rights Reserved.

`?smoothness` *`S_smoothness`*

Level of smoothness of the resulting curve.

Valid values:

- `G0` specifies that the resulting curve is continuous
- `G1` specifies that the derivative of the resulting curve is continuous.
- `G2` specifies that the second order derivative of the resulting curve is continuous.

Value Returned

`o_path`	ID of the resulting ccPath.
`nil`	An error occurred.

Examples

```
ccCreatePath(cv "waveguide"
             list(list(ccCreateCurve("cos(t)" "sin(t)" -3.14159265:3.14159265)
             list(0:0 0) "sqrt(t)"))
             )
=> cc@0x269c83f0

ccCreatePath(cv "waveguide"
             list(list(ccCreateCurve("t" "0" 0:1) list((0:0) 0))
             list(ccCreateCurve("1" "t" 0:1) list((0:0) 0))
             )
             ?defaultWidth 0.1)
=> *Error* (CC-200014): Cannot create PolyCurve: required smoothness is G1, but
smoothness between segments 0 and 1 is G0: tangent angle 0.000000 != 90.000000.

ccCreatePath(cv "waveguide"
             list(list(ccCreateCurve("t" "0" 0:1) list((0:0) 0))
             list(ccCreateCurve("1" "t" 0:1) list((0:0) 0)))
             ?defaultWidth 0.1
             ?smoothness 'G0
            )
=> *Error* (CC-200012): ccPath requires G1 smoothness or higher.
```


© 2020 All Rights Reserved.

ccCreatePie

```
ccCreatePie(
     d_cellviewId
     l_lpp
     [ ?angle n_angle ]
     [ ?startRadius n_startRadius ]
     [ ?length n_length ]
     [ ?endWidth n_endWidth ]
     [ ?endLength n_endLength ]
     [ ?offset l_offset ]
     [ ?rotation n_rotation ]
     [ ?genFigs { t | nil } ]
     )
     => o_surface / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a pie slice figure bounded by outer circle and optionally by an inner circle, two radiuses at given angles, and optionally by an outer capping of given length. The function generates different figures based on the input as follows:

When the startRadius is `nil`, the pie shape that is generated is a true pie shape, with the length being the radius.

When the startRadius is larger than `0.0`, a donut shape is generated, with the inner part radius determined by the value, while the length is used to represent the thickness of the donut.

When endWidth is specified, a termination is generated, caping the coupler by a shape that is conceptually the union of a rectangle and the last pie shape, with the inner-curved radius as specified. In this case, endWidth must be smaller than the arc defined by sum of radius and length and larger than the arc from radius.

If rotation or offset is specified, the resulting the object undergoes corresponding transformation.

© 2020 All Rights Reserved.

Arguments

d_cellViewId Database ID of a cellview.

l_lpp *tx_layer* | (*tx_layer tx_purpose*)

A layer or a layer-purpose pair.

?angle *n_angle* Angle between the starting radius and the positive X-axis.

The ending radius will have the angle *n_angle* so that the X-axis is the symmetry axis of the resulting figure, before the rotation or offset is applied.

The default is `30` degrees.

?startRadius *n_startRadius*

Radius of the inner circle.

The default is `nil`, which means there is no inner circle.

?length *n_length*

Difference between the radiuses of the outer circle and the inner circle representing donut thickness, or a circle radius when *g_startRadius* is `nil`.

?endWidth *n_endWidth*

Width of the capping rectangle.

The default is `nil`, which means that the width is not capped.

?endLength *n_endLength*

Length of the capping rectangle.

The default is `nil`, which means length is not capped.

?offset *l_offset*

`list(`*n_lhs n_rhs*`)`

Offset of the resulting figure.

The default is `(0,0)`.

?rotation *n_rotation*

Rotation of the resulting figure.

The default is `(0,0)`.

© 2020 All Rights Reserved.

`?genFigs`

Specifies whether geometry is generated for the resulting figure.

The default is `t`, which means that geometry is generated.

Value Returned

`o_surface`	ID of the resulting ccSurface.
`nil`	An error.occured.

Examples

```
ccCreatePie(cv "waveguide" ?angle 45 ?startRadius 2 ?length 5 ?genFigs nil)
=> cc@0x269c83e0

ccCreatePie(cv "waveguide" ?length 5 ?endLength 4)
=> *** ERROR *** end length is too small. It must be at least 5 to generate the cap.
```


© 2020 All Rights Reserved.

ccCreatePolygon

```
ccCreatePolygon(
     d_cellview
     l_lpp
     l_points
     ?offset l_offset
     ?rotation n_rotation
     ?genFigs g_genFigs
     )
     => o_surface | nil
```

Description

(ICADVM20.1 Photonics Only) Creates a polygon with the specified vertices as the ccSurface in the specified cellview and on the specified layer-purpose pair. The polygon is transformed according to the specified offset and rotation. Optionally, OpenAccess figures (polygons) are created.

Arguments

d_cellViewId	Database ID of a cellview.
l_lpp	A layer or a layer-purpose pair. *tx_layer* | (*tx_layer tx_purpose*)
l_points	List of points. list(*n_x n_y*) **Note:** If the list of points do not close the polygon, it is closed automatically.
?offset l_offset	Offset value for the surface. list(*n_lhs n_rhs*) The default is (0,0).
?rotation n_rotation	Rotation angle for the surface. The default is (0,0).

© 2020 All Rights Reserved.

?genFigs g_genFigs

Specifies whether OpenAccess figures are generated.

The default is `t`, which means that figures are generated.

Value Returned

o_surface	ID of the resulting ccSurface. The polygon is the boundary of the surface.
`nil`	An error occurred.

Example

```
cv = geGetEditCellView()
Polygon1 = ccCreatePolygon( cv list("waveguide" "drawing") list( 0:0 10:0 15:5
10:20 5:10 0:5))
cc:0x257b2970
Polygon2 = ccCreatePolygon( cv list("waveguide" "drawing") list( 0:0 10:0 15:5
10:20 5:10 0:5) ?offset 30:30 ?rotation 20.0)
cc:0x30225850
```

Two polygons are created as shown below:

© 2020 All Rights Reserved.

ccCreateRect

```
ccCreateRect(
     d_cellview
     l_lpp
     l_llPoint
     l_urPoint
     ?offset l_offset
     ?rotation n_rotation
     ?genFigs g_genFigs
     )
     => o_surface | nil
```

Description

(ICADVM20.1 Photonics Only) Creates a rectangle with the specified coordinates as a ccSurface object on the specified cellview and layer-purpose pair. The rectangle is then transformed according to the specified offset and rotation. Optionally, OpenAccess figures (polygons) are created.

Arguments

d_cellViewId	Database ID of a cellview.
l_lpp	A layer or a layer-purpose pair. *tx_layer* | (*tx_layer tx_purpose*)
l_llPoint	Lower-left point of the rectangle. list(*n_x n_y*)
l_urPoint	Upper-right point of the rectangle. list(*n_x n_y*)
?offset l_offset	Offset value for the rectangle transformation. list(*n_lhs n_rhs*) The default is (0,0).
?rotation n_rotation	Rotation angle for the rectangle transformation. The default is (0,0).

© 2020 All Rights Reserved.

?genFigs g_genFigs

Specifies whether OpenAccess figures are generated for the resulting ccSurface.

The default is `t`, which means that the figures are generated.

Value Returned

o_surface	ID of the resulting ccSurface.
`nil`	An error occurred.

Example

```
ccCreateRect(cv '("metal" "drawing") -1:-1 2:2 ?rotation 180)
> cc:0x20f7d91e
ccCreateRect(cv '("metal" "drawing") 0:0 5:10 ?genFigs nil)
> cc@0x26d92e0
```


© 2020 All Rights Reserved.

ccCreateSurface

```
ccCreateSurface(
     d_cellViewId
     l_lpp
     o_boundary
     [ l_holes ]
     )
     => o_surface / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccSurface object on the specified layer-purpose pair from the specified boundary, which is ccPolyCurve with positive area and a list of holes, which are ccPolyCurve objects with negative areas.

Arguments

d_cellViewId	Database ID of a cellview.
l_lpp	*tx_layer* | (*tx_layer tx_purpose*)
	A layer or a layer-purpose pair.
o_boundary	A ccPolyCurve object with positive area.
l_holes	(*o_holes...*)
	A list of ccPolyCurves with negative areas.

Value Returned

o_surface	ID of the resulting ccSurface.
nil	An error occurred.

Examples

```
circ = ccCreatePolyCurve(ccCreateCurve("cos(t)" "sin(t)" -3.14159265:3.14159265)
                         ?close t)                          )
=> cc@0x269c83f0

ccCreateSurface(cv "waveguide" circ
                list(ccTransformPolyCurve(ccReversePolyCurve(circ) '((0 0) 0.2))))
=> cc@0x28f45441
```


© 2020 All Rights Reserved.

ccCreateSurfaceFromPath

```
ccCreateSurfaceFromPath(
     o_path
     [ ?facetNames l_facetNames  ]
     )
     => o_surface | nil
```

Description

(ICADVM20.1 Photonics Only) Creates a ccSurface object from the boundary of the given ccPath. Begin and end facets of the ccPath object in the resulting ccSurface boundary are renamed according to the value of facetNames, or are not renamed if facetNames is `nil`.

Arguments

o_path	ID of a ccPath object.
?facetNames *l_facetNames*	`list(`*t_beginFacet t_endFacet*`) | nil` List of new names for begin (*t_beginFacet*) and end facets (*t_endFacet*) of the ccPath object.

Value Returned

o_surface	ID of the resulting ccSurface
`nil`	An error occurred.

Example

In the following example, we create a ccPath object, check the old names of begin and end facets, call `ccCreateSurfaceFromPath`, and then check the new facet names in the resulting object.

```
path = ccCreatePath(cv "waveguide" list(list(ccCreateCurve("cos(t)" "sin(t)" -
3.14159265:3.14159265) list(0:0 0) 0.1)))
> cc@0x22ffefd0

path~>beginFacet~>name
> "begin"
```


© 2020 All Rights Reserved.

```
path~>endFacet~>name
> "end"

surf = ccCreateSurfaceFromPath(path ?facetNames '("begin1" "end1"))
> cc@0x252fc5b0

// The list of segments of the path's boundary includes two arcs // (indexes 0 and
2) and two facets (indexes 1 and 3).

nth(1 surf~>boundary~>segments)~>objType
> "ccFacet"

nth(1 surf~>boundary~>segments)~>name
> "end1"

nth(3 surf~>boundary~>segments)~>objType
> "ccFacet"

nth(3 surf~>boundary~>segments)~>name
> "begin1"
```


© 2020 All Rights Reserved.

Functions for ccShape Geometries

ccCopyFig

```
ccCopyFig(
     o_shape
     [ ?cv d_cellViewID ]
     [ ?lpp g_lpp ]
     [ ?transform l_transform ]
     )
     => o_shape2 / nil
```

Description

(ICADVM20.1 Photonics Only) Copies the specified ccShape object to the specified cellview and layer-purpose pair (LPP) by using the specified transform. The copied cellview and LPP default to the ones of the source ccShape.

Note: The ccShape object copy is transient (without any OpenAccess geometry attached) even when the source oaShape is persistent (with OpenAccess geometry attached).

Arguments

o_shape	ID of a ccShape object.
?cv *d_cellViewId*	ID of a cellview.
?lpp *g_lpp*	A layer or a layer-purpose pair in the specified cellview in the following format: *tx_layer* | (*tx_layer tx_purpose*)
?transform *l_transform*	A transformation object in the following format: ((*n_offsetX n_offsetY*) *n_angle*) where: ■ *n_angle* is the rotation angle. ■ *n_offsetX* and *n_offsetY* determine the offsets of the X-axis and Y-axis in this format (0,0).

© 2020 All Rights Reserved.

Value Returned

o_shape2	ID of the destination ccShape.
nil	An error occurred.

Example

```
arc = ccCreateLine(cv "metal"
                      ccCreatePolyCurve(list(list(ccCreateCurve("cos(t)" "sin(t)"
                                                                 0:3.14159265)))))
> cc@0x271bf960
;Create OpenAccess geometry
ccGenFigs(arc)
> (db:0x225edc9a)

arc~>figs
> (db:0x225edc9a)

arc2 = ccCopyFig(arc ?transform list((0:0) 90))
> cc@0x290dba90
;Check whether the OpenAccess geometry of the copied figure is empty
arc2~>figs
> nil

arc2~>beginPt
> (0.0 1.0)

arc2~>endPt
> (0.0 -1.0)
```


© 2020 All Rights Reserved.

ccConvertIntoCurvy

```
ccConvertIntoCurvy(
     d_fig
     )
     => o_owner / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a new ccShape to which the specified database figure is attached.

Note: This function supports rectangles and truncated pathSegs.

Argument

d_fig	Database ID of a figure.

Value Returned

o_owner	ID of the new ccShape to which the specified figure is attached.
nil	The operation failed.

Examples

```
ccConvertIntoCurvy(dbCreateRect(cv "waveguide" list(0:0 1:0.5)))
=> cc@0x15a2201f

ccConvertIntoCurvy(dbCreateDot(cv "waveguide" 1:1))
=> *Error* (CC-200028): oaShapes other than rectangles and truncated pathSegs are
not supported in this version.
```


© 2020 All Rights Reserved.

ccGenFigs

```
ccGenFigs(
     o_shape
     [ ?attach { t | nil } ]
     [ ?snapToMFG { t | nil } ]
     [ ?accuracy { n_accuracy | nil } ]
     )
     => l_figs / nil
```

Description

(ICADVM20.1 Photonics Only) Generates geometry for the specified ccShape and optionally attaches it to the originating ccShape.

Note: This attachment is accessible through the `ccShape~>"figs"` accessor.

Arguments

o_shape
: ID of a ccShape object.

`?attach { t | nil }`
: Specifies whether to attach the resulting figures to the originating ccShape.

  The default value `t` specifies that the resulting figures are attached to the originating ccShape.

`?snapToMFG { t | nil }`
: Specifies whether to snap the output figures to the manufacturing grid.

  The default value `t` specifies that the output figures are snapped to the manufacturing grid.

`?accuracy { n_accuracy | nil }`
: Discretization accuracy for retrieving the accuracy from the manufacturing grid.

  The default value `nil` specifies that using the step of the manufacturing grid as the discretization accuracy value.

© 2020 All Rights Reserved.

Value Returned

l_figs	List of DB figure IDs.
nil	The operation failed.

Example

```
line = ccCreateLine(cv "waveguide"
                    ccCreatePolyCurve(list(ccCreateCurve("t" "0" -1:1)
                    ccCreateCurve("cos(t)" "sin(t)" 0:3.14159265))
                    ?close t
                    )
=> cc@0x269c83f0

figs = ccGenFigs(line)
=> (db:0x2166d79a)

(car figs)~>objType
=> "line"

(car figs)~>nPoints
=> 10
```


© 2020 All Rights Reserved.

ccGenOffsetFig

```
ccGenOffsetFig(
     o_path
     g_lpp
     ?side t_side
     [ ?inner n_value | ?outer n_value | ?inner n_value | ?outer n_value ]
     )
     => o_shape / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a new ccShape in the same cellview as the specified path on the specified layer-purpose pair. This shape is constructed by connecting two offsets of the edge of the specified path by using the inner and outer values, respectively, with two facets. The origins of these facets are the same as the origins of the facets of the specified path.

Arguments

o_path	ID of a ccPath object. The edges and facets of this path are used to generate an offset ccShape.
g_lpp	A layer or a layer-purpose pair in the specified cellview in the following format: *tx_layer* | (*tx_layer tx_purpose*)
?side *t_side*	Defines the edge and offset direction. Valid values: "left", "right"

© 2020 All Rights Reserved.

`?inner` *`n_value`* | `?outer` *`n_value`* | `?inner` *`n_value`* | `?outer` *`n_value`*

The inner and outer offset values.

The values for the offset can be both negative and positive. The default is `0`.

A positive offset value implies that the offset direction of the specified `side` is to the right, and a negative value implies that it is to the left.

This is an optional value, however, the value for the `inner` offset should be less than the value of the `outer` offset. Therefore, one of these values need to be specified.

Value Returned

`o_shape`	ID of the created ccShape.
`nil`	An error occurred.

Example

Creates a ccPath with a varying discontinuous width.

```
path = ccCreatePath(
  deGetCellView()
  "y0"
  list(
    list(ccCreateCurve("t" "0" -1:0) list(0:0 0) 0.2)
    list(ccCreateCurve("t" "0"  0:2) list(0:0 0) "0.2+t/10")
    list(ccCreateCurve("t" "0"  0:1) list(2:0 0) 0.3)
  )
)
```

Creates a shape from the left edge of the path offset by `0.1` to the left and the left edge of the path connected with facets:

```
ccGenFigs(ccGenOffsetFig(path "y1" ?side "left" ?outer 0.1))
```

Creates a shape from the right edge of the path offset by `0.2` to the left and the right edge of the path offset by `0.3` to the right connected with facets:

```
ccGenFigs(ccGenOffsetFig(path "y2" ?side "right" ?inner -0.2 ?outer 0.3))
```


© 2020 All Rights Reserved.

ccGetFromFig

```
ccGetFromFig(
     d_fig
     )
     => o_owner / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the ccShape object attached to the given database figure.

Note: This operation is the reverse of `obj~>figs` accessor for a ccShape object.

Arguments

d_fig	Database ID of a figure.

Value Returned

o_owner	ID of the ccShape object to which the specified figure is attached.
`nil`	The figure is not attached to a ccShape object.

Example

```
figs = ccGenFigs(path)
=> (db:0x2558af35 db:0x2558af3d)

ccGetFromFig(car figs)
=> cc@0x15a2201f

path
=> cc@0x15a2201f
```


© 2020 All Rights Reserved.

ccMoveFig

```
ccMoveFig(
     o_shape
     [ ?cv d_cellViewId ]
     [ ?lpp g_lpp ]
     [ ?transform l_transform ]
     )
     => o_shape2 / nil
```

Description

(ICADVM20.1 Photonics Only) Moves the specified ccShape to the specified cellview and layer-purpose pair (LPP) using the specified transform. The cellview and LPP default to the ones of the source ccShape.

Note: Only transient ccShapes (without OpenAccess geometry attached) can be moved. Moving a persistent (with OpenAccess geometry attached) ccShape results in error.

Arguments

`o_shape`	ID of a ccShape object.
`?cv` *`d_cellViewId`*	ID of a cellview.
`?lpp` *`g_lpp`*	A layer or a layer purpose pair in the specified cellview in the following format *`tx_layer`* `|` `(`*`tx_layer tx_purpose`*`)`
`?transform` *`l_transform`*	A transformation object in the following format: `((`*`n_offsetX n_offsetY`*`)` *`n_angle`*`)` where: *`n_angle`* is the rotation angle. *`n_offsetX`* and *`n_offsetY`* determine the offsets of the X-axis and Y-axis in this format `(0,0)`.

© 2020 All Rights Reserved.

Value Returned

o_shape2	ID of the destination ccShape.
`nil`	An error occurred.

Example

```
arc = ccCreateLine(cv "metal"
                   ccCreatePolyCurve(list(list(ccCreateCurve("cos(t)" "sin(t)"
                                                             0:3.14159265)))))
> cc@0x250fff00

arc = ccMoveFig(arc ?transform list((0:0) 0))
> cc@0x28d15dc0
; Create OpenAccess geometry
ccGenFigs(arc)
> (db:0x222ddc9a)

arc2 = ccMoveFig(arc ?transform list((0:0) 90))
> *Error* (CC-200006): Moving a persistent ccFig cc:0x28d15dc0 is not allowed.
```


© 2020 All Rights Reserved.

ccRemoveFigs

```
ccRemoveFigs(
     o_shape
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Removes the database figures attached to the given ccShape.

Argument

o_shape	ID of a ccShape object.

Value Returned

t	Attached database figures are removed from the specified ccShapes or no figures are attached.
nil	An error occurred.

Example

```
ccRemoveFigs(pathID)
=> t
```


© 2020 All Rights Reserved.

Functions for Curvy Core Mathematical Objects

ccOffsetPolyCurve

```
ccOffsetPolyCurve(
     o_polyCurve
     n_value
     [ ?key g_key ]
     )
     => o_polyCurve2
```

Description

(ICADVM20.1 Photonics Only) Returns a copy of the specified polyCurve that is expanded by the specified offset value.

Arguments

o_polyCurve	ID of a ccPolyCurve object.
n_value	The offset value. A positive offset value implies that the offset direction is to the right, and a negative value implies that it is to the left.
?key *g_key*	A flag that outputs a polyCurve with `"raw"` offset when set to a value that is not `nil`. A `"raw"` offset polyCurve has cusps, which can be useful for debugging. The default is `nil`.

Value Returned

o_polyCurve2	ID of the resulting ccPolyCurve.
`nil`	An error occurred.

Example

```
arc = ccCreatePolyCurve(list(list(ccCreateCurve("cos(t)" "sin(t)" 0:3.14159265))))
> cc@0xc70cee6c

arc2 = ccOffsetPolyCurve(arc 1)
```


© 2020 All Rights Reserved.

```
> cc@0x86ca13ca
; A circle arc of radius 1 gets expanded to an arc of radius 2.
arc2~>beginPt
> (2.0 0.0)

arc2~>endPt
> (-2.0 0.0)
```


© 2020 All Rights Reserved.

ccRenameFacet

```
ccRenameFacet(
     o_facet
     t_name
     )
     => o_facet2 / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a copy of the specified facet with a different name.

Arguments

o_facet	ID of a ccFacet object.
t_name	A new name for the destination facet.

Value Returned

o_facet2	ID of the resulting ccFacet.
nil	An error occurred.

Example

```
rfacet = ccCreateFacet(2:0 ?angle -90 ?offsets 1:1 ?name "right")
> cc@0x86ca13ca

tfacet = ccRenameFacet(ccTransformFacet(rfacet '((0 0) 90)) "top")
> cc@0xc70cee6c

tfacet~>name
"top"
```


© 2020 All Rights Reserved.

Functions to Query Object Attributes for Curvy Core Mathematical Objects

ccCubicSplineConnector

```
ccCubicSplineConnector(
     l_points
     [ ?d1 n_d1 ]
     [ ?d2 n_d2 ]
     )
     => o_polyCurve / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a polyCurve representing a cubic spline function in the form where *y* is a function of *x* through the given list of points `(x,y)` with strictly increasing coordinates *x*.

```
y = f(x)
```

Optionally, *n_d1* and *n_d2* specify the first derivatives of the function `f(x)` at the first and the last point, respectively. Otherwise, if the first derivative at any of the endpoints is not specified, the second derivative of the function `f(x)` at the corresponding endpoint is zero.

Arguments

l_points	List of *N* > 3 points with strictly increasing coordinates `n_x1` <...< *n_xN*. `List(list(`*n_x1 n_y1*`)...list (`*n_xN n_yN*`))`
`?d1` *n_d1*	First derivative of the cubic spline function `f(x)` at the first point.
`?d2` *n_d2*	First derivative of the cubic spline function `f(x)` at the last point.

© 2020 All Rights Reserved.

Value Returned

o_polyCurve	ID of the resulting ccPolyCurve.
nil	An error occurred.

Example

Create a cubic spline through four points with coordinates `(2, 4)`,`(5, 2)`,`(7, 3)`,`(10, 1)` with zero first derivative at the first point and zero second derivative at the last point:

```
ccCubicSplineConnector(list(2:4 5:2 7:3 10:1) ?d1 0)
> cc@0x1b6a029
```


© 2020 All Rights Reserved.

ccGetCurvePoint

```
ccGetCurvePoint(
     g_curve
     n_t
     )
     => l_xy
```

Description

(ICADVM20.1 Photonics Only) Returns a point on a curve or a polyCurve segment at the specified value of the curve function parameter.

Arguments

g_curve

ID of a ccCurve object or a segment of a ccPolyCurve object in the form:

`list(o_curve l_transform)`

where,

- *o_curve* is the ID of a ccCurve object.
- *l_transform* is the transformation in the format:

  `list(list(n_x n_y) n_angle)`

  where:

  - *n_angle* is the rotation angle around `(0, 0)`.
  - *n_x* and *n_y* determine the offsets along the X-axis and the Y-axis in this format `(0,0)`.

Value Returned

`l_xy`

Coordinates of the point on a curve or a polyCurve segment.

Example

```
c=ccCreateCurve("t" "t^2" 0:1)
> cc@0xe74d47d9
```


© 2020 All Rights Reserved.

```
ccGetCurvePoint(c 0.5)
> (0.5 0.25)
ccGetCurvePoint(list(c list(1:1 180)) 0.5)
> (0.5 0.75)
```


© 2020 All Rights Reserved.

ccGetCurveDerivative

```
ccGetCurveDerivative(
     g_curve
     n_t
     [ x_order ]
     )
     => l_xy
```

Description

(ICADVM20.1 Photonics Only) Returns a derivative of a curve or a polyCurve segment at the specified value of the curve function parameter.

Arguments

g_curve

ID of a ccCurve object or a segment of a ccPolyCurve object in the form:

`list(o_curve l_transform)`

where,

- *o_curve* is the ID of a ccCurve object.
- *l_transform* is the transformation in the format:

  `list(list(n_x n_y) n_angle)`

  where:

  - *n_angle* is the rotation angle around `(0, 0)`.
  - *n_x* and *n_y* determine the offsets along the X-axis and the Y-axis in this format `(0,0)`.

n_t

The value of the curve function parameter.

x_order

A non-negative integer specifying the order of the derivative. The default is `1`.

© 2020 All Rights Reserved.

Value Returned

l_xy	Components of the derivative of a curve or a polyCurve segment.

Example

```
c=ccCreateCurve("t" "t^2" 0:1)
> cc@0xe74d47d9
ccGetCurveDerivative(c 0.5)
> (1.0 1.0)
ccGetCurveDerivative(list(c list(1:1 180)) 0.5)
> (-1.0 -1.0)
```


© 2020 All Rights Reserved.

ccGetPolyCurveMinRadius

```
ccGetPolyCurveMinRadius(
     o_polyCurve
     )
     => n_minRadius
```

Description

(ICADVM20.1 Photonics Only) Returns the minimum curvature radius of a polyCurve, ignoring any tangent discontinuity points.

Arguments

o_polyCurve	ID of a ccPolyCurve object.

Value Returned

n_minRadius	Minimum curvature radius of a polyCurve. Zero return value corresponds to zero maximum curvature, which is the infinite curvature radius.

Example

```
;Create a ccPolyCurve object
sinus = ccCreatePolyCurve(ccCreateCurve("t" "sin(t)" 0:8*atan(1)))
> cc@0x3afa4cca
;Return the minimum radius of the ccPolyCurve object
ccGetPolyCurveMinRadius(sinus)
> 1.0
```


© 2020 All Rights Reserved.

ccIsLinearCurve

```
ccIsLinearCurve(
     o_curve
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Checks if the given ccCurve is linear.

Arguments

o_curve	ID of a ccCurve object.

Value Returned

`t`	The specified curve is linear.
`nil`	The specified curve is not linear.

Example

```
ccIsLinearCurve(ccCreateCurve("t" "1" 0:1))
=> t
ccIsLinearCurve(ccCreateCurve("cos(t)" "sin(t)" -3.14:3.14))
=> nil
```


© 2020 All Rights Reserved.

Functions to Query Object Attributes for ccShapes

ccGetAll

```
ccGetAll(
     d_cellViewId
     )
     => l_shapes / nil
```

Description

(ICADVM20.1 Photonics Only) Returns all persistent ccShapes in the specified cellview.

Arguments

d_cellViewId	Database ID of a cellview.

Value Returned

l_shapes	List of ccShape IDs.
nil	An error occurred.

Example

```
ccGetAll(cv)
=> (cc@0x269c83f0)
```


© 2020 All Rights Reserved.

ccGetPathBoundary

```
ccGetPathBoundary(
     o_path
     )
     => o_boundary / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the ccBoundary object of the specified ccPath.

Arguments

o_path	ID of a ccPath object.

Value Returned

o_boundary	ID of a ccBoundary object.
nil	An error occurred.

Example

```
ccGetPathBoundary(ccConvertIntoCurvy(dbCreatePathSeg(cv "waveguide" -1:-1 0.5:0.5
1 "truncate" "truncate")))
=> cc@0x15a2201f
```


© 2020 All Rights Reserved.

ccGetPathCenterLine

```
ccGetPathCenterLine(
     o_path
     )
     => o_centerLine / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the center line of the path as an open ccPolyCurve.

Arguments

o_path	ID of a ccPath object.

Value Returned

o_centerLine	ID of a ccPolyCurve object forming the center line.
nil	An error occurred.

Example

```
path = ccCreatePath(cv "waveguide"
                    list(list(ccCreateCurve("t" "1" -1:0)
                         list(0:0 0))
                    list(ccCreateCurve("sin(t)" "cos(t)" 0:3.14)
                         list(0:0 0)))
                    ?defaultWidth 0.01
                    )
=> cc@0x2596b240

ccGetPathCenterLine(path)
=> cc@0x15a2820b
```


© 2020 All Rights Reserved.

ccGetPathEdge

```
ccGetPathEdge(
     o_path
     S_side
     )
     => o_edge / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a new ccPolyCurve that is either the left or the right edge of the boundary of the specified path.

Arguments

o_path	ID of a ccPath object.
S_side	Side of the boundary. Valid values: '`left` and '`right`

Value Returned

o_edge	ID of a ccPolyCurve object forming the specified edge of the boundary.
`nil`	The operation failed.

Example

```
path = ccCreatePath(cv "waveguide"
                    list(list(ccCreateCurve("t" "0" 0:10)
                         list(0:0 0) "sqrt(t)"))))
=> cc@0x262f8170

caar(ccGetPathEdge(path 'left)~>segments)~>??
(cc@0x28f45441 objType "ccCurve" xExpr "t" yExpr "(0.5*sqrt(t))" range (0.0 10.0))
caar(ccGetPathEdge(path 'right)~>segments)~>??
(cc@0xe9c99bef objType "ccCurve" xExpr "t" yExpr "((-0.5)*sqrt(t))" range (0.0
10.0))
```


© 2020 All Rights Reserved.

ccFindSurfaceFacet

```
ccFindSurfaceFacet(
     o_surface
     t_name
     )
     => o_facet / nil
```

Description

(ICADVM20.1 Photonics Only) Searches for a facet with the specified name on the specified surface.

Arguments

o_surface	ID of a ccSurface object.
t_name	Name of a facet.

Value Returned

o_facet	ID of the found facet.
nil	The facet is not found.

Example

```
lfacet = ccCreateFacet(2:0 ?offsets 1:1 ?name "left")
=> cc@0x15a2201f
rfacet = ccRenameFacet(ccTransformFacet(lfacet rot90) "right")
=> cc@0x15a22027
surf = ccCreateSurface(ccCreatePolyCurve(list(lfacet curve rfacet)))
=> cc@0x258af018
ccFindSurfaceFacet(suff "left")
=> cc@0x15a2201f
```


© 2020 All Rights Reserved.

Functions for Transformations

ccLayerAnd

```
ccLayerAnd(
     g_shapes1
     g_shapes2
     [ ?cellView d_cellViewId ]
     [ ?lpp l_lpp ]
     )
     => l_surface / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a list of new surface shapes derived from the areas of shapes of the first input argument *g_shapes1* that overlap areas of shapes of the second input argument *g_shapes1*.

Arguments

g_shapes1 *o_shape* | (*l_shapes...*)

A shape or a list of shapes of the first input argument.

g_shapes2 *o_shape* | (*l_shapes...*)

A shape or a list of shapes of the second input argument.

?cellView *d_cellViewId*

A cellview ID in which output shapes are created. If not specified, output shapes are created in the cellview of the first shape of the first input argument *g_shapes1*.

?lpp *l_lpp*

tx_layer | (*tx_layer tx_purpose*)

A layer-purpose pair in the given cellview on which output shapes are created. If not specified, output shapes are created on the layer-purpose pair of the first shape of the first input argument *g_shapes1*.

© 2020 All Rights Reserved.

Value Returned

l_surface	List of resulting surface shapes.
nil	Returns an empty list.

Example

```
c=ccCreateCurve("3*cos(t)" "3*sin(t)" 0:8*atan(1))
=> cc@0x36adc1b
s1=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(0:2 0)) ?close t))
=> cc@0x2d609190
s2=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(-sqrt(3):-1 0)) ?close
t))
=> cc@0x20480bc0
s3=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(sqrt(3):-1 0)) ?close
t))
=> cc@0x29143150
s=ccLayerAnd(s1 list(s2 s3))
=> (cc@0x27eeea10)
```


© 2020 All Rights Reserved.

ccLayerAndNot

```
ccLayerAndNot(
     g_shapes1
     g_shapes2
     [ ?cellView d_cellViewId ]
     [ ?lpp l_lpp ]
     )
     => l_surface / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a list of new surface shapes derived from the areas of shapes of the first input argument *g_shapes1* that do not overlap any areas of shapes of the second input argument *g_shapes2*.

Arguments

g_shapes1 *o_shape* | (*l_shapes...*)

A shape or a list of shapes of the first input argument.

g_shapes2 *o_shape* | (*l_shapes...*)

A shape or a list of shapes of the second input argument.

?cellView *d_cellViewId*

A cellview ID in which output shapes are created. If not specified, output shapes are created in the cellview of the first shape of the first input argument *g_shapes1*.

?lpp *l_lpp*

tx_layer | (*tx_layer tx_purpose*)

A layer-purpose pair in the given cellview on which output shapes are created. If not specified, output shapes are created on the layer-purpose pair of the first shape of the first input argument *g_shapes1*.

© 2020 All Rights Reserved.

Value Returned

l_surface	List of resulting surface shapes.
nil	Returns an empty list.

Example

```
c=ccCreateCurve("3*cos(t)" "3*sin(t)" 0:8*atan(1))
=> cc@0x36adc1b
s1=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(0:2 0)) ?close t))
=> cc@0x2d609190
s2=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(-sqrt(3):-1 0)) ?close
t))
=> cc@0x20480bc0
s3=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(sqrt(3):-1 0)) ?close
t))
=> cc@0x29143150
s=ccLayerAndNot(s1 list(s2 s3))
=> (cc@0x27eeea10)
```


© 2020 All Rights Reserved.

ccLayerOr

```
ccLayerOr(
     g_shapes1
     g_shapes2
     [ ?cellView d_cellViewId ]
     [ ?lpp l_lpp ]
     )
     => l_surface / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a list of new surface shapes derived by combining the areas of shapes of the first input argument *g_shapes1* and the areas of shapes of the second input argument *g_shapes2*.

Arguments

g_shapes1 *o_shape* | (*l_shapes...*)

A shape or a list of shapes of the first input argument.

g_shapes2 *o_shape* | (*l_shapes...*)

A shape or a list of shapes of the second input argument.

?cellView *d_cellViewId*

A cellview ID in which output shapes are created. If not specified, output shapes are created in the cellview of the first shape of the first input argument *g_shapes1.*

?lpp *l_lpp*

tx_layer | (*tx_layer tx_purpose*)

A layer-purpose pair in the given cellview on which output shapes are created. If not specified, output shapes are created on the layer-purpose pair of the first shape of the first input argument *g_shapes1*.

© 2020 All Rights Reserved.

Value ReturnedReturn

l_surface	List of resulting surface shapes.
nil	Returns an empty list.

Example

```
c=ccCreateCurve("3*cos(t)" "3*sin(t)" 0:8*atan(1))
=> cc@0x36adc1b
s1=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(0:2 0)) ?close t))
=> cc@0x2d609190
s2=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(-sqrt(3):-1 0)) ?close
t))
=> cc@0x20480bc0
s3=ccCreateSurface(cvId "y0" ccCreatePolyCurve(list(c list(sqrt(3):-1 0)) ?close
t))
=> cc@0x29143150
s=ccLayerOr(list(s1 s2) list(s3))
=> (cc@0x27eeea10)
```


© 2020 All Rights Reserved.

ccMirrorFacet

```
ccMirrorFacet(
     o_facet
     )
     => o_facet2 / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a mirrored or reversed copy of the given ccFacet. If the facet is viewed as a segment, it reverses the direction, which mean that it goes from the end of the original facet to its beginning. However, this function also updates the radius attribute in such a way that two G2-connected paths have mirrored facets at the connection point.

Arguments

o_facet	ID of a ccFacet object.

Value Returned

o_facet2	ID of the mirrored facet.
nil	The operation failed.

Example

```
cMirrorFacet(lfacet)
=> cc@0x15a2201f
```


© 2020 All Rights Reserved.

ccReversePolyCurve

```
ccReversePolyCurve(
     o_polyCurve
     )
     => o_polyCurve2 / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a reversed copy of the given polyCurve, which is the polyCurve with the same geometry but in the opposite direction, which is from the end of the original polyCurve to its beginning.

This function can be used to turn positive-closed polyCurves into negative polyCurves, and negative-closed polyCurves into positive polyCurves.

Arguments

o_polyCurve	ID of a ccPolyCurve object.

Value Returned

o_polyCurve2	ID of the reversed ccPolyCurve object.
nil	An error occurred.

Example

```
rpc = ccReversePolyCurve(ccCreatePolyCurve(ccCreateCurve("t" "0" 0:1)))
=> cc@0x269c83f0
caar(rpc~>segments)~>??
=> (cc@0x7a0766dd objType "ccCurve" xExpr "t" yExpr "0" range (1.0 0.0))
```


© 2020 All Rights Reserved.

ccSmoothenFig

```
ccSmoothenFig(
     o_shape
     n_radius
     )
     => o_surface / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the result of smoothening of the specified ccShape as a new transient ccSurface.

Arguments

o_shape	ID of a ccShape object.
n_radius	Radius of the turn made by the bend. With positive radius, this function smoothens concave cavities of the ccShape object. With negative radius, it smoothens convex bumps.

Value Returned

o_surface	ID of the resulting ccSurface object.
nil	An error occurred.

Example

```
ccSmoothenFig(path 1)
=> cc@0x15a2820b
```


© 2020 All Rights Reserved.

ccSmoothenPolyCurve

```
ccSmoothenPolyCurve(
     o_polyCurve
     n_radius
     )
     => o_polyCurve2 | nil
```

Description

(ICADVM20.1 Photonics Only) Returns a smoothened copy of the given ccPolyCurve.

Arguments

o_polyCurve	ID of a ccPolyCurve object.
n_radius	Radius of the turn made by the bend. With positive radius, this function smoothens concave cavities of the ccShape object. With negative radius, it smoothens convex bumps.

Value Returned

o_polyCurve2	ID of the smoothened ccPolyCurve object.
nil	An error occurred.

Example

```
ccSmoothenPolyCurve(pc)

=> cc@0x269c83f0
```


© 2020 All Rights Reserved.

ccTransformFacet

```
ccTransformFacet(
     o_facet
     l_transform
     )
     => o_facet2 | nil
```

Description

(ICADVM20.1 Photonics Only) Returns a transformed copy of the given ccFacet object.

Arguments

o_facet	ID of a ccFacet object.
l_transform	((*n_offsetX n_offsetY*) *n_angle*) A transformation object specified by an offset specified by *n_offsetX* and *n_offsetY* and a rotation angle *n_angle* around (0,0).

Value Returned

o_facet2	ID of the transformed ccFacet object.
nil	The operation failed.

Example

```
ccTransformFacet(lfacet list(0:0 90))
=> cc@0x15a2201f
```


© 2020 All Rights Reserved.

ccTransformPolyCurve

```
ccTransformPolyCurve(
     o_polyCurve
     l_transform
     )
     => o_polycurve2 | nil
```

Description

(ICADVM20.1 Photonics Only) Returns a transformed copy of the given ccPolyCurve.

Arguments

o_polycurve	ID of a ccPolyCurve object.
l_transform	((*n_offsetX n_offsetY*) *n_angle*) A transformation object specified by an offset specified by *n_offsetX* and *n_offsetY* and a rotation angle *n_angle* around (0,0).

Value Returned

o_polycurve2	ID of the transformed polyCurve object.
nil	The operation failed.

Example

```
pc = ccCreatePolyCurve(ccCreateCurve("t" "0" 0:1))
=> cc@0xa1d42cee
tpc = ccTransformPolyCurve(pc list(0:0 90))
=> cc@0xe9835dc
pc~>segments
=> ((cc@0x4fe6429d ((0.0 0.0) 0.0)))
tpc~>segments
=> ((cc@0x4fe6429d ((0.0 0.0) 90.0)))
```


© 2020 All Rights Reserved.

Functions for Connectors

ccBendConnector

```
ccBendConnector(
     l_port
     n_radius
     n_bangle
     n_rmin
     n_gamma
     )
     => l_connector / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a bend connector making a circular turn from the specified starting port with the radius, bend angle, and gamma fraction given in ccWaypointConnector. The connector consists of a central circular arc and two apex clothoids matching its curvature at one end and with zero curvature at the other end. Gamma is the ratio of the circular bend angle to the total bend angle.

If the bend radius is specified as `0`, the bend connector is created with the radius at the interface between the clothoids and the central circular arc equal to `rmin`. If you set gamma to `1-(cangle/bangle)` bend connectors are created containing fixed clothoid sections with angle varying from `0` to `cangle` and radius varying from `0` to `rmin`.

© 2020 All Rights Reserved.

Constraints

- `0 < |bangle| < 2*PI`
- `0 <= gamma <= 1`
- `gamma = 0` or `gamma = 1` when `bangle > PI`

Arguments

l_port

`list(nil 'xy l_point 'facet l_facet) | nil`

The starting port.

where,

- *l_point*
  `list(n_x n_y)`
  A list of two-dimensional point represented by *n_x* and *n_y*.
- *l_facet*
  `?list(nil 'angle n_angle ['radius n_radius] ['width n_width]`
  A facet is specified by the angle *n_angle*. Values for *n_radius* and *n_width* are not required and are ignored if specified.

n_radius

Radius of the turn made by the bend.

n_bangle

Angle of the turn made by the bend.

Valid values: Greater than `0` and less than the product of `2` and `Pi`.

n_rmin

Minimum allowed curvature radius.

n_gamma

A numeric value between `0` and `1` indicating the ratio of the circular bending angle to the overall bending angle.

- When the value is `1`, the connector consists of a single circular arc.
- When the value is `0`, the connector consists of two clothoids.

© 2020 All Rights Reserved.

Value Returned

l_connector	`list(nil 'connector` *o_polyCurve* `'message` *t_message*`)` The resulting connector (ccPolyCurve ID) bundled with debug or error information. where, ■ *o_polyCurve* specifying the resulting ccPolyCurve connector. ■ *t_message* is a string holding the debug information or an error message describing the reason why the connector calculation was not successfully completed.
`nil`	An error occurred.

Example

```
port = '(nil xy (3000  250) facet (nil angle 20))
radius = 100
bangle = -300
rmin = 0.1
gamma = 1
ccBendConnector(port radius bangle rmin gamma)
```

```
=> (nil connector cc@0x293deeb9 message "Length=523.598776 MinRadius=100.000000")
```


© 2020 All Rights Reserved.

ccCreateConnectorWidthExpressionList

```
ccCreateConnectorWidthExpressionList(
     n_startWidth
     n_endWidth
     S_taperStyle
     o_polyCurve
     [ ?startClamp n_startClampLength ]
     [ ?endClamp n_endClampLength ]
     )
     => l_segmentsWidths
```

Description

(ICADVM20.1 Photonics Only) Creates a list of curved segments with the corresponding width expressions defining a tapered connector path of the specified tapering style with the specified start and end width around the centerline specified by the input polyCurve.

The segments of the input polyCurve are assumed to be parametrized by the arc length in the same way as the segments created by the connector generation functions ccBendConnector, ccCurveConnector, and ccWaypointConnector.

The tapering width function can optionally be clamped to have zero derivative at either or both the ends of the connector with the length of the clamping regions specified by the specified clamp lengths.

The width function `w(t)` of the connector is represented by the following equation:

```
w(t) = f(t) + g(t)
```

where,

- `t` is the arc length measured from the start of the connector.
- `f(t)` is the non-clamped tapering width function (`linear`, `exp`, `parabolic`, and `log`)
- `g(t)` is the clamping function represented by the following equation:

  ```
  g(t) = C1 * exp(-t / D1) + C2 * exp(t / D2) + C3 + C4 * t
  ```

  where,

  - `D1` is *`n_startClampLength`*
  - `D2` = *`n_endClampLength`*
  - `C1`, `C2`, `C3`, and `C4` are calculated as follows:

© 2020 All Rights Reserved.

At each end of the connector:

```
w(t) = f(t)
```

At each clamped-end of the connector:

```
w'(t) = 0
```

The following is the plot of the function `w(t)`, which is the width of the connector as a function of the arc length:

Arguments

n_startWidth	The start width of the connector.
n_endWidth	The end width of the connector.
S_taperStyle	The taper style of the connector. Valid values: `'linear`, `'exp`, `'parabolic`, and `'log`.
o_polyCurve	A polyCurve specifying the centerline of the connector with the segments parametrized by the arc length.
n_startClampLength	Length of the clamping region at the start of the connector.
n_endClampLength	Length of the clamping region at the end of the connector.

© 2020 All Rights Reserved.

Value Returned

l_segmentsWidths List of polyCurve segments, including the width expressions, defining a tapered connector path of the specified tapering style in the following format:

`((o_curve l_transform) t_width))`

where,

o_curve is the ID of a ccCurve object.

l_transform is a transformation object in the following format:

`((n_offsetX n_offsetY) n_angle)`

where:

n_offsetX and *n_offsetY* determine the offsets of the X-axis and Y-axis respectively in the format `(0,0)`.

n_angle is the rotation angle.

t_width is the width expression specifying the width of the segment.

Example

```
c = ccBendConnector('(nil xy (0 0) facet (nil angle 30)) 50 120 20 0.5)
> (nil connector cc@0xda456df6 message "Length=116.804672 MinRadius=37.180082")


s = ccCreateConnectorWidthExpressionList(5 20 'exp c->connector ?startClamp 3.0
?endClamp 3.0)
> ((cc@0xb1fb573d
      ((-7.105427e-15 -3.552714e-15) -60.0)
"5*exp((0.01186848385793852)*(1*t+(0)))+(0.2021274047854202)*exp(-t/(3))+(-
9.0738813184265e-18)*exp(t/(3))+(-0.2021274047854202)+(0.008033382305447485)*t"
    )
    (cc@0x722f07af
      ((43.30127 2.724799) -120.0)
"5*exp((0.01186848385793852)*(1*t+(38.93489058117659)))+(4.66898511029175e-
07)*exp(-t/(3))+(-3.928220028334695e-12)*exp(t/
(3))+(0.1106514562739377)+(0.008033382305447485)*t"
    )
    (cc@0xa0bb9ee6
      ((86.60254 -1.776357e-14) 60.0) "5*exp((0.01186848385793852)*(-
1*t+(116.8046717435298)))+(2.49124859626004e-18)*exp(-t/(-3))+(-
0.7362091783926537)*exp(t/(-3))+(0.7362091783926537)+(-0.008033382305447485)*t"
```


© 2020 All Rights Reserved.

```
    )
 )

ccGenFigs(ccCreatePath(cv "y1" s))
> (db:0x21d0d49b)
```

© 2020 All Rights Reserved.

ccCurveConnector

```
ccCurveConnector(
     l_port1
     l_port2
     n_rmin
     [ ?method x_method ]
     [ ?parameters l_parameters ]
     )
     => l_connector | nil
```

Description

(ICADVM20.1 Photonics Only) Creates a curve connector with two specified ports and a minimum curvature radius. Depending on the input, the connector consists of different combinations of line, circle, and clothoid curve segments.

Arguments

l_port1 `list(nil 'xy l_point 'facet l_facet)`

Starting port for the connector where:

- `l_point`
  `list(n_x n_y)`
  A two-dimensional point represented by *n_x* and *n_y*.
- *l_facet*
  `?list(nil 'angle n_angle ['radius n_radius] ['width n_width]`
  A facet is specified by the angle *n_angle*. Values for *n_radius* and *n_width* are not required and are ignored if specified.

© 2020 All Rights Reserved.

l_port2 `list(nil 'xy l_point 'facet l_facet)`

Ending port for the connector.

- `l_point`
  `list(n_x n_y)`

  A two-dimensional point represented by *n_x* and *n_y*.

- *l_facet*
  `list(nil 'angle n_angle ['radius n_radius] ['width n_width]`
  A facet is specified by the angle *n_angle*. Values for *n_radius* and *n_width* are not required and are ignored if specified.

n_rmin Minimally allowed curvature radius.

`?method` *x_method*

Method number from `0` to `9`.

The default is `0`. For more information see Specifying Method Values.

`?parameters` *l_parameters*

`list(`*n_parameters*`...)`
List of additional numeric parameters passed to the connector calculation methods.

The default is `nil`.

© 2020 All Rights Reserved.

Value Returned

l_connector	`list(nil 'connector` *o_polyCurve* `'message` *t_message* `['segments` *l_segments*`])` The resulting connector (ccPolyCurve ID) bundled with debug or error logs, where, ■ *o_polyCurve* is the resulting ccPolyCurve connector. ■ *t_message* is a string holding the debug printouts and error messages. ■ *l_segments* is a list of segment lengths and minimum or maximum curvatures rather than the segments themselves (`o_polyCurve~>segments`) and is only set if *l_method*`>0`. If *l_method*`=0` or in case of error, *l_segments* is set to `nil`
`nil`	The operation failed.

Example

```
port1 = '(nil xy (-5 0) facet (nil angle   0 radius 1))
port2 = '(nil xy  (0 1) facet (nil angle -90 radius 1))
ccCurveConnector(port1 port2 0.01)
> (nil connector cc@0x20fc10ea message "Method=6 Parameter=1.000000
Length=8.322122 MinRadius=0.145183")
```

Specifying Method Values

ccCurveConnector optionally accepts a calculation method number and a list of numeric values that is interpreted as a parameter value or as list of weights of the cost function depending on the value, which is between `0` and `9`.

Method number	Result
`0` or unspecified	All applicable methods are tried for all applicable values of parameters.
`1`	LineLine
`2`	SmallCircleLine
`3`	LargeCircleLine with varying clothoid length

© 2020 All Rights Reserved.

4	SmallCircles with symmetric clothoids, 180 degree maximum turn at the ends and varying line length
5	SmallCircles with unsymmetric clothoids, 180 degree maximum turn at the ends and varying line length
6	LargeCircles with varying clothoid length
7	SmallCircles with symmetric clothoids, 360 degree maximum turn at the ends and varying line length
8	SmallCircles with unsymmetric clothoids, 360 degree maximum turn at the ends and varying line length.
9	LargeSmallCircle with 360 degree maximum turn at the ends.

Unspecified or 0 Method Value

If `?method` is not specified or is set to `0`, then all applicable methods are tried for all applicable values of parameters. In this case, if *`l_parameters`* is empty, then the result is the solution found first. If *`l_parameters`* is not empty, then it is assumed to contain list of four weights of cost function.

```
l_parameters = list(n_A n_B n_C n_D)
```

The result is the solution with the smallest value of the cost function defined by:

$$\int [A + Be^{-C/k} + D(dk/ds)^2]\, ds$$

where `k=1/R` is the curvature and `s` is the arc length.

Non-zero Method Malue

If a non-zero method number is specified, then only this particular method is used. In this case, if `l_parameters` is non-empty, then the method is applied for only for one value of the parameter given by the first element of *`l_parameters`*. Otherwise, if *`l_parameters`* is empty, then multiple applicable values of the parameter are tried out until the first solution is found.

© 2020 All Rights Reserved.

ccSpiralConnector

```
ccSpiralConnector(
     n_lengthInnerSpiral0
     n_lengthOuterSpiral1
     x_numberOfTurns
     n_minRadius
     n_gamma
     [ ?portLocation t_location]
     )
     => l_connector / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a spiral connector that consists of an inner S-shaped figure of two joined circular bend connectors. Each endpoint of S is prolonged by a straight extension and then by N-turnings of a spiral circling around S.

© 2020 All Rights Reserved.

Arguments

n_lengthInnerSpiral0

Size of the innermost turn of the spiral, *L0*.

n_lengthOuterSpiral1

Size of the outermost turn of the spiral *L!*.

x_numberOfTurns Number of turns on the left and right part of the spiral. Total number of turns is `2 * x_N`.

n_minRadius Minimum allowed curvature radius.

n_gamma A numeric value between `0` and `1`, indicating a ratio of the circular bending angle to the overall bending angle.

When the value is `1`, each turn of spiral consists of a single circular arc.

When the value is `0`, each turn of spiral consists of two clothoids.

?portLocation *t_location*

Location of the ports.

Valid values: `"center"`, `"side"`

The default is `"center"`, which means the default spiral connector with the location of the ports as shown in the above figure.

When set to `"side"` the default spiral connector is extended by additional connector segments (shown in red) such that the ports are located at the sides at the top of the spiral.

© 2020 All Rights Reserved.

Value Returned

l_connector	`list(nil 'connector` *o_polyCurve* `'message` *t_message*`)` The resulting connector and a message string where, ■ *o_polyCurve* is the ID of the resulting ccPolyCurve connector. ■ *t_message* is a string holding the debug printouts and error messages.
`nil`	The operation failed.

Example

```
ccSpiralConnector(50 100 3 6 0)
=> (nil connector cc@0xc8c2798a message "Length=2162.349132 MinRadius=6.000000")
```


© 2020 All Rights Reserved.

ccTromboneConnector

```
ccTromboneConnector(
     l_legs
     n_rmin
     n_gamma
     )
     => l_connector / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a trombone-shaped connector that consists of five legs of given lengths, smoothly connected by waypoint connectors with the given gamma ratio and the minimum curvature radius.

Arguments

l_legs	(*n_l1 n_l2 n_l3 n_l4 n_l5*) List of five length values for the legs of the trombone.
n_rmin	Minimally allowed curvature radius.
n_gamma	A numeric value between `0` and `1`, indicating a ratio of the circular bending angle to the overall bending angle. ■ When the value is `1`, each turn of trombone consists of a single circular arc. ■ When the value is `0`, each turn of trombone consists of two clothoids.

© 2020 All Rights Reserved.

Value Returned

l_connector	`list(nil 'connector o_polyCurve 'message t_message)` The resulting connector (ccPolyCurve ID) and a message string. where, ■ *o_polyCurve* is the resulting ccPolyCurve connector. ■ *t_message* is a string holding the debug printouts and error messages.
`nil`	The operation failed.

Example

```
ccTromboneConnector('(50 50 50 50 50) 12 0)
=> (nil connector cc@0xc7e73ea2 message "Length=221.267246 MinRadius=12.000000")
```


© 2020 All Rights Reserved.

ccWaypointConnector

```
ccWaypointConnector(
     l_port1
     l_port2
     l_points
     n_rmin
     n_gamma
     )
     => l_connector / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a waypoint connector between the specified points with the given minimum curvature radius and the given portion (gamma) of the circular bending angle in the overall bending angle.

If both ports are `nil`, then at least three waypoints must be specified as input. The connector starts at the first waypoint with the tangent given by the line between the first two waypoints and ends at the last waypoint with the tangent given by the line between the last two waypoints.

If both ports are not `nil`, specify the initial and the final points, tangents, and radiuses of the connector.

If no waypoints are specified, the start point and the end point are connected by a curve connector.

If a single waypoint is specified, it is connected to the start and the end point by a pair of curve connectors with zero curvature at the waypoint.

If two waypoints are specified, they are connected to each other with a straight line and to the start and the end point with a pair of curve connectors with zero curvatures at the waypoints.

If three or more waypoints are specified, they are connected to each other by a waypoint connector like in the case where both ports are `nil` and to the start and the end point by a pair of curve connectors with zero curvature at the waypoints.

It is an error to only specify one port as `nil` and the other port as not `nil`.

The waypoint connector consists of a sequence of lines, circles, and clothoids. If the calculation of the waypoint connector can only be partially completed, then the result will only contain segments between the starting point and the point where the calculation failed.

© 2020 All Rights Reserved.

Arguments

l_port1

`list(nil 'xy l_point 'facet l_facet) | nil`

Starting port for the connector.

Unlike ccCurveConnector, ports in waypoint connector are allowed to be simultaneously set to `nil`. However, a nil port specified with a non-nil port will result in an error.

l_port2

`list(nil 'xy l_point 'facet l_facet) | nil`

Ending port for the connector.

Unlike in ccCurveConnector, ports in waypoint connector are allowed to be simultaneously set to `nil`. However, a nil port + non-nil port will result in an error.

l_point

`list(n_x n_y)`

A two-dimensional point.

l_facet

`list(nil 'angle n_angle ['radius n_radius] ['width n_width]`

A facet is specified by the angle *n_angle*. Values for *n_radius* and *n_width* are not required and are ignored if specified.

l_points

(*l_point*...)

A list of two-dimensional points.

n_rmin

Minimally allowed curvature radius.

n_gamma

A numeric value between `0` and `1`, indicating a ratio of the circular bending angle to the overall bending angle.

- When the value is `1`, each bend of the connector consists of a single circular arc.
- When the value is `0`, each bend of the connector consists of two clothoids.

© 2020 All Rights Reserved.

Value Returned

l_connector	`list(nil 'connector` *o_polyCurve* `'message` *t_message* `['incomplete` *t_incomplete*`])` The resulting connector (ccPolyCurve ID) bundled with debug or error logs. where, ■ *o_polyCurve* is the resulting ccPolyCurve connector. If *t_incomplete* is not `nil`, then the connector only contains segments between the starting point and the point where the calculation failed. ■ *t_message* is a string holding the debug printouts and error messages. ■ *t_incomplete* is a string describing why connector can only be partially calculated. If the complete connector could be calculated, the value is `nil`.
`nil`	The operation failed.

Example

```
port1 = '(nil xy ( 100  100) facet (nil angle  60 radius 0))
port2 = '(nil xy (1000 -100) facet (nil angle 120 radius 0))
waypoints = '((500 300))
rmin = 20
gamma = 1
wp=ccWaypointConnector(port1 port2 waypoints rmin gamma)
=> (nil connector cc@0x196cc479 message "Method=1 Method=1 Length=1496.947836
MinRadius=71.306912")
p = ccCreatePath(geGetEditCellView() list("y1" "drawing") wp->connector->segments
?defaultWidth 0.5)
ccGenFigs(p)
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

ccWideConnector

```
ccWideConnector(
     g_connector
     l_lengthWidths
     n_clamp1
     n_clamp2
     )
     => l_segmentWidths
```

Description

(ICADVM20.1 Photonics Only) Creates a specification of a path around the specified connector centerline with the width specified by a cubic spline function.

Arguments

g_connector

A polyCurve or a list of connected polyCurve segments representing the connector centerline.

o_polyCurve | (*l_segment...*)

Each polyCurve segment is assumed to be parametrized by the ascending or descending arc length and an arbitrary initial value.

l_segment

A polyCurve segment specified by a ccCurve object and a transformation applied to this curve:

(*o_curve l_transform*)

where,

- *o_curve* is the ID of a ccCurve object.
- *l_transform* is a transformation object in the following format:

  ((*n_offsetX n_offsetY*) *n_angle*)

  where:

  - *n_angle* is the rotation angle.
  - *n_offsetX* and *n_offsetY* determine the offsets of the X-axis and Y-axis in this format (0,0).

o_curve

ID of a ccCurve object.

© 2020 All Rights Reserved.

l_transform

A transformation object in the following format:

```
((n_offsetX n_offsetY) n_angle)
```

where:

- *n_angle* is the rotation angle
- *n_offsetX* and *n_offsetY* determine the offsets or the X-axis and Y-axis in this format `(0,0)`.

l_lengthWidths

```
((n_length n_width)...)
```

List of spline points specifying the widths of the connector at the specified arc lengths, measured along the centerline of the connector from its first point.

If no width is specified at the arc length corresponding to one of the following:

- First point of the connector, then the width there is set to the width at the spline point with the largest arc length that is smaller than the minimum arc length.
- Last point of the connector, then the width there is set to the width at the spline point with the smallest arc length that is larger than the maximum arc length.

If no such spline point exists, then the width at:

- First point of the connector is set to the width at the spline point with the smallest arc length that is larger than the minimum arc length.
- Last point of the connector is set to the width at the spline point with the largest arc length that is smaller than the maximum arc length.

n_clamp1

The value of the first derivative of the spline function at the first connector point.

The default is `nil`, which means that the spline function has zero second derivative at the first connector point.

n_clamp2

The value of the first derivative of the spline function at the last connector point.

The default is `nil`, which means that the spline function has zero second derivative at the last connector point.

© 2020 All Rights Reserved.

Value Returned

l_segmentWidths List of polyCurve segments including width expressions that can be passed as an argument to the ccCreatePath function.

`((o_curve l_transform t_width)...)`

where,

- *o_curve* is the ID of a ccCurve object.
- *l_transform* is a transformation object in the following format:

  `((n_offsetX n_offsetY) n_angle)`

  where:

  - *n_angle* is the rotation angle.
  - *n_offsetX* and *n_offsetY* determine the offsets of the X-axis and Y-axis in this format `(0,0)`.
- *t_width* is the width expression as a function of the arc length parameter.

Example

```
connector=ccWaypointConnector(nil nil '((0 0) (20 0) (30 10)) 3 1.0)->connector
cc@0x865e2023
path=ccWideConnector(connector '((10 2) (30 3)) nil -0.1)
((cc@0xc56b5a5
    ((0.0 0.0) 0.0) "((2+(0.01935412284*t))+((-0.0001935412284)*(t^3)))"
    )
    (cc@0x5553f7ea
    ((0.0 0.0) 0.0) "(((1.294376315+(0.2310412284*t))+((-
0.02116871056)*(t^2)))+(0.0005120824568*(t^3)))"
    )
    (cc@0x38b6da2a
    ((18.75736 3.0) -90.0) "(((1.55965145+((-
0.02258597774)*t))+(0.007647233365*(t^2)))+(0.0005120824568*(t^3)))"
    )
    (cc@0x2d58b6d8
    ((0.0 0.0) 0.0) "(((4.81304763+((-
0.5682818167)*t))+(0.03108374613*(t^2)))+((-0.0005120824568)*(t^3)))"
    )
    (cc@0x4543edc5
```


© 2020 All Rights Reserved.

```
    ((0.0 0.0) 0.0) "(((3+(0.1*t))+(0.03576862324*(t^2)))+((-
0.01512250106)*(t^3)))"
    )
)
```


© 2020 All Rights Reserved.

Photonics Flow SKILL functions

This section covers the following photonics flow SKILL functions:

- phoAddWaveguidePorts
- phoCompose
- phoComputeModeProps
- phoHiCompose
- phoCreatePort
- phoDrawCurveDeltaConnectorSymbol
- phoGetConnectorAnchors
- phoGenWaveguide
- phoHiDecompose
- phoHiEditCompositeWG
- phoIsWaveguide
- phoTechDeleteModePropDataTable
- phoTechGetModePropDataTableParamValues
- phoTechHasModeProp
- phoTechHasModePropDataTable
- phoTechIsModePropLocalOnly
- phoTechLoadModePropDataTable

© 2020 All Rights Reserved.

phoAddWaveguidePorts

```
phoAddWaveguidePorts(
     ccPath
     t_startPortName
     t_startPortDirection
     t_endPortName
     t_endPortDirection
     [ ?n_portPinWidth n_portPinWidth ]
     )
     => l_port / nil
```

Description

(ICADVM18.1 Photonics Only) Adds circular optical ports at both ends of a ccPath. The ports have the same layer-purpose pair as the ccPath. For both start and end ports, the net, terminal, and pin names use the specified port name. Terminal direction is taken from the corresponding `portDirection` argument.

The optional `portPinWidth` (in user units) allows an over-ride of the port figure width. This is used for tapered waveguides to avoid very large shapes that might extend outside of the boundary of the original path port figure shapes.

© 2020 All Rights Reserved.

Arguments

o_path	ID of a ccPath object.
t_startPortName	The name of the start port. For example, `"opt0"`.
t_startPortDirection	The direction of the start port. The default is `"inputOutput"`.
t_endPortName	The name of the end port. For example, `"opt1"`. The default is `"end"`.
t_endPortDirection	The direction of the end port. The default is `"inputOutput"`.
?n_portPinWidth n*_portPinWidth*	Attribute value of the width for the specified photonics pinFig in user units.

Value Returned

l_port	List with descriptions for the two ports that are created. **Note:** The position information is relative to the origin of the cellview in which the port exists.
`nil`	The ports are not created, or an error occurs.

Example

```
cv = geGetEditCellView()
ccCurve = ccCreateCurve( "t" "3*sin(t/3.0)" 0:10)

ccPath = ccCreatePath(cv list("waveguide" "drawing") list(ccCurve) ?defaultWidth
0.5)
ccGenFigs(ccPath)
```

© 2020 All Rights Reserved.

```
phoAddWaveguidePorts(ccPath "start" "inputOutput" "end" "inputOutput”)
(
    (nil xy
    (0.0 0.0) name "start" direction "inputOutput" pinFigId db:0x211fbe1b facet
    (nil width 0.5 angle 135.0 radius 0.0)
    )
    (nil xy
    (10.0 -0.572) name "end" direction "inputOutput" pinFigId db:0x211fbe1c facet
        (nil width 0.5 angle 225.53 radius -43.31903
        )
    )
)
```

© 2020 All Rights Reserved.

phoCompose

```
phoCompose(
     l_dbInstID
     l_libCellView
     )
     => t_dbInstId / nil
```

Description

(ICADVM20.1 Photonics Only) Composes the specified list of waveguides into a composite of the specified master.

Arguments

l_dbInstID	*t_insts* | *l_insts* A single database instance or a list of database instances that are turned into a composite waveguide. These instances must be valid waveguide building blocks that can be placed inside a composite waveguide.
l_libCellView	A list representing the library cellview master of the composite waveguide to be created.

Value Returned

t_dbInstId	Database instance ID of the composite waveguide created.
nil	The composite waveguide was not created because an error occurred or the specified master is not a composite waveguide.

Example

Create a composite waveguide using the specified waveguide master.

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView() ; This should be a layout cellview
    wg1 = dbFindAnyInstByName(cellView "Opt0")
    wg2 = dbFindAnyInstByName(cellView "Opt1")
    ; wg1 and wg2 are valid waveguide building blocks
```


© 2020 All Rights Reserved.

```
    composite = phoCompose(list(wg1 wg2) list("gopdk" "waveguideComposite"
"layout"))
  ))
```


© 2020 All Rights Reserved.

phoComputeModeProps

```
phoComputeModeProps(
     o_occPath
     n_waveLength
     n_temperature
     t_mode
     [ valueFlags l_valueFlags ]
     [ alpha n_alpha ]
     )
     => l_values / nil
```

Description

(ICADVM20.1 Photonics Only) Computes the path length and average values for the selected mode properties along the center line of the given path

Arguments

o_occPath	A curvy path for modeProp computation.
n_waveLength	Input wavelength parameter. This value should be present in the modeProp table.
n_temperature	Input temperature parameter. This value should be present in the modeProp table.
t_mode	Input modeProp parameters. This value should be present in the modeProp table.
l_valueFlags	List of modeProp values to compute. The default is `nil`, which means all modeProp values are computed.
alpha n_alpha	The required computation precision. The default is `0.1`, which is the maximum allowed value. Specifying a value greater than `0.1` results in an error.

© 2020 All Rights Reserved.

Value Returned

l_values	`('length` *`n_length`* `['neff` *`n_neff`*`] ['ng` *`n_ng`*`] ['loss` *`n_loss`*`] ['D` *`n_D`*`])` List of modeProp parameter and value pairs starting with `'length` *`n_length`* and followed by the parameter name along with value pairs.
`nil`	The operation fails.

Example

```
phoComputeModeProps(p 1.5475e-6 323 "TE" '(neff ng))
```

■ `> (nil length 668.0854 neff 1.67572 ng 2.052839)`

© 2020 All Rights Reserved.

phoHiCompose

```
phoHiCompose(
     l_dbInstID
     [ ?compositeMaster l_compositeMaster ]
     )
     => d_InstID / nil
```

Description

(ICADVM20.1 Photonics Only) Uses a list of waveguide building blocks that are abutted into a single chain and creates a composite waveguide using these building blocks. The instances used for building the composite waveguide could be bound or unbound at the time of waveguide creation, but all the source waveguide instances must originally be bound to the same schematic instance. The composite waveguide is created at the same location as the source waveguide instances and is bound to the same schematic instance as the source instances.

Arguments

l_dbInstID	List of database instance IDs abutted together to be used as source for creating the composite waveguide. If a single instance is selected as the source, the SKILL function creates a waveguide using only the selected instance.
?compositeMaster *l_compositeMaster*	The library cell view that specifies the master of the composite waveguide to be created.

Value Returned

d_InstID	Database instance ID of the composite waveguide created.
nil	The composite waveguide was not created.

Example

```
when(win = hiGetCurrentWindow()
  when(lcv = geGetEditCellView(win)
    i0 = dbFindAnyInstByName(lcv "I0")
    i1 = dbFindAnyInstByName(lcv "I1")
    i2 = dbFindAnyInstByName(lcv "I2")
```


© 2020 All Rights Reserved.

```
    composite = phoHiCompose(list(i0 i1 i2))
  )
)
```

Composes three abutted instances into a composite waveguide instance.

Related Topics

- Composite Waveguide Editor (ICADVM20.1 Only)
- Editing Composite Waveguides (ICADVM20.1 Only

© 2020 All Rights Reserved.

phoCreatePort

```
phoCreatePort(
     d_cellViewId
     l_lpp
     t_x
     t_y
     t_netName
     t_termName
     t_pinName
     n_width
     n_angle
     n_radius
     [ n_portWidth ]
     [ n_portHeight ]
     => d_portShape / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a physical connection point (port) for an optical net on layer type `waveguide`. The area of the optical port is used to trigger auto-abutment of the instances to ensure that the ports with matching attributes (layer, angle, or width) are positioned to avoid any cuts between the facets.

Note: For auto-abutment, the width attribute of the optical port does not have to match the physical width and height of the shape (ellipse) used to draw the port.

Arguments

d_cellViewId	Database ID of a cellview.
l_lpp	Layer-purpose pair for used to create the port. Valid value for layer is `waveguide`. Valid value for purpose is `drawing`.
t_x	X-coordinate for the center of the port. **Note:** A port is drawn as a circle.
t_y	Y-coordinate for the center of the port.
t_netName	Name of the optical net. Net is created if it does not exist. If the net already exists, it should have signal type as optical.
t_termName	Name of the terminal.

© 2020 All Rights Reserved.

t_pinName	Name of the pin. If a pin name is not specified, it is created.
n_width	Width of the optical facet.
n_angle	Angle of the optical facet.
n_radius	Radius of the optical facet.
n_portWidth	Width to draw the physical port (width of the ellipse). **Note:** If no default width is specified, `phoCreatePort` uses the `minWidth` for the specified layer-purpose pair. If the `minWidth` for the layer-purpose pair is not defined in the technology database, a port of minimum size (2DBU) is created.
n_portHeight	Height to draw the physical port (height of the ellipse). **Note:** If no default width is specified, `phoCreatePort` uses the `minWidth` for the specified layer-purpose pair. If the `minWidth` for the layer-purpose pair is not defined in the technology database, a port of minimum size (2DBU) is created.

Value Returned

d_portShape	Database ID of the port that is created.
`nil`	Port was not created.

Example

```
phoCreatePort(cv list("waveguide" "drawing") 10.0 5.0 "inputOutput" "Opt1" "Opt1"
nil 5.0 90.0 0.0)
```


© 2020 All Rights Reserved.

```
=> db:0x215db89c
```

© 2020 All Rights Reserved.

phoDrawCurveDeltaConnectorSymbol

```
phoDrawCurveDeltaConnectorSymbol(
     d_cellViewID
     f_mag
     [ ?startPortName t_startPortName ]
     [ ?startPortDirection t_startPortDirection ]
     [ ?endPortName t_endPortName ]
     [ ?endPortDirection t_endPortDirection ]
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Generates the symbol artwork for a photonics Curve Delta connector waveguide. The start port is defined as the side of the waveguide (in the layout), where the mathematical equation describing the centerline starts (usually starts at `0`) and the end port is where it finishes. This concept is used to enable proper stitching in the composite waveguide.

Arguments

d_cellViewID
: Database ID of the cellview (usually a Pcell) where the artwork is created.

f_mag
: The level of magnification.

?startPortName *t_startPortName*
: The name of the start port. For example, `"opt0"`.

?startPortDirection *t_startPortDirection*
: The direction of the start port. The default is `"inputOutput"`.

?endPortName *t_endPortName*
: The name of the end port. For example, `"opt1"`. The default is `end`.

?endPortDirection *t_endPortDirection*
: The direction of the end port. The default is `"inputOutput"`.

© 2020 All Rights Reserved.

Value Returned

`t`	The symbol artwork is created successfully.
`nil`	An error occurred and the artwork was not created.

Example

```
pcDefinePCell(
    list( ddGetObj("gopdk") "wgCurveConnector" "symbol" "schematicSymbol")
        (
        (mag float 1.0)
    )

    let( ()
    phoDrawCurveDeltaConnectorSymbol( pcCellView mag
        ?startPortName "Opt1"
        ?endPortName "Opt2"
    )
)
)
```


© 2020 All Rights Reserved.

phoGetConnectorAnchors

```
phoGetConnectorAnchors(
     dbInstID
     )
     => list( l_anchorStartDescription l_anchorEndDescription))
```

Description

(ICADVM20.1 Photonics Only) Reads the supported optical nets connected to the instance terminals of an instance to return the information about the start and end anchor points for an associated connector waveguide. A connector waveguide has exactly two directed optical ports.

Arguments

dbInstID	Database instance IDs of the specified instance connector.

Value Returned

list(l_anchorStartDescription l_anchorEndDescription))

Anchor start and end point for the specified connector waveguide returned in the following list format:

```
list( anchorLPP
              anchorXY
              angle
              width
              radius
             )
```

Example

```
instId = car(geGetSelSet())
```

Gets a database pointer to a connector waveguide instance selected in the layout canvas.

```
phoGetConnectorAnchors(instId)
((("waveguide" "drawing")
    (-44.534 12.882) 0.0 0.5 nil
    )
```


© 2020 All Rights Reserved.

```
    (("waveguide" "drawing")
    (95.568 7.032) 0.0 0.5 nil
    )
)
```

Gets the anchor information for the selected connector instance.

© 2020 All Rights Reserved.

phoGenWaveguide

```
phoGenWaveguide(
     o_path
     [ ?excludeLayers l_excludeLayers ]
     ) => l_paths / nil
```

Description

(ICADVM20.1 Photonics Only) Generates a list of derived ccShapes for the specified ccPath, using the waveguide definition of the layer-purpose pair of the specified ccPath. The derived shapes on the layers listed in `excludeLayers` are skipped during the generation. For derived shapes of type `offset` with inner value, the inner edge of the generated derived shape is coincident with the centerline of the path.

Note: Currently, this function cannot generate derived shapes if at the same time path has variable width and offset or enclosure values in the waveguide definition are not constant.

Arguments

o_path	ID of a ccPath.
l_excludeLayers	List of waveguide layer names to exclude. `list(s_layer...)`

Value Returned

l_shapes	List of the generated ccShape IDs. `list(o_shape...)`
`nil`	An error occurred.

Example

Create a ccPath

```
arc = ccCreatePath(
    cv "waveguide"
    list(list(ccCreateCurve("cos(t)" "sin(t)" -3.14159265:3.14159265)
              list(0:0 0)
              "0.01")))
=> cc@0x209363e0
```


© 2020 All Rights Reserved.

Generate a list of ccShapes using the waveguide definition of the `waveguide` layer. Additionally, exclude the `metal` layer while generating the list of ccPaths.

```
phoGenWaveguide(arc ?excludeLayers '("metal"))
=> (cc@0x208b2bc0 cc@0x25c99690)
```

Related Topics

waveguideTemplates

© 2020 All Rights Reserved.

phoHiDecompose

```
phoHiDecompose(
     l_dbCompWaveInstID
     )
     => l_dbInstID / nil
```

Description

(ICADVM20.1 Photonics Only) Uses a list of composite waveguides or a single composite waveguide and flattens the waveguide to bring the instances inside to the current level. The newly separated waveguide building blocks are bound to the original schematic composite in a complex binding. The instances of the waveguide building blocks are abutted together into a chain and the chain has the same input and output connectivity as the original composite waveguide.

Arguments

l_dbCompWaveInstID	List of database instance IDs of the composite waveguides abutted together to form the flattened composite waveguide. If a single composite waveguide instance is selected as the source, the newly created waveguide instance uses only the selected waveguide instance.

Value Returned

l_dbInstID	List of database instance IDs created if a single composite waveguide instance was flattened. List of lists of database instance IDs created if multiple composite waveguide instances are flattened.
`nil`	No composite waveguide instance was flattened.

Example

```
when(win = hiGetCurrentWindow()
  when(lcv = geGetEditCellView(win)
    i0 = dbFindAnyInstByName(lcv "I0")
    insts = phoHiDecompose(i0)
```

Edits the parameters of the first instance and composes back into a composite waveguide.

© 2020 All Rights Reserved.

```
    car(insts)~>w = "2u"
    phoHiCompose(insts)
  )
)
```

Related Topics

Composite Waveguide Editor (ICADVM20.1 Only)

Editing Composite Waveguides (ICADVM20.1 Only

© 2020 All Rights Reserved.

phoHiEditCompositeWG

```
phoHiEditCompositeWG(
     )
```

Description

(ICADVM20.1 Photonics Only) Opens the Composite Waveguide Editor for the currently selected instance. The editor is displayed only when a single composite waveguide instance is selected. The editor, once open, needs to be manually closed.

Arguments

None

Value Returned

None

Example

```
phoHiEditCompositeWG()
```

Related Topics

Composite Waveguide Editor (ICADVM20.1 Only)

Editing Composite Waveguides (ICADVM20.1 Only)

© 2020 All Rights Reserved.

phoIsWaveguide

```
phoIsWaveguide(
     [ ?masterID d_masterID ]
     [ ?lcv l_lcv ]
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified master cellview or the library cellview is of type *waveguide*.

Arguments

`?masterID` *`d_masterID`*

Database cellview ID of the instance master that you want to check.

Note: If both `masterID` and `lcv` are specified, the SKILL function uses the `masterID` for the check.

`?lcv` *`l_lcv`*

List containing the names of the library, cell, and view that you want to check.

Note: If both `masterID` and `lcv` are specified, the SKILL function uses the `masterID` for the check.

Value Returned

`t`	The specified master cellview, or the specfied library cellview if the master cellview is not specified, was found to be of type *waveguide*.
`nil`	The specified master cellview or the specified library cellview is not a waveguide.

Related Topics

Generating Optical Chains (ICADVM20.1 Only)

© 2020 All Rights Reserved.

phoTechDeleteModePropDataTable

```
phoTechDeleteModePropDataTable(
     d_techDB
     t_waveguideDefName
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Deletes the modeProp table for the specified waveguideDef from the given technology database.

Arguments

d_techDB	Database ID of the technology database in which the modeProp table is searched.
t_waveguideDefName	Name of the waveguideDef.

Value Returned

`t`	The modeProp table was found and deleted.
`nil`	The modeProp table was not found.

Example

```
tech_wgd~>refs = list(tech_modeprop)
> (db:0x028ff592)
phoTechHasModePropDataTable(tech_wgd "SiN Waveguide")
> t
phoTechDeleteModePropDataTable(tech_wgd "SiN Waveguide") ;; non-local delete
> nil
phoTechDeleteModePropDataTable(tech_modeprop "SiN Waveguide")
> t
```


© 2020 All Rights Reserved.

phoTechGetModePropDataTableParamValues

```
phoTechGetModePropDataTableParamValues(
     d_techDB
     t_waveguideDefName
     S_param
     )
     => l_values / nil
```

Description

(ICADVM20.1 Photonics Only) Returns the list of values specified in the modeProp table for the given parameter.

Arguments

d_techDB	ID of the technology database storing the modeProp table.
t_waveguideDefName	Name of the waveguideDef.
S_param	Name of the parameter. Valid values are `'wavelength`, `'width`, `'curvature`, `'temperature` and `'mode`, or the value of corresponding string.

Value Returned

l_values	(*n_value...*) | (*t_value...*) A list of numeric values (*n_value*) that are valid for all parameters other than `'mode` or a list of strings (*t_value*) that are valid for `'mode`.
nil	The modeProp table is not found.

Example

```
phoTechGetModePropDataTableParamValues(tech "SiN Waveguide" 'width)
=> (0.0 3e-05 6e-05)
phoTechGetModePropDataTableParamValues(tech "SiN Waveguide" 'mode)
=> ("TE" "TM")
```


© 2020 All Rights Reserved.

phoTechHasModeProp

```
phoTechHasModeProp(
     d_techDB
     t_waveguideDefName
     s_param
     g_value
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified parameter value is present in the modeProp table for the specified waveguide in the specified technology database.

Note: The search recurses into the referenced databases. The database that stores the waveguideDef and the database that stores the modeProp table are not necessarily the same. Therefore, the given waveguideDef does not need to exist in the technology database.

© 2020 All Rights Reserved.

Arguments

d_techDB ID of the technology database in which the search is performed.

t_waveguideDefName

Name of the waveguideDef.

s_param

modeProp parameter name for which the technology database is searched.

g_value

modeProp parameter value for which the technology database is searched.

Value Returned

`t` The specified parameter value was found in the modeProp table for the specified waveguide.

`nil` The specified parameter value was not found.

Example

```
tech_wgd~>refs = list(tech_modeprop)
=> (db:0x028ff592)
phoTechHasModePropDataTable(tech_modeprop "SiN Waveguide")
=> t
phoTechHasModePropDataTable(tech_wgd "SiN Waveguide")
=> t
```


© 2020 All Rights Reserved.

phoTechHasModePropDataTable

```
phoTechHasModePropDataTable(
     d_techDB
     t_waveguideDefName
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Checks whether the given technology database contains a modeProp table for the given waveguideDef.

Note: The search recurses into referenced databases. The database that stores the waveguideDef and the database that stores the modeProp table are not necessarily the same. Therefore, the given waveguideDef does not need to exist in technology database.

Arguments

d_techDB	ID of the technology database in which search is performed.
t_waveguideDefName	Name of the waveguideDef.

Value Returned

t	The modeProp table for the given waveguideDef name was found.
nil	The modeProp table for the given waveguideDef name was not found.

Example

```
tech_wgd~>refs = list(tech_modeprop)
=> (db:0x028ff592)
phoTechHasModePropDataTable(tech_modeprop "SiN Waveguide")
=> t
phoTechHasModePropDataTable(tech_wgd "SiN Waveguide")
=> t
```


© 2020 All Rights Reserved.

phoTechIsModePropLocalOnly

```
phoTechIsModePropLocalOnly(
     d_tech
     t_waveguideDefName
     )
     => t / nil
```

Description

(ICADVM18.1 Photonics Only) Retrieves the `localOnly` flag of the modeProp table for the specified waveguideDef. The table is searched only in the local technology database.

Arguments

d_techDB	ID of the technology database in which search is performed.
t_waveguideDefName	Name of the waveguideDef.

Value Returned

`t`	The `localOnly` flag is set.
`nil`	The `localOnly` flag is not set, the modeProp data is missing, or an error occurred.

Example

```
; "hier_top" references "hier_modeProp"
hier_top~>refs~>name
> ("hier_modeProp")

; load oneline.txt as a local only modeProp table
phoTechLoadModePropDataTable(hier_top "oneline.txt"
                             ?waveguideDefName "SiN Waveguide"
                             ?overwrite t
                             ?localOnly t)
> t

phoTechIsModePropLocalOnly(hier_top "SiN Waveguide")
> t
```


© 2020 All Rights Reserved.

```
; hier_top only has data for mode "TE"
phoTechGetModePropDataTableParamValues(hier_top "SiN Waveguide" 'mode)
> ("TE")

; hier_modeProp has data for modes "TE" and "TM"
phoTechGetModePropDataTableParamValues(hier_modeProp "SiN Waveguide" 'mode)
> ("TE" "TM")



; re-merge hier_modeProp table by setting localOnly to nil
phoTechLoadModePropDataTable(hier_top "oneline.txt"
                             ?waveguideDefName "SiN Waveguide"
                             ?overwrite t
                             ?localOnly nil)
> t

phoTechIsModePropLocalOnly(hier_top "SiN Waveguide")
> nil

; hier_top still has data only for the "TE" mode, ; but with localOnly=nil, all
getter APIs merge in the "TM" data ; from hier_modeProp.
phoTechGetModePropDataTableParamValues(hier_top "SiN Waveguide" 'mode)
> ("TE" "TM")
```


© 2020 All Rights Reserved.

phoTechLoadModePropDataTable

```
phoTechLoadModePropDataTable(
     d_techID
     t_path(
     [ ?waveguideDefName t_waveguideDefName ]
     [ ?localOnly g_localOnly ]
     [ ?overwrite g_overwrite ]
     )
     => t / nil
```

Description

(ICADVM20.1 Photonics Only) Loads the modeProp table from the specified data file for the given waveguide definition and stores it in the library `data.dm` file. If no waveguideDef is specified, the modeProp table is loaded for all the waveguide definitions encountered in the data file.

Arguments

d_techID ID of the technology database file in which the specified waveguideDef is stored.

t_path Path to the data file.

?localOnly *g_localOnly*

A boolean parameter indicating how modeProp table is retrieved.

- When set to `t`, the contents of the specified modeProp table is retrieved only from the local technology database. This is the default.
- When set to `nil`, the table is assembled from the modeProp tables in the technology database hierarchy.

The value of this parameter is saved in the technology database so that every individual modeProp table has its `localOnly` flag. Once set, this flag permanently affects the modeProp table contents exposed to the modeProp APIs.

Note: This default value `t` is in effect for the modeProp tables created by the older versions of this API.

© 2020 All Rights Reserved.

`?overwrite` *`g_overwrite`*

A boolean parameter indicating whether content of the data file overwrites the existing modeProp table.

- When set to `t`, the content of the data file completely overwrites the existing modeProp table. This is the default.
- When set to `nil`, the content of the data file is merged into the existing modeProp table.

`t_waveguideDefName` Name of the waveguide definition.

Value Returned

`t` At least one waveguideDef table was loaded successfully.

For best practices on successfully loading a waveguideDef table, see Additional Information.

`nil` No `waveguideDef` table was loaded successfully.

Example

```
(system "cat data.txt")
> waveguideDefName "SiN Waveguide"
> mode 99 = "TE"
> mode 100 = "TM"
> 1 0 0 0  99 0 0 0 0
> 1 0 0 0 100 0 0 0 0
>

phoTechLoadModePropDataTable(tech "data.txt" ?waveguideDefName "SiN Waveguide")
> t
```

Additional Information

To successfully load a waveguideDef table, here are a few related prerequisites that should be fulfilled:

- **Correct syntax**

© 2020 All Rights Reserved.

The waveguideDef table should be loaded using the correct syntax, which begins with the waveguideDefName line, optionally followed by mode mappings that are followed by table lines of nine numeric values.

- **Completeness of the mode name mapping**

  If present, the mode-name mappings in the waveguideDef table should be complete. All IDs in the table should be mapped to appropriate waveguideDef mode names.

- **Completeness of the table data**

  The waveguideDef table should be complete. The table should list all possible combinations otherwise the function will terminate.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

18

Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions

This chapter provides the syntax, descriptions, and examples for the SKILL functions that you can use in conjunction with the Pegasus Interactive and the Virtuoso Integrated Physical Verification System (VIPVS) editing feature.

pgssDRC

```
pgssDRC(
     d_cellviewID
     t_rulesFile
     t_layerMap
     t_objectMap
     l_options
     g_function
     )
     => t / nil
```

Description

Launches Pegasus DRC and processes the generated shapes in accordance with the specified function. The `pgssDRC` SKILL function needs Pegasus license.

Arguments

d_cellviewID	Specifies the cellview to process. This must be of type maskLayout. It cannot be a Pcell variant.
t_rulesFile	Specifies the path to the Pegasus rules file.

© 2020 All Rights Reserved.

t_layerMap	Specifies the path to the layer map file. If no layer map file is specified, that is, if it is specified as an empty string ("""), the layer map file in the cell's technology library is used.
t_objectMap	Specifies the path to the object map file. If no object map file is specified, that is, if it is specified as an empty string ("""), the object map file from the cell's technology library is used. To avoid using the default object map file, set the value to "/dev/null".

l_options	Specifies a disembodied property list of control option. Options specified here override what is set in the rules file. Supported values are:
	`control->area = list( x1:y1 x2:y2 )`
	`control->enableColoring = t/nil`
	`control->abortOnLayoutError = t/nil`
	`control->excludeCells "lib1/cell1/view1 ... "`
	`control->excludeCells "opclib/*/layout ... "`
	`control->ruleSet = list( "rule1" "rule2" ... )`
	Wildcard characters are allowed in `excludeCells` as shown in the fifth example above. This will exclude every cell from the `opclib` library.
	Rules listed in `ruleSet` must exist in the rule deck.

g_function	Specifies a function object which is applied to each shape one at a time. It can be a procedure name, like `'proc` (note the leading quote), or a lambda value.
	The procedure must take one argument which is a disembodied property list with the following members:
	`shape->edges = list( list(x1:y1 x2:y2) ... )` if it is an edge
	`shape->points = list( x1:y1 x2:y2 ... )` if it is a polygon
	`shape->caption`
	`shape->checkName`

© 2020 All Rights Reserved.

	Returned shapes are either a list of edges or a list of points based on how the rules are written. A rule does not require a caption, so you need to make sure it exists before using it. If the function returns the symbol `'stop`, Pegasus exits and no more shapes are generated. Any other value means continue processing.

Value Returned

`t`	No operational errors occur and Pegasus DRC was launched successfully.
`nil`	Pegasus DRC failed to launch due to some error. For example, it might be that the rules failed to compile.

Examples

Run Pegasus and dump the shapes to the CIW.

```
pgssDRC( cellView "rules.pvl" "rule_layer.map" "" nil lambda( (shape)
println(shape) t) )
```


© 2020 All Rights Reserved.

pvsApplyDRC

```
pvsApplyDRC
     d_cellview
     t_rulesFile
     t_layerMap
     t_objectMap
     l_options
     g_function
     )
     => t / nil
```

Description

Launches Virtuoso IPVS and process generated shapes.

Arguments

d_cellview	Specifies the cellview to process. This must be a mask layout cell. It cannot be a Pcell variant.
t_rulesFile	Specifies the path to the PVS rules file.
t_layerMap	Specifies the path to the layerMap file. If no layerMap file is specified, layerMap in the cell's technology library is used.
t_objectMap	Specifies the path to the objectMap file. If no objectMap file is specified, objectMap from the cell's technology library is used. To avoid using the default objectMap, set the value to `"/dev/null"`.

© 2020 All Rights Reserved.

l_options

Specifies a disembodied property list of control option. Options specified here override what is set in the rules file. Supported values are:

```
control->area = list( x1:y1 x2:y2 )
```

```
control->enableColoring = t/nil
```

```
control->abortOnLayoutError = t/nil
```

```
control->excludeCells "lib1/cell1/view1 ... "
```

```
control->ruleSet = list( "rule1" "rule2" ... )
```

Wildcard characters are allowed in `excludeCells` such as `"opclib/*/layout"` which will exclude every cell from `opclib` library.

Rules listed in `ruleSet` must exist in the rule deck.

g_function

Specifies a function object which is applied to each shape one at a time. It can be a procedure name, like 'proc (note the leading quote), or a lambda value.

The procedure must take one argument which is a disembodied property list with the following members:

```
shape->edges = list( list(x1:y1 x2:y2) ... )
if it is an edge
```

```
shape->points = list( x1:y1 x2:y2 ... ) if it
is a polygon
```

```
shape->caption
```

```
shape->checkName
```

Returned shapes are either a list of edges or a list of points based on how the rules are written. A shape will never have both. The caption value may not be present (a rule does not require a caption). So you need to make sure it exist before using it.

If the function returns the symbol 'stop, PVS will exit and no more shapes will be generated. Any other value means continue processing.

© 2020 All Rights Reserved.

Value Returned

`t`	No operational errors occur and Virtuoso IPVS launched successfully.
`nil`	Operation problem occurs, such as the rules fail to compile.

Examples

Run PVS and dump the shapes to the CIW.

```
pvsApplyDRC( cv "rules.pvl" "" "" nil lambda( (shape) println(shape) t)
```


© 2020 All Rights Reserved.

19

Attribute Retrieval and Modification Database Access Functions

This chapter includes the following topics:

- dbGet Functions
- Using the ~> Operator
- dbSet Functions
- Attribute Types
- Object Types
- Generic Attributes
  - Generic Object Attributes
  - Generic Figure Attributes
  - Generic Shape Attributes
  - Generic anyInst Attributes
  - Generic Boundary Attributes
  - Generic Blockage Attributes
  - Generic Via Attributes
  - Generic Via Header Attributes
  - Generic widthSpacingPattern Attributes
  - Generic widthSpacingPatternGroup Attributes
  - Generic relatedSnapPattern Attributes
  - Generic widthSpacingSnapPatternDef Attributes
  - Generic rowRegion Attributes (ICADVM181.1 Only)

© 2020 All Rights Reserved.

- Attributes of Cellviews
- Attributes of Instances, Instance Headers, and Mosaics
- Attributes of Instance Terminals, Pins, and Terminals
- Attributes of Bus Definitions, Nets, and Signals
- Attributes of Layer Purpose Pairs and Layer Headers
- Attributes of Groups, Group Members, and Properties
- Attributes of Objects and Figures
- Attributes of Shapes
- Technology File Attributes

dbGet Functions

You can retrieve the attributes of any valid *dbObject* by using one of the various forms of the `dbGet` function. This function is used for most attribute retrievals that do not require any other arguments. Other functions, such as `dbGetMemNetSigName`, take additional arguments to retrieve the desired attributes.

dbGetq, dbGet

```
dbGetq( d_object St_AttrOrPropName )
          => g_value

     dbGet( d_object St_AttrOrPropName )
          => g_value
```

Description

Retrieves the value of an attribute or a property associated with a database object.

For `dbGetq` and the "~>" operator, the attribute or property name *`St_AttrPropName`* can be specified as either a string or a symbol. For `dbGet`, this argument can also be specified as a variable because `dbGet` evaluates its second argument.

The value of an attribute can be a string, an integer, a floating-point number, a symbol, a Boolean value, a relationship, another database object, or a list. The following are examples of list values.

- A single point represented by a two-element list containing x and y coordinates.

© 2020 All Rights Reserved.

- A list containing two lists. For example two points, each a list of an x and y, or the bBox of an object.
- A variable sized list of points such as the points of a path or polygon.
- A two element list containing layerName and purposeName (the “lpp” of a particular shape).
- The transformation of an instance defined by three elements. The first, a “point” (a list of two, an x and y), the second, “orientation” (a string, e.g. “R0”) and the third, a float (the magnification, e.g. 1.0).
- A one-to-many relationship or a one-to-one relationship. The value of a relationship is a database object in the case of a one-to-one or many-to-one relationship, or a list of database objects in the case of a one-to-many relationship such as all the terminals in a cellview.

The value of a user-defined property can be a string, an integer, a floating-point number, a file name string, a hierProp, an IL expression string, an IL list, an NLP expression string, a time value, or a Boolean value. Time-valued properties are converted to strings. Boolean values appear as strings with the value `TRUE` or `FALSE` rather than `t` or `nil`. In this way, you can distinguish between existence and `nil` (FALSE) or undefined.

`dbGet`, `dbGetq`, and the ~> operator can be used with SKILL control statements to sequence over each value of an object’s attribute that contains a list, as illustrated below:

```
foreach( shape cellview~>shapes
         ; process the shape here...
)
```


© 2020 All Rights Reserved.

Arguments

d_object	*dbObject* of the object to be queried.
St_AttrOrPropName	The attribute or property name to be retrieved. It can be specified as a SKILL symbol or a string. For dbGet, this can also be a SKILL variable containing a symbol or a string that represents the attribute or property name.

Value Returned

g_value	Value of the attribute or property *St_AttrOrPropName* of object *d_Object*.

Examples

Retrieves the value of the *shapes* attribute associated with a cellview object *c* by any of the following:

```
dbGetq(c shapes)          ; Does not evaluate second argument
c~>shapes                 ; Equivalent to above
c~>"shapes"               ; Equivalent to above
dbGet(c 'shapes)          ; Evaluates both its arguments
str = 'shapes             ; Assigns attribute name
dbGet(c str)               ; "str" is evaluated to shapes
```


© 2020 All Rights Reserved.

Using the ~> Operator

Retrieving the List of Attribute Names for a *dbObject*

The ~> ? operation returns a list of all available attributes for a *dbObject*. The list of attributes is based on the database object's type.

d_object ~> ? => l_attributeNames

The sample below finds the names of all attributes of a property:

```
<1> cv = dbOpenCellViewByType(nil "inv" "symbol")
; open a cellview
db:2066752
<2> prop = car(cv~>prop)     ; ; get the first property
db:2068776
<3> prop~>?              ;; fetch the attribute names
(cellview objType prop enum name object range value valueType)
```

Retrieving the List of Attributes/Values for a *dbObject*s

The ~> ?? operation returns a list of all available attributes and their values as a disembodied property list for a *dbObject*. The list of attributes is based on the database object's type.

d_object ~> ?? => l_attributeValuePairs

In the previous example, the property's attributes and values could be asked by the following:

```
<4> prop~>??
(db:2068776 cellview db:2066752 objType "prop"
prop nil enum nil name "lastSchematicExtraction"
object db:2066788 range nil
value "May 20 19:41:08 1991" valueType "time")
```

The following is an example of retrieving attributes of a rowHeader from the cellview.

```
<1> cv = geGetEditCellView()
<2> cv~> rowHeaders~>??
((db:0x03b02412 cellView db:0x03b0001d objType "rowHeader"
prop nil siteDef db:0x03b02514 siteDefName
"CORE" siteDefWidth 0.51 siteDefHeight 4.59
bBox
((0.0 0.0)
(8.67 4.59)
) rows
(db:0x03b01912)
)
)
```

The following is an example of retrieving attributes for a pathSeg.

© 2020 All Rights Reserved.

```
<1> pathSeg = geGetSelectedSet()
(db:0x03b01a1f)
<2> pathSeg~>??
((db:0x03b01a1f cellView db:0x03b0001d objType "pathSeg"
prop nil bBox
((-9.435 0.665)
(-5.465 0.935)
) children
nil groupMembers nil isAnyInst nil
isShape t matchPoints nil net
nil parent nil pin nil
purpose "drawing" textDisplays nil assocTextDisplays
nil markers nil figGroup nil
layerName "METAL1" layerNum 3 lpp
("METAL1" "drawing") connRoutes nil width 0.27
beginExt 0.0 endExt 0.0 beginPt
(-5.6 0.8) endPt
(-9.3 0.8) boundary
((-9.435 0.665)
(-9.435 0.935)
(-5.465 0.935)
(-5.465 0.665)
)
beginStyle "extend" endStyle "extend" topology
"none" isOrthogonal t route nil
)
)
```


© 2020 All Rights Reserved.

dbSet Functions

With a SKILL program, you can set several of the attributes of valid *dbObjects* using one of the various forms of the *dbSet* function. Use this function when you do not need to use arguments. There are other functions, such as `dbCreateTerm`, that take additional arguments to completely add or modify information in the database.

dbSetq, dbSet

```
dbSetq( d_object g_value St_AttrOrPropName )
          => g_value / nil

     dbSet( d_object g_value St_AttrOrPropName )
          => g_value / nil
```

Description

Modifies the value of an attribute or a property associated with a database object.

Note: The equal sign ("=") and *`g_value`* are required for the "~>" implementation of the `dbSet` function, unlike the "=>" ("yields") symbol used previously, which documents the output of the `dbGet` function. Also, the "~>" operator does not accept the "?" or "??" operations when combined with the assignment operator ("=").

For `dbSetq` and the "~>" operator, the attribute or property name *`St_AttrPropName`* can be specified as either a string or a symbol. For `dbSet`, this argument can also be specified as a variable because `dbSet` evaluates its second argument.

If the specified *`St_AttrOrPropName`* is not an attribute name of object *`d_Object`*, it is treated as the name of a property for that object. If a property by that name does not exist, one is created.

Attaching and detaching an object to and from another object can be done by setting their relationship attributes. The types of relationship that can be set using this method are limited to one-to-one and many-to-one. The attribute value is set to *`dbObject`* to attach and `nil` to detach.

The next section of this chapter contains tables that describe the database objects and their attributes. Within the tables, attributes that can be set are labeled.

© 2020 All Rights Reserved.

Arguments

d_object	Database object.
g_value	The new value for the attribute or property *St_AttrOrPropName*.
St_AttrOrPropName	Specifies the name of the attribute or property to be set. It can be specified as a SKILL symbol or a string. For `dbSet`, this can also be a SKILL variable containing a symbol or a string that represents the attribute or property name.

Value Returned

g_value	Returns the new value if successful.
`nil`	Returns `nil` if not.

Examples

Sets the value of the *width* attribute associated with a path object *p* by any of the following:

```
dbSetq(p 2 width) ;      ; Does not evaluate third argument
p~>width = 2 ;           ; Equivalent to above
p~>"width" = 2 ;         ; Equivalent to above
dbSet(p 2 'width) ;      ; Evaluates 1st & 3rd arguments
w = 'width ;             ; Assigns attribute name
dbSet(p 2 w) ;      ; "w" is evaluated to width

; Attach figure f to net n:
dbSetq(f n net) ;        ; Does not evaluate third argument
f~>net = n ;              ; Equivalent to above

; Detach figure f from its net:
dbSet(f nil 'net) ;      ; Evaluates 1st & 3rd argument
f~>net = nil ;      ; Equivalent to above
```


© 2020 All Rights Reserved.

Attribute Types

Attributes in the tables describe what type of value the attribute has. Knowing the attribute type helps you understand what type of input to give to *dbSet* or what type of output to expect from *dbGet*.

Type	Meaning
bBox	The value is a boundary box (list of two points, lower left and upper right of a rectangular area that usually encloses the object).
integer	The value is an integer.
float	The value is a floating-point number.
string	The value is a string. For strings whose possible values are enumerated, the possibilities are listed with a description of each.
boolean	The value is either true (`t`) or false (`nil`).
objId	The value is an object identifier (*dbObject*).
point	The value is a point, that is, a list of two integers describing a coordinate pair.
list	The value is a list of some compound type (that is, a type made up of two or more of the previously mentioned types). The compound type is described in the value description.

Many of the attribute values have the type *l_objId* which means the value is a list of *objId*s. For example, in the Cellview object description, the shapes attribute is described as being of type objId. In reality, the value is a list of *objId*s. Thus, the description of the shapes attribute notes that the value is a list comprised of the primitive attribute type *objId.* Also, the attribute type is preceded by a *l_* (such as *l_objId*). This convention is used in all the tables.

Object Types

The Cadence design database supports the following object types:

© 2020 All Rights Reserved.

Table 19-1

© 2020 All Rights Reserved.

Generic Attributes

Some attributes are generic to figures and shapes in the database. Generic attributes are listed in the following tables.

Generic Object Attributes

Object attributes are those attributes common to all database objects.

Attribute	Modify	Type	Description
cellView	no	objId	The cellview that contains this object.
objType	no	string	The name of the object's type.
prop	no	l_objId	List of properties on this object.

Generic Figure Attributes

Figure attributes are common to database figures (shapes and anyInst).

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
assocTextDisplays	no	l_objId	List of text displays with this figure as their associate.
bBox	no/yes	bBox	The figure's bounding box. Only the bBox of a dot, ellipse, rectangle, or text can be modified.
children	no	l_objId	List of objects that have this object as their parent.
figGroup	no	objId	List the ID of the figGroup to which this figure belongs.
groupMembers	no	l_objId	List of group members for this object.
isAnyInst	no	boolean	Indicates whether the figure is an instance.
isShape	no	boolean	Indicates whether the figure is a shape.
markers	no	l_objId	List of markers for this object.
matchPoints	yes	boolean	Indicates whether the figure's children have points that match the figure's points.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
net	yes	objId	Net object ID if the figure is associated with a net
parent	yes	objId	The figure's parent.
pin	no	objId	Pin object if the figure is associated with a pin.
purpose	yes	string/	The figure's layer purpose.
textDisplays	no	l_objId	List of text displays owned by this figure object.

© 2020 All Rights Reserved.

Generic Shape Attributes

Attribute	Modify	Type	Description
Note: include Generic Figure Attributes.			
connRoutes	no	l_objId	List of all route start/end connections that connect to this shape.
isUnshielded	yes	boolean	Indicates whether this shape is not shielded.
shieldedNet1	yes	netId	Sets the first shielded net for this shape to the specified bit net. A shape may shield up to two nets.
shieldedNet2	yes	netId	Sets the second shielded net for this shape to the specified bit net.
layerName	yes	string	The shape's layer name.
layerNum	yes	integer	The shape's layer number.
lpp	yes	list	List consisting of the shape's layer number and purpose.

routeStatus	yes		Status of the route associated with the via.
			`"normal"`
			The normal routing status. The route can be modified manually or by an automatic tool.
			`"fixed"`
			Automatic tools cannot edit the route but the route can be manually edited.
			`"locked"`
			The route cannot be edited by automatic tools or manually.

Generic anyInst Attributes

Attribute	Modify	Type	Description
Note: include Generic Figure Attributes.			
baseName	yes	string	Base name of the instance. The *name* of an instance can contain a vector expression by which *baseName* and *numInst* are derived. *numInst* can also be defined at creation.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
cellName	no	string	Cell name of the master for this instance.
instHeader	no	objId	Instance header that belongs to this instance.
instTerms	no	l_objId	List of instance terminals that belong to this instance.
libName	no	string	Name of the library that contains the master for this instance.
master	yes	objId	Cellview object corresponding to the master of this instance.
name	yes	string	Name of the instance.
numInst	yes	integer	Number of times the instance is iterated (default = `1`).
viewName	no	string	View name of the master for this instance.

© 2020 All Rights Reserved.

Generic Boundary Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
area	no	float	Area of the point array of the boundary in square user units.
assocTextDisplays	no	l_objId	List of text displays with this boundary's object as their associate.
bBox	no	bBox	The boundary's bounding box.
children	no	l_objId	List of objects that have this boundary as their parent.
edgeNames	no	l_string	List of boundary edge names.
figGroup	no	objId	List the ID of the figGroup to which this boundary belongs.
groupMembers	no	l_objId	List of group members for this boundary.
markers	no	l_objId	List of markers for this boundary.
numEdges	no	integer	Number of edges (points) of the boundary.
parent	yes	objId	Parent figure of the boundary.
points	no	l_point	List of points for the specified boundary.
textDisplays	no	l_objId	List of text displays owned by this boundary.

Generic Blockage Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
area	no	float	Area of the blockage in user units.
assocTextDisplays	no	l_objId	List of text displays with this blockage as their associate.
bBox	no	bBox	The blockage's bounding box.
children	no	l_objId	List of objects that have this blockage as their parent.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
density	yes	float	Percentage of the total design area the metal in the specified layer may occupy.
figGroup	no	objId	List the ID of the figGroup to which this blockage belongs.
groupMembers	no	l_objId	List of group members for this blockage.
isOrthogonal	no	boolean	Indicates if the blockage is orthogonal. When `t`, the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When `t`, the blockage is a rectangle.
isPushedDown	yes	boolean	Indicates if the blockage is pushed down from a higher level block.
owner	yes	objId	ID of the owner of the object.
parent	yes	objId	Parent object of the blockage.

type	no	string	Type of blockage.
			`viaRouting`
			`placement`
			`wiring`
			`fill`
			`slot`
			`pin`
			`feedthru`
			`screen`

textDisplays	no	l_objId	List of text displays owned by this blockage.

© 2020 All Rights Reserved.

Generic Via Attributes

Via attributes are common to via objects.

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
`objType` is listed as: `stdVia`, `customVia`, or `nil`.			
bBox	no	bBox	bBox associated with this via.
children	no	l_objId	List of objects that have this via as their parent.
connRoutes	no	l_objId	List of all route start/end connections that connect to this via.

direction	yes	string	Direction of the via.
			`"layer2ToLayer1"`
			Indicates a via which follows a layer2 to layer1 direction
			
			`"layer1ToLayer2"`
			Indicates a via which follows a layer1 to layer2 direction

figGroup	no	objId	List the figure group ID to which this via belongs.
groupMembers	no	l_objId	List of group members for this via.
isUnshielded	yes	boolean	Indicates whether this via (standard or custom) is not shielded.
shieldedNet1	yes	netId	Sets the first shielded net for this via to the specified bit net. A via may shield up to two nets.
shieldedNet2	yes	netId	Sets the second shielded net for this via to the specified bit net.
markers	no	l_objId	List of markers for this via.
net	yes	objId	ID of the net associated with the via.
orient	yes	string	Orientation of the via
			`"R0"` not rotated

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"R90"` rotated 90 degrees
			`"R180"` rotated 180 degrees
			`"R270"` rotated 270 degrees
			`"MY"` mirrored on the y axis
			`"MYR90"` mirrored on the y axis and rotated 90 degrees counterclockwise
			`"MX"` mirrored on the x axis
			`"MXR90"` mirrored on the x axis and rotated 90 degrees counterclockwise
origin	yes	point	Location of the origin of the via (in user units)
parent	yes	objId	Parent figure of the via.
pin	yes	objId	ID of the pin associated with the via.
route	no	objId	ID of the route associated with the via.

routeStatus	yes	string	Status of the route associated with the via.
			`"normal"`
			The normal routing status. The route can be modified manually or by an automatic tool.
			`"fixed"`
			Automatic tools cannot edit the route but the route can be manually edited.
			`"locked"`
			The route cannot be edited by automatic tools or manually.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description

topology	yes	string	`"none"`
			`"stripe"`
			`"standardCellWire"`
			`"IOWire"`
			`"ring"`
			`"padRing"`
			`"blockRing"`
			`"coreWire"`

viaHeader	no	objId	ID of the viaHeader associated with the via.

Finding Via Parameters

Generic Via Header Attributes

Via header attributes are common to via headers.

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
viaDef	no	objId	Database ID for the technology file of the `viaDef` associated with this viaHeader. The return value is `nil` if the viaHeader is not bound.
viaDefName	no	string	Cached string attribute which returns the name of the associated `viaDef`. Usually available even if the viaHeader is not bound.
master	no	objId	ID of the cellview that is acting as the master of the viaHeader. Can be `nil` if the viaHeader is not bound.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
subHeaders	no	l_objId	List of database IDs that identifies sub-via-Headers. The attribute value is `nil` if the viaHeader being queried is not a super-viaHeader.
vias	no	l_objId	Database ID of the vias associated with the viaHeader.

overrideParams	no	l_params	List of override parameters and values if they exist.
			`"layer1Enc"`
			`"layer2Enc"`
			`"imp1Enc"`
			`"imp2Enc"`
			`"layer1Offset"`
			`"layer2Offset"`
			`"cutSpacing"`
			`"originOffset"`
			`"cutSize"`
			`"cutLayer"`
			`"cutColumns"`
			`"cutRows"`
			`"cutWidth"`
			`"cutHeight"`
			`"cutPattern"`

Generic widthSpacingPattern Attributes

Note: Includes Generic Object Attributes.

Attribute	Modify	Type	Description
name	no	string	Name of the object.
patterns	no	l_patterns	List of pattern specifications.
offset	no	float	Distance of the first track from the period track.
repeatOffset	no	Boolean	A flag that specifies whether the offset is applied when the pattern repeats.
shiftColor	no	Boolean	A flag that specifies whether the track color shifts when the pattern repeats.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
startingColor	no	string	Color of the first track. Valid values are "`grayColor`", "`mask1Color`", "`mask2Color`", and "`mask3Color`".

Generic widthSpacingPatternGroup Attributes

Note: Includes Generic Object Attributes.

Attribute	Modify	Type	Description
name	no	string	Name of the object.
patternNames	no	l_patterns	List of pattern names in the group.

Generic relatedSnapPattern Attributes

Note: Includes Generic Object Attributes.

Attribute	Modify	Type	Description
name	no	string	Name of the object.
relatedSnapPattern List	no	l_relatedsnap Patterns	List of related snap pattern specifications.
rowRegionSpecNa mes	no	l_rowRegion SpecNames	List of row region specification names.

Generic widthSpacingSnapPatternDef Attributes

Note: Includes Generic Object Attributes.

Attribute	Modify	Type	Description
name	no	string	Name of the object.
layer	no	integer	Layer ID of the object.
purpose	no	integer	Purpose ID of the object.
period	no	float	Spacing between coarse-grain period tracks.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
direction	no	string	The pattern direction. Valid values are "`horizontal`" and "`vertical`".
offset	no	float	Distance of the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).
snappingLayers	no	l_snapLayers	List of snapping layer specifications.
patternNames	yes	l_string	List of pattern names in the snap pattern definition.
patternGroupNames	yes	l_string	List of pattern group names in the snap pattern definition.
defaultActiveName	yes	string	Name of the default active pattern.

© 2020 All Rights Reserved.

Generic rowRegion Attributes (ICADVM181.1 Only)

The rowRegion Attributes include Generic Object Attributes, Generic Object Attributes, Row Attributes, Generic relatedSnapPattern Attributes, and placeCells attributes.

The following are placeCells rowRegion attributes.

Attribute	Modify	Type	Description
leftBoundaryCell	yes	string	Name of the left boundary cell.
leftBoundaryCellMirrored	yes	Boolean	Specifies whether the left boundary cell is mirrored. The default is `t`.
rightBoundaryCell	yes	string	Name of the right boundary cell.
rightBoundaryCellMirrored	yes	Boolean	Specifies whether the right boundary cell is mirrored. The default is `nil`.
topLeftBoundaryCell	yes	string	Name of the top left boundary cell.
bottomLeftBoundaryCell	yes	string	Name of the bottom left boundary cell.
leftCornerCellsMirrored	yes	Boolean	Specifies whether the left corner cells are mirrored. The default is `nil`.
topRightBoundaryCell	yes	string	Name of the top right boundary cell.
bottomRightBoundaryCell	yes	string	Name of the bottom right boundary cell.
rightCornerCellsMirrored	yes	Boolean	Specifies whether the right corner cells are mirrored. The default is `t`.
topBoundaryCells	yes	l_string	List of top boundary cells.
bottomBoundaryCells	yes	l_string	List of bottom boundary cells.
tapCells	yes	l_string	List of tap cell names.
fillCells	yes	l_string	List of fill cell names.
libList	yes	l_string	List of boundary cell library names.
viewList	yes	l_string	List of boundary cell view names list. The default is `layout`.

© 2020 All Rights Reserved.

Attributes of Cellviews

Cellview Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
anyInstCount	no	integer	Number of instances.
areaBoundaries	no	l_objId	List of boundaries of type `areaBoundaries` that exist in the cellview.
assocTextDisplays	no	l_objId	List of text displays associated with the cellview.
bBox	no	bBox	The cell's bounding box.
blockages	no	l_objId	List of all blockages that exist in the cellview.
cell	no	objId	Cell expressed as a database object.
cellName	no	string	Name of the cell.

cellType	yes	string	`"none"`
			`"block"`
			`"blockRing"`
			`"cover"`
			`"coverBump"`
			`"pad"`
			`"padSpacer"`
			`"core"`
			`"coreSpacer"`
			`"coreAntenna"`
			`"corner"`
			`"softMacro"`
			`"via"`
			`"blockBlackBox"`
			`"padAreaIO"`

cellViewType	no	string	`"maskLayout"`
			`"schematic"`
			`"symbolic"`
			`"schematicSymbol"`

clusters	no	l_objId	List of all clusters that exist in the cellview.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
conns	no	l_objId	List of all connected instance terminals that exist in the cellview.
createTime	no	string	Returns the time of creation of the cellview.
DBUPerUU	yes	float	Number of database units in one user unit.
figGroups	no	l_objId	List of all figGroups that exist in the cellview.
fileName	no	string	Relative path of the cellview (including the directory that defines the library).
fileTimeStamp	no	string	The date and time when the cellview was last saved.
gCellPatterns	no	l_objId	List of all gcellPattern objects that exist in the cellview.
groups	no	l_objId	List of all groups that exist in the cellview.
groupMembers	no	l_objId	List of all group members that exist in the cellview.
guides	no	l_objId	List of all guides that exist in the cellview.
instanceMasters	no	l_objId	List of all cellviews corresponding to masters of instances in the cellview.
instances	no	l_objId	List of all the instances of cellviews in the cellview.
instHeaderRefs	no	l_objId	List of all instance headers for instances of the cellview when the cellview is used as a master (defined only for currently open cellviews).
instHeaders	no	l_objId	List of all instance headers for instances of the cellview.
instRefs	no	l_objId	List of all instances of the cellview when the cellview is used as a master (defined only for currently open cellviews).
isParamCell	no	Boolean	Indicates whether the cellview is a parameterized cell.
layerPurposePairs	no	l_objId	List of all layer-purpose objects that exist in the cellview.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
layerHeaders	no	l_objId	List of all layerHeaders that exist in the cellview.
lib	no	objId	ID of the library that contains the cellview.
libName	no	string	Name of the library that contains the cellview.
lpps	no	l_objId	List of all layer-purpose objects in the cellview.
markers	no	l_objId	List of all markers that exist in the cellview.
memInst	no	see `=>`	List of pairs that describe all individual instances in the cellview. The pairs are defined as lists that contain an instance object and the index of the unique instance.
mode	no	string	Mode the cellview is opened in: “`r`”, “`w`”, “`a`”, or "`s`". Pcell submasters are opened in "`s`" mode (this mode is for internal use and cannot be set by the user).
modifiedButNot Saved	no	Boolean	Indicates whether the cellview has been modified since the last save.
modifiedCounter	no	string	The number of times the cellview has been modified (relative indicator).
mosaics	no	l_objId	List of all the mosaics in the cellview.
needRefresh	no	Boolean	Specifies whether the cellview must be refreshed (`t`) or not (`nil`).
netCount	no	integer	Number of nets in the cellview.
nets	no	l_objId	List of all the nets in the cellview.
objType	no	string	Type of the object, in this case `"cellview"`.
prBoundary	no	l_objId	List of all boundaries of type `PRBoundary` in the cellview .
rows	no	l_objId	List of all row objects that exist in the cellview.
rowHeaders	no	l_objId	List of all rowHeaders that exist in the cellview.
routes	no	l_objId	List of all routes that exist in the cellview.
shapes	no	l_objId	List of all shapes that exist in the cellview.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
signals	no	l_objId	List of all signals that exist in the cellview.
sigNames	no	l_string	List of all signal names that exist in the cellview.
sitePattern	yes	l_objId	List of all sitePatterns that exist in the cellview. Setting a cellview sitePattern to `nil` is equivalent to deleting the sitePattern.
snapBoundary	no	l_objId	List of all boundaries of type `snapBoundary` that exist in the cellview.
steiners	no	l_objId	List of all Steiner objects that exist in the cellview. The returned list represents all Steiner objects in the cellview as Steiners or Steiner Pins.
subMasters	no	l_objId	List of different parameterizations for a parameterized cell.
superMaster	no	objId	Supermaster of a parameterized cell.
termCount	no	integer	Number of terminals in the cellview.
terminals	no	l_objId	List of all terminals in the cellview.
textDisplays	no	l_objId	List of text displays owned by the cellview.
trackPatterns	no	l_objId	List of all track patterns that exist in the cellview.

userUnits	yes	string	Names of user units expressed as strings.
			`"micron"`
			`"millimeter"`
			`"centimeter"`
			`"meter"`
			`"mil"`
			`"inch"`

vias	no	l_objId	List of all via objects that exist in the cellview.
viaHeaders	no	l_objId	List of all viaHeaders that exist in the cellview. These can be custom or standard viaHeaders. These can also be super-viaHeaders, sub-viaHeaders, or simple viaHeaders (which is neither a super-viaHeader nor a sub-viaHeader).

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
viaMasters	no	l_objId	List of masters associated with all the viaHeaders in the cellview.
view	no	objId	View expressed as a database object.
viewName	no	string	Name of the view.
widthSpacingSnapPatternDefs	no	l_objId	List of all widthSpacingSnapPatternDefs in the cellview.

© 2020 All Rights Reserved.

Attributes of Instances, Instance Headers, and Mosaics

- Instance Attributes
- Instance Header Attributes
- Mosaic Attributes

Instance Attributes

Design hierarchy is modeled through the placement of cell instances.

Attribute	Modify	Type	Description
Note: include Generic anyInst Attributes			
conns	no	l_objId	List of all connected instance terminals for this instance (applicable only if the instance is not a mosaic).
clusters	yes	l_objId	List of all clusters that exist in the instance.
blockages	no	l_objId	List all blockages that exist in the instance.
mag	yes	float	Magnification for the instance. Always = `1.0`. Magnification of instances is not supported.
orient	yes	string	Orientation string for this instance:
			`"R0"` not rotated
			`"R90"` rotated 90 degrees
			`"R180"` rotated 180 degrees
			`"R270"` rotated 270 degrees
			`"MY"` mirrored on the y axis
			`"MYR90"` mirrored on the y axis and rotated 90 degrees counterclockwise

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"MX"` mirrored on the x axis
			`"MXR90"` mirrored on the x axis and rotated 90 degrees counterclockwise
physOnly	no	boolean	When `t`, the instance data is physical rather than physical and logical. When a cellview is written from SoCE, the cellview contains EMH data and is physical and logical. Instances inherit this `physOnly` status from the cellview.

source	yes	string	The different sources you can assign to an instance. Valid Values:
			`netlist`
			`timing`
			`dist`
			`test`
			`user`

status	yes	string	Status of this instance for placement
			suggested the object has been placed but it may be placed again by future attempts
			`firm` placement of the object cannot be changed by an automatic placement tool
			`locked` placement of the object cannot be changed
			`placed` the object has been placed but it may be placed again by future attempts

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`unplaced` defined logically in database, but not physically placed in the design
transform	yes	l_points	List of values describing the instance's transformation with respect to the master. The list contains the following information in this order:
			`offset` Same as *xy* attribute
			`orient` Same as *orient* attribute
			`mag` Magnification for the instance. Always = 1.0. Magnification of instances is not supported.
xy	yes	point	xy location of the instance (in user units).

Instance Header Attributes

Instance headers are automatically created and deleted by the database. Properties should not be placed on instance headers and instance headers should not be placed in groups. All of the instances associated with an instance header have the same master as the instance header

Attribute	Modify	Type	Description
Note: includes Generic Object Attributes			
cellName	yes	string	Name of the master cell for this instance header.
instances	no	l_objId	List of instances associated with this instance header.
libName	yes	string	Name of the library that contains the master for this instance header.
master	yes	objId	The master cellview of this instance header.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
timeStamp	no		Time stamps are no longer supported and have been replaced by counters. Return value is NULL.
variants	no	l_objId	List of instance headers for sub-masters of the master of this instance header if the master is parameterized.
viewName	yes	string	Name of the master's view for this instance header.

Mosaic Attributes

A mosaic is a two-dimensional array of instances, stored in a compact representation.

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this mosaic as their associate.
bBox	no	bBox	The mosaic's bounding box.
cellName	no	string	Name of the cell for this mosaic.
children	no	l_objId	List of objects that have this mosaic as their parent.
groupMembers	no	l_objId	List of group members for this mosaic.
instHeader	no	string	Name of the instance header.
instanceList	no	l_objId	List of mosaic instances of masters used in the mosaic.
instTerms	no	l_objId	List of instance terminals that belong to this mosaic.
libName	no	string	Name of the library for this mosaic.
mosaicType	no	string	Type of mosaic "`simple`"
name	yes	string	Name of the mosaic.
numInstances	no	integer	Number of masters in the instance list.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
pin	no	objId	Pin object if the mosaic is associated with a pin.
purpose	yes	string	Layer purpose.
master	no	string	Name of the master instance.
rows	yes	integer	Number of rows in the mosaic.
columns	yes	integer	Number of columns in the mosaic.
tileArray	no	l_tile	Not supported. Row-major vector of mosaic tiles that have pairs of index and icon for the mosaic.
uX	yes	l_float	List of distances (in user units) between adjacent columns.
uY	yes	l_float	List of distances (in user units) between adjacent rows.
viewName	yes	integer	Name of the view for this mosaic.

© 2020 All Rights Reserved.

Attributes of Instance Terminals, Pins, and Terminals

- Instance Terminal Attributes
- Pin Attributes
- Terminal Attributes

Instance Terminal Attributes

An instance terminal describes a net connection at one level of the hierarchy with a net at a lower level of the hierarchy. The connection of the lower level net is made only in the context of the instance associated with the connection. An instance terminal does not have to connect to a net.

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this instTerm as their associate.
connRoutes	no	l_objId	A list of all routes that connected to this instTerm.
direction	yes	string	Direction of the associated terminal. If the terminal does not exist on the master, a value of `"unknown"` is returned.
			`"input"` input terminal
			`"inputOutput"` both input and output terminal
			`"jumper"` jumper terminal
			`"output"` output terminal
			`"switch"` terminal is a switch
			`"unused"` terminal is not used
groupMembers	no	l_objId	List of group members for this instTerm.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
inst	no	objId	Instance to which the instTerm belongs.
markers	no	l_objId	List of all markers that are associated with the instTerm.
name	yes	string	Name of the associated master terminal.
net	yes	objId	Net to which the instTerm is attached.
numBits	no	integer	Width of this instTerm in bits.
term	no	objId	Terminal corresponding to the instTerm.
termPosition	no	integer	Returns the position to which this terminal is assigned. Assigning a position or indicating an order for terminals is typically used by netlisters to compare the pin position to the position specified in the netlist.
textDisplays	no	l_objId	List of all text displays owned by this instTerm.
widthOK	no	boolean	Flag denoting whether the instTerm width is valid with respect to its connected net and instances on which it ends.

Pin Attributes

A pin describes an instance where a physical connection can be made to the associated terminal.

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
accessDir	yes	l_string	List of strings that define directions from which the pin can be accessed. Possible values are: `"left"`, `"right"`, `"top"`, `"bottom"`, `any` combination of those values, and `"none"`. `nil` means unspecified.
accessDirection	yes	integer	Value of *accessDir* described as an integer (for compatibility with previous software versions). They are: `4` (left), `8` (right), `1` (top), `2` (bottom), any summation of those values, and `0` (none). `nil` means unspecified and not 0.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
assocTextDisplays	no	l_objId	List of text displays with this pin as their associate
connRoutes	no	l_objId	A list of all start/end connection of routes that connected to this pin.
fig	no	objId	The first figure that describes the pin.
figs	no	l_objId	Multiple pin figures are supported. `figs` will list all the figures of the pin when there is more than one.
groupMembers	no	l_objId	List of group members for this pin
markers	no	l_objId	List of all markers for this pin.
name	yes	string	Name of the pin. This is sometimes confused with the terminal name, which is used more often
net	yes	objId	Net attached to the pin.
objType	no	string	Type of object, in this case `"pin"`
term	no	objId	Terminal that contains the pin
textDisplays	no	l_objId	List of text displays owned by this pin
type	yes	string	Pin type of the `oaPin` object. `nil`: Specifies that the pin is a default pin, which is connected by normal "default" routing. `core`: Specifies the pin is a "core" ring connection point. `bump`: Specifies the pin is a "bump" connection point. A bump pin should only be connected to a bump.

Terminal Attributes

A terminal allows a net to be connected to another net at the next level up in the hierarchy.

Attribute	Modify	Type	Description

Note: include Generic Object Attributes

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
assocTextDisplays	no	l_objId	List of text displays with this terminal object as their associate
connRoutes	no	l_objId	A list of all routes that connected to the terminal.
direction	yes	string	`"input"` input terminal
			`"inputOutput"` both input and output terminal
			`"jumper"` jumper terminal
			`"output"` output terminal
			`"switch"` terminal is a switch
			`"unused"` terminal is not used
groupMembers	no	l_objId	List of all group members for this terminal.
isInterface	no	boolean	Indicates whether this terminal is an interface terminal. For more information, see Additional Information Related to isInterface Attribute.
markers	no	l_objId	List of all markers for this terminal
mustJoinTerms	no	l_objId	Indicates that terminals on different nets must be connected. See Property Bag Functions for a description of pin modeling. Value values are `nil` (not a mustJoinTerm) or a list of mustJoin terminals in that mustJoin group.
name	yes	string	Name of this terminal.
net	no	objId	Net associated with this terminal.
numBits	no	integer	Width of this terminal in bits.
objType	no	string	Type of object, in this case “term”.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
pinCount	no	integer	Number of pins connected to the terminal.
pinConnectedMethod	yes	string	Sets and gets the pin connection method for this terminal.
physOnly	no	boolean	When `t`, the terminal is physical rather than physical and logical. When a cellview is written from SoCE, the cellview contains EMH data and is physical and logical. Terminals inherit this `physOnly` status from the cellview.
pins	no	l_objId	List of all pins on the terminal.

routeMethod	yes	string	Describes the route method.
			`"geometric"`
			`"symbolic"`

textDisplays	no	l_objId	List of text displays owned by the terminal object.

Additional Information Related to isInterface Attribute

isInterface is a flag on a terminal in a design that specifies whether this terminal is part of the original interface specification of the design. Its value is independent of whether the terminal is implicit or explicit. It is primarily used by the `oa2verilog` translators and by the OpenAcess interface into *Innovus*.

Let's consider a schematic pin `A<0:1>` in Layout XL.

The *Generate All From Source* or *Update Components And Nets* command creates two explicit layout terminals `A<0>` and `A<1>`. Additionally, when the `createImplicitBusTerminals` option is enabled (default value), Layout XL creates an implicit bus terminal `A<0:1>`. Being implicit, this bus terminal does not show up in the Navigator or the `cv~>terminals` SKILL API.

Layout XL also sets the isInterface flag of the implicit bus terminal `A<0:1>` to true and the isInterface flag of the explicit bit terminals `A<0>` and `A<1>` to false. This maintains the information of the original schematic design, where the `A<0:1>` terminal was the interface and the bit terminals were no-interface.

If the `createImplicitBusTerminals` option is disabled, layout only has the `A<0>` and `A<1>` explicit terminals and the isInterface flag on both the terminals is set to true.

© 2020 All Rights Reserved.

Attributes of Bus Definitions, Nets, and Signals

Bus Net Definition Attributes

Attribute	Modify		Description
Note: include Generic Object Attributes			
baseName	no	string	Returns base name for the busNetDef.
bitOrder	no	string	Returns bit order for the busNetDef. If the busNetDef is implicit, “none” will be returned.
minIndex	no	integer	Returns smallest bit number in use for the busNetDef.
maxIndex	no	integer	Returns largest bit number in use for the busNetDef.
numBits	no	integer	Returns the number of bits covered for this busNetDef.
numBitsUsed	no	integer	Returns the number of busNetBits used for this busNetDef.

Bus Terminal Definition Attributes

Attribute	Modify		Description
Note: include Generic Object Attributes			
baseName	no	string	Returns base name for the busTermDef.
bitOrder	no	integer	Returns bit order for the busTermDef. If the busTermDef is implicit, “none” will be returned.
minIndex	no	integer	Returns smallest bit number in use for the busTermDef.
maxIndex	no	integer	Returns largest bit number in use for the busTermDef.
numBits	no	integer	Returns the number of bits covered for this busTermDef.
numBitsUsed	no	integer	Returns the number of busTermBits used for the busTermDef.

© 2020 All Rights Reserved.

Vector Instance Definition Attributes

Attribute	Modify		Description
Note: include Generic Object Attributes			
baseName	no	string	Returns base name for the vectorInstDef.
minIndex	no	integer	Returns smallest bit number in use for the vectorInstDef.
maxIndex	no	integer	Returns largest bit number in use for the vectorInstDef.
bitOrder	no	integer	Returns or sets bit order for the vectorInstDef. If the vectorInstDef is implicit, “none” will be returned. bitOrder can be set if the bus definition is explicit. Valid values for modification: String value.
numBits	no	integer	Returns the number of bits covered for this vectorInstDef.
numBitsUsed	no	integer	Returns the number of vectorInstBits used for this vectorInstDef.

Net Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
allInstTerms	no	l_objId	List of instance terminals connected to this net and any of the net’s subnets.
aliases	no	l_string	List of names depicting alternative names for the signal (no duplicates are printed).
assocTextDisplays	no	l_objId	List of text displays with this net object as their associate.
connStatus	yes	string	The status of net connections.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"connected"` The connections are physically implemented by the objects of this net (for example, abutting).
			`"edgeConnected"` The connections are physically implemented, but one or more connections are made through edge-assigned pins.
			`"connectOutside"` The connections are not physically implemented and need to be.
			`"needToConnect"` One or more of the connections are not implemented and need to be connected above them in the hierarchy to implement them.
criticality	yes	integer	Criticality of the net (value from 0 to 255).
figs	no	l_objId	List of all figures that use this net.
groupMembers	no	l_objId	List of all group members for this object.
instTermCount	no	integer	Number of instance terminals connected to the net.
instTerms	no	l_objId	List of all instTerms connected to the net.
isGlobal	yes	boolean	Defines whether the signal is known globally.
markers	no	l_objId	List of all markers associated with the net.
memInstTerms	no	see =>	List of lists, each containing an instance terminal object followed by an integer index describing the bit of the instance terminal the signal is attached t.
memNets	no	see =>	List of lists, each containing a net object followed by an integer index, describing which bit of the net the signal associates.
memTerms	no	see =>	List of lists, each containing a terminal object followed by an integer index describing the bit of the terminal the signal to which it is attached.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
name	no	string	Name of the net.
numBits	no	integer	Number of signals in this net.
parent	no	objId	Parent net if this is a child net, otherwise `nil`.
pins	no	l_objId	List of pins that use the terminal associated with this net.

routePattern	yes	string	Specifies the type of route pattern.
			`"steiner"`
			`"balanced"`
			`"trunk"`

routes	no	l_objId	List of all routes associated with the net.
shieldNet1	yes	netId	Shield net 1 value for this net. A net can have zero, one, or two shield nets.
shieldNet2	yes	netId	Shield net 2 value for this net.
signals	no	l_objId	List of signals associated with this net.
sigNames	no	l_string	List of names of the signals in this net.
sigType	yes	objId	Type of the signal. One of the following:
			`"analog"` Distinguishes between analog and digital signals in mixed-signal designs.
			`"clock"` A timing critical clock
			`"ground"` The reference signal for the cellview
			`"optical"` An optical signal
			`"singleModeOptical"` A single mode optical signal
			`"multiMode"` A multimode optical signal

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"reset"` A reset signal
			`"scan"` Part of a test scan chain
			`"signal"` An ordinary signal
			`"supply"` Provides power for the cellview
			`"testLatch"` Part of a test scan chain
			`"tieHi"` Tied to logical 1
			`"tieLo"` Tied to logical 0
			`"tieOff"` Tied to logical 1 or logical 0 but does not specify the value

source	yes	string	The different sources you can assign to a net. Valid Values:
			`netlist`
			`timing`
			`dist`
			`test`
			`user`

subnet	no	l_objId	List of immediate child nets (child nets attached directly to this net).
subnets	no	l_objId	List of all child nets (child nets that traverse the entire net hierarchy).
term	no	objId	Terminal associated with the net.
terms	no	l_objId	List all terminals on a net. However, more than on terminal is not supported in DFII and indicates a shorted terminal.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
textDisplays	no	l_objId	List of text displays owned by this net object.

Signal Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
aliases	no	l_string	List of names depicting alternative names for the signal (no duplicates are printed).
groupMembers	no	l_objId	List of all group members for this object.
isGlobal	yes	boolean	Defines whether the signal is known globally.
memInstTerms	no	see =>	List of lists, each containing an instance terminal object followed by an integer index describing the bit of the instance terminal the signal is attached t.
memNets	no	see =>	List of lists, each containing a net object followed by an integer index, describing which bit of the net the signal associates.
memTerms	no	see =>	List of lists, each containing a terminal object followed by an integer index describing the bit of the terminal the signal to which it is attached.
name	no	string	Name of the net.
sigType	no	objId	Type of signal. One of the following:
			`"analog"` Distinguishes between analog and digital signals in mixed-signal designs
			`"clock"` A timing critical clock
			`"ground"` The reference signal for the cellview
			`"reset"` A reset signal

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"scan"` Part of a test scan chain
			`"signal"` An ordinary signal
			`"supply"` Provides power for the cellview
			`"testLatch"` Part of a test scan chain
			`"tieHi"` Tied to logical 1
			`"tieLo"` Tied to logical 0
			`"tieOff"` Tied to logical 1 or logical 0 but does not specify the value

© 2020 All Rights Reserved.

Attributes of Layer Purpose Pairs and Layer Headers

Layer-Purpose Pair Attributes

Layer-purpose pairs are automatically created and deleted by the database. This means that properties should not be placed on layer-purpose pairs, and groups should not contain layer-purpose pairs. All shapes associated with a layer-purpose pair have the same layer and purpose as the layer-purpose pair. A layer-purpose pair can be present, but not contain any shapes.

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
layerName	no	string	Name of the layer.
layerNum	no	integer	Layer number of this layer.
layerHeader	no	objId	ID of the layerHeader for the layer.
nShapes	no	integer	Number of shapes on this layer and purpose.
purpose	no	string	The name of the layer purpose.
markers	no	l_objId	List of all markers on this layer purpose pair.
shapes	no	l_objId	List of all shapes with this layer and purpose.

© 2020 All Rights Reserved.

Layer Header Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
blockageBBox	no	bBox	Blockage bBox.
blockages	no	l_objId	List of blockage IDs that are associated with the layerHeader.
guideBBox	no	bBox	Guide bBox.
guides	no	l_objId	List of guides.
layer	no	string	Layer name.
lpps	no	l_objId	List of lpps ID that are associated with the layerHeader.
steinerBBox	no	bBox	Steiner bBox.
steiners	no	l_objId	List of steiner IDs that are associated with the layerHeader.

Layer Header attributes can be accessed as follows:

```
layerHeader~>objType => "layerHeader"
cv~>layerHeaders => l_layerHeaderIds / nil
lpp~>layerHeader => d_layerHeaderId / nil
```


© 2020 All Rights Reserved.

Attributes of Groups, Group Members, and Properties

- Group Attributes
- Group Member Attributes
- Property Attributes

© 2020 All Rights Reserved.

Group Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
deleteWhen	yes	string	The values that can be used are:
			`deleteNever` The group itself is never automatically deleted no matter which group member deleted.
			`deleteFirst` The group itself is automatically deleted once the first group member is deleted.
			`deleteLast` The group itself is automatically deleted once the last group member is deleted
defName	no	string	The name of a group definition object associated with this group.
groupMembers	no	l_objId	List of group members contained in this group
isCollection	yes	boolean	`t` if the group is a collection, `nil` otherwise
isOrdered	yes	boolean	`t` if the groupMembers are ordered, `nil` otherwise
isSet	yes	boolean	`t` if the group is a set, `nil` otherwise
isUniqueName	yes	boolean	`t` if the group name is unique, `nil` otherwise
members	no	l_objId	List of group members contained in this group
name	yes	string	Name of this group
objects	no	l_objId	List of objects contained in this group (referred to by group members)

Group Member Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
group	no	objId	Group containing this group member
object	no	objId	Object this group member refers to

© 2020 All Rights Reserved.

Property Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this property object as their associate.
enum	no	string	If the property's type is *fileName*, *ILExpr*, *NLPExpr*, or *string*, it can be an enumerated property, which can assume only one of the values in this list. *Enum* is a range for string-valued properties.
name	yes	string	Name of the property
object	no	objId	Database object with this property
range	no	see =>	If the property value is *float*, *int*, or *time*, this property is either `nil` or a list containing the lower and upper bound. The type of this attribute, therefore, depends on *valueType*.
value	yes	see =>	Value of this property. The type of this attribute depends on *valueType.*
valueType	no	string	The type of the value of this property:
			`"boolean"` Boolean
			`"fileName"` file name string
			`"float"` Floating-point integer
			`"hierProp"` Property list
			`"ILExpr"` IL expression string
			`"ILList"` IL list
			`"int"` Integer

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"NLPExpr"` NLP expression string
			`"string"` String
			`"time"` Time-valued string

© 2020 All Rights Reserved.

Attributes of Objects and Figures

- Blockages
  - Area Blockage Attributes
  - Area Halo Attributes
  - Layer Blockage Attributes
- Boundaries
  - Area Boundary Attributes
  - Cluster Boundary Attributes
  - prBoundary Attributes
  - Snap Boundary Attributes
    - Cluster Attributes
- figGroup Attributes
- GCellPatterns Attributes
- Guide Attributes
- Marker Attributes
- Route Attributes
- Row Attributes
  - Row Header Attributes
- Steiner Attributes
- Custom Via Attributes
- Standard viaVariant. Attributes
- Custom Via Header Attributes
- Standard Via Header Attributes
- Track Pattern Attributes

© 2020 All Rights Reserved.

Area Blockage Attributes

Attribute	Modify	Type	Description
Note: include Generic Blockage Attributes			
nPoints	no	l_point	Number of points for the blockage.
objType	no	string	Type of object, in this case `"areaBlockage"`.
points	yes	l_points	List of points of the blockage.
type	no	string	Type of blockage `placement`
isSoft	yes	boolean	If `t`, standard cells are not placed within the blockage area during initial placement. Though later optimization phases can use the blockage area.

Area Halo Attributes

Attribute	Modify	Type	Description
Note: include Generic Blockage Attributes			
nPoints	no	l_point	Number of points for the blockage.
offsets	yes	l_points	List of points for the offset of the area halo.
points	no	l_points	List of points of the blockage.

type	no	string	Type of blockage
			`routing`
			`wiring`
			`fill`
			`slot`
			`pin`
			`feedthru`
			`screen`

isSoft	no	boolean	Indicates the soft attribute for a derived blockage. An `areaHalo` with this attribute indicates that standard cells should not be placed within the halo, but later optimization phases can use the halo area.

© 2020 All Rights Reserved.

Example:

```
areaHaloId~>offsets
=> list(f_top f_bottom f_left f_right) / nil
areaHaloId~>offsets = list(f_top f_bottom f_left f_right)
=> t / nil
```


© 2020 All Rights Reserved.

Layer Blockage Attributes

Attribute	Modify	Type	Description
Note: include Generic Blockage Attributes			
effectiveWidth	yes	float	The effective width in user units. Specifies the minSpacing spacing value from the layer spacing rule or table spacing rules to prevent placement or warn of a violation if objects are closer than the rule specification.
layer	yes	string	Layer of the blockage.
layerHeader	yes	l_objId	LayerHeader for the layer of the blockage.
nPoints	no	l_point	Number of points for the blockage.
objType	no	string	Type of object, in this case `"layerBlockage"`.
points	yes	l_points	List of points of the blockage.

type	no	string	Type of blockage
			`routing`
			`wiring`
			`fill`
			`slot`
			`pin`
			`feedthru`
			`screen`

spacing	no	g_value	Sets the spacing attribute for a blockage. The spacing and effective width attributes are mutually exclusive. By default, the spacing attribute is unset, which means the blockage effective width is used to set spacing.
allowPGNet	no	boolean	Sets the `allowPGNet` attribute for this blockage. When set, shapes for the signal types power and ground are allowed within the blockage area

Blockage attributes can be accessed as follows using a cellview database ID:

```
blockageId~> attributeName
```


© 2020 All Rights Reserved.

Blockage attributes can be modified as follows:

```
blockageId~> attributeName = attributeValue
```

Example:

```
layerBlockageId~>layer=> tx_layer /nil
layerBlockageId~>layer = tx_layer
layerBlockageId~>layerHeader => d_layerHeaderId /nil
layerBlockageId~>points = l_points => t /nil
```


© 2020 All Rights Reserved.

Layer Halo Attributes

Attribute	Modify	Type	Description
Note: include Generic Blockage Attributes			
nPoints	no	l_point	Number of points for the blockage.
points	yes	l_points	List of points of the blockage.

Area Boundary Attributes

Attribute	Modify	Type	Description
Note: include Generic Boundary Attributes			
name	yes	string	Name of boundary.
isOrthogonal	no	boolean	Indicates if the blockage is orthogonal. When `t`, the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When `t`, the blockage is a rectangle.

Cluster Boundary Attributes

Attribute	Modify	Type	Description
Note: include Generic Boundary Attributes			
clusters	no	objId	ID of the cluster associated with the boundary.
isOrthogonal	no	boolean	Indicates if the blockage is orthogonal. When `t`, the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When `t`, the blockage is a rectangle.
name	yes	string	Name of boundary.
objType	no	string	Type of object `"clusterBoundary"`.

© 2020 All Rights Reserved.

prBoundary Attributes

Attribute	Modify	Type	Description
Note: include Generic Boundary Attributes			
blockages	no	l_objId	List of blockages associated with the prBoundary.
coreBoxSpec	yes	list	List the coreBoxSpec of the boundary. See example below.
IOBox	yes	list	IO box of the boundary. See example below.
isOrthogonal	no	boolean	Indicates if the blockage is orthogonal. When `t`, the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When `t`, the blockage is a rectangle.
objType	no	string	Type of object `"PRBoundary"`.

Example:

```
prBoundaryId~>coreBoxSpec
=> l_coreBoxSpec / nil
prBoundaryId~>coreBoxSpec = l_coreBoxSpec
=> t / nil
```

The coreBoxSpec is a list of the following format:

```
coreBoxSpec = list(l_bBox g_isHorizontal f_rowSpacing
           t_spacingType t_flipType g_siteDef)
g_siteDef = d_siteDefId | t_siteDefName
```

When retrieving the coreBoxSpec from a prBoundary, if the siteDef of the coreBoxSpec is bound, an ID will be return, otherwise, a name will be returned. When setting the coreBoxSpec, either a name or an ID is accepted.

```
prBoundaryId~>IOBox
=> l_bBox / nil
prBoundaryId~>IOBox = l_bBox
=> t / nil
```

prBoundaryId~>blockages

```
=> l_blockageId / nil
```


© 2020 All Rights Reserved.

Snap Boundary Attributes

Attribute	Modify	Type	Description
Note: include Generic Boundary Attributes			
objType	no	string	Type of object `"snapBoundary"`.

Cluster Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
blockages	no	l_objId	List of text displays with this cluster as their associate.
boundaries	no	l_objId	List of clusterBoundaries owned by this cluster. The list is automatically updated whenever a clusterBoundary is deleted or added for a specific cluster.
clusterType	yes	string	Type of cluster.
			`inclusive` Instances other than the instances in this cluster may be placed in the cluster's region.
			`exclusive` Only instances in this cluster may be placed in the cluster's region.
			`suggested` Instances in this cluster may be placed outside the cluster's region.
groupMembers	no	l_objId	List of group members for this object.
instances	no	l_objId	List of instances that are direct members of this cluster. Instances can be added to or removed from a cluster using *`instanceId`*`~>cluster`.
isParent	no	boolean	Indicates whether the cluster has other clusters (child-clusters or sub-clusters) as its members.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
markers	no	l_objId	List of all markers associated with this cluster.
name	yes	string	Name of the cluster. The name can be modified if the cellview does not have a cluster of the same name.
parent	yes	objId	Parent object database ID of the cluster object. The cluster can be assigned a new parent as long as it does not create a cyclic relationship of the cluster hierarchy. The parent can also be set to `nil` to break a parent-child relationship.
subClusters	no	l_objId	List of clusters (sub-clusters or child-clusters) that are direct members of this cluster. Clusters can be added to, or removed from a cluster using `clusterId~>parent`.

© 2020 All Rights Reserved.

figGroup Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this figure object as their associate.
bBox	no	bBox	The figure's bounding box.
children	no	l_objId	List of figures that have this figures as their parent.
figGroup	no	objId	List the ID of the figGroup to which this figure belongs.
figs	no	l_objId	List of all figures that are members of this figGroup.
groupMembers	no	l_objId	List of group members for this object.
isAnyInst	no	boolean	Indicates whether the figure is an instance.
isShape	no	boolean	Indicates whether the figure is a shape.
markers	yes	l_objId	List of all markers for this object.
parent	yes	objId	The figure's parent.
name	yes	string	Name of the figure.
orient	yes	string	Orientation of the figure
			`"R0"` not rotated
			`"R90"` rotated 90 degrees
			`"R180"` rotated 180 degrees
			`"R270"` rotated 270 degrees
			`"MY"` mirrored on the y axis
			`"MYR90"` mirrored on the y axis and rotated 90 degrees counterclockwise

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"MX"` mirrored on the x axis
			`"MXR90"` mirrored on the x axis and rotated 90 degrees counterclockwise
origin	yes	point	Location of the origin of the figGroup (in user units)
ordered	yes	boolean	When `t`, members in the figGroup will maintain the same order as when they were added. When `nil`, members can be removed in any order.
textDisplays	no	l_objId	List of text displays owned by this figure object.
type	no	string	Type of figGroup: none modgen syncClone placeArea rowFigure

© 2020 All Rights Reserved.

GCellPatterns Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this object as their associate.
children	no	l_objId	List of objects that have this object as their parent.
count	no	x_count	The number of edges defined by the pattern. When there is a single GCellPattern for each direction, the pattern declares count edges that define count-1 columns or rows. These columns or rows will have indexes from 0 to count-2 in the congestion map.
groupMembers	no	l_objId	List of group members for this object.
isHorizontal	no	boolean	Indicates whether the gCell pattern is horizontal or not. GCellPatterns with isHorizontal set to `t` will define columns, and GCellPatterns with isHorizontal set to `nil` will define rows.
markers	no	l_objId	List of all markers for this object.
parent	no	objId	The gCellPattern parent.
space	no	n_space	The spacing for the pattern.
start	no	n_coord	The starting coordinate for the pattern. The start of the GCellPattern with isHorizontal set to `t` will be the left X coordinate of the first column and the startCoord of the GCellPattern with isHorizontal set to `nil` will define the bottom Y coordinate of the lowest row.
textDisplays	no	l_objId	List of text displays owned by the object.

Guide Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
endPoint	yes	l_points	Ending point of the guide.
startPoint	yes	l_points	Beginning point of the guide.
followLayers	yes	boolean	Indicates whether the layers specified must be followed when attempting to replace this guide with physical routing.
route	no	objId	ID of the attached route.
startLayer	yes	string	Start layer of the guide.
endLayer	yes	string	End layer of the guide.

© 2020 All Rights Reserved.

Marker Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this figure object as their associate.
bBox	yes	bBox	The figure's bounding box.
children	no	l_objId	List of shapes that have this object as their parent.
deleteWhen	yes	string	Indicates when markers are automatically deleted. When modifying the attribute, `t` is returned if the `deleteWhen` value is set successfully. Returns `nil` if the set is unsuccessful.
			`deleteNever` Delete the marker only by explicit deletion.
			`deleteFirst` Delete the marker when any associated object is removed.
			`deleteLast` Delete the marker when all associated objects are removed.
			`deleteModify` Delete the marker when any associated object is modified.
groupMembers	no	l_objId	List of group members for this object.
isClosed	yes	boolean	Indicates if the pointArray of the marker is closed.
isVisible	yes	boolean	Indicates if the marker is visible.
msg	yes	string	Message associated with the marker.
nPoints	no	integer	Number of marker points.
objects	no	l_objId	List of object database IDs associated with the marker.
parent	yes	objId	Parent figure of the marker.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
points	yes	l_points	List of marker points.
severity	yes	string	Indicates the severity of the violation shown by the marker. Returns one of the following predefined values. If a value is set other than the predefined values, `unsupported` is returned.
			`acknowledgedWarning` The warning has been acknowledged by the designer.
			`annotation` User generated information.
			`criticalError` Tool generated error. Usually requires a design review to sign- off.
			`error` Tool generated error.
			`fatalError` Tool generated error. Cannot be signed off, as it will always result in a non function /non manufacturable design. Such error are generated usually from the process end (design rule) where the fab/foundry does not want to allow sign-offs.
			`info` Tool generated information.
			`signedOffCriticalError` The error has been reviewed by the design team, and has been signed-off.
			`signedOffError` The error has been signed-off.
			`warning` Tool generated warning.
			`unsupported` None of the above values.
shortMsg	yes	string	shortMsg associated with the marker.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
tool	yes	string	Indicates name of the tool associated with the marker.
textDisplays	no	l_objId	List of text displays owned by the terminal object.

© 2020 All Rights Reserved.

Route Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this figure object as their associate.
bBox	no	bBox	The figure's bounding box.
children	no	l_objId	List of shapes that have this object as their parent.
figGroup	no	objId	List the ID of the figGroup to which this route belongs.
groupMembers	no	l_objId	List of group members for this object.
hasGuide	no	boolean	Indicating whether the route has one or more guides.
isContiguous	no	boolean	Indicates whether all objects in the route, which may be pathSegs, vias and/or guides, are contiguous. Since a route can span multiple layers, the contiguity of a route is determined on the basis of the points at which each adjacent object connects to the next and the layer where each connection is made. Specifically, this function returns true if the end point of each object matches the begin point of the next object on the route and if the layers of objects at their connection points match. Also returns true for single-object routes or zero-object (empty) routes.
isGlobal	yes	boolean	Boolean attribute that indicates whether this route has been marked as a global route.
markers	no	l_objId	List of markers for this route
net	yes	objId	ID of the net assigned to the route. If the route has does not have an associated net, the value is `nil`.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
numObjects	no	integer	Number of route elements in the route. The value should always be greater than or equal to 1. The value is updated automatically when the number of route elements in the route are modified.
objects	yes	l_objId	

	Describes each route element in the route. The value is a list which has a variable number of sub-lists, each indicating an individual route element as follows:
	`'(l_routeElement1 l_routeElement2 l_routeElement3 ... l_routeElementN)`
	Each sub-list can represent either a segment, customVia, or a stdVia. The first element of the sub-list is always a symbol value to distinguish the route element type. Depending upon this value, a specific format exists for each type as described below. l_routeElement can be only one of the following:
	`'('segment l_beginLocationPoint l_endLocationPoint t_layerName f_segmentWidth)`
	`'('customVia l_locationPoint t_orient d_customViaDefId l_customViaParams)`
	`'('stdVia l_locationPoint t_orient d_stdViaDefId l_stdViaParams)`		

parent	yes	objId	Parent figure of the route.
routeStatus	yes	string	Route status. `normal`, `fixed`, `locked`
startConn endConn	yes	objId	Database ID of the object which connects at the start or end of the route. If there is no object that connects to the start or end of the route, the value is `nil`. Valid start/end objects: steiner, steinerPin, pin, term, instTerm, shape, route.
textDisplays	no	l_objId	List of text displays owned by the terminal object.

© 2020 All Rights Reserved.

Row Attributes

Attribute	Modify	Type	Description
Note: include Generic Figure Attributes			
name	yes	string	Name of the specified row.
rowHeader	no	objId	Database ID of the rowHeader associated with the specified row.
siteDef	yes	objId	Database ID of the siteDef associated with the specified row.
siteOrient	yes	string	Orientation of the `siteDef` associated with the specified row. Valid values for modification: `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, `MXR90`.
numSites	yes	integer	Number of sites for the specified row.
xy	yes	l_float	Origin point for the specified row.
orient	yes	string	Orientation of the row. Valid values for modification: `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, `MXR90`.

Row Header Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
bBox	no	bBox	bBox of all the rows that refer to the row header.
rows	no	l_objId	List of rows that refer to the row header.
siteDef	no	objId	Database ID of the siteDef associated with the row header.
siteDefHeight	no	float	Height of the siteDef associated with the row header.
siteDefName	no	string	Name of the siteDef associated with the row header.
siteDefWidth	no	float	Width of the siteDef associated with the row header.

© 2020 All Rights Reserved.

Steiner Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
bBox	yes	bBox	bBox associated with the specified steiner
connRoutes	no	l_objId	A list of route IDs that connected to the steiner.
layer	yes	string	Number of layer associated with the Steiner.
layerHeader	no	objId	Database ID of the layer header associated with the Steiner.

Steiner attributes can be accessed as follows:

```
steinerId~> attributeName
steinerPinId~> attributeName
```

Steiner pin attributes can be modified as follows:

```
steinerPinId~> attributeName = attributeValue
```

Example:

```
steinerId~>layer => t_layerNum
steinerId~>layer = tx_layerNum => t
```

Track Pattern Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
assocTextDisplays	no	l_objId	List of text displays with this object as their associate.
children	no	l_objId	List of shapes that have this object as their parent.
groupMembers	no	l_objId	List of group members for this object.
isHorizontal	yes	boolean	Indicates whether the track pattern is horizontal or not. When `t`, the track pattern is horizontal.
markers	no	l_objId	List of markers associated with the track pattern.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
numTracks	yes	integer	Number of track lines of the specified track pattern.
parent	no	objId	Parent figure of the track pattern.
routingLayer	yes	string or integer	Indicates the name of the routing layer of the specified track pattern. Number or name of the routing layer can be used for the attribute value. If the given value is `nil`, the routing layer of the track pattern will be removed.
space	yes	float	Value of the space between tracking lines of the specified track pattern.
start	yes	float	Coordinate where the track pattern begins.
textDisplays	no	l_objId	List of text displays owned by the terminal object.

© 2020 All Rights Reserved.

Custom Via Attributes

Attribute	Modify	Type	Description
Note: include Generic Via Attributes			
cellName	no	string	Name of the cellview to which the via belongs.
libName	no	string	Name of library to which the via belongs.
viewName	no	string	Name of the view to which the via belongs.

Standard Via Attributes

Attribute	Modify	Type	Description
include Generic Via Attributes			
layer1Enc	yes	float	List of distances from the cut edge to the edge of layer 1 in horizontal and vertical directions, in user units. Standard vias are connections between two adjacent signal layers on the chip that are being connected. The cut layer is where the chip processing cuts away the oxide between the two adjacent signal layers to provide the connection between them. The layer enclosure via parameters specify the width and height for the rectangles on the conducting layers that lie above and beneath the cut. **Example** `list("layer1Enc" '(0.25 0.75))`
layer2Enc	yes	float	List of distances from the cut edge to the edge of layer 2 in horizontal and vertical directions, in user units. See layer1Enc. **Example** `list("layer2Enc" '(0.25 0.75))`

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
imp1Enc	yes	float	List of distances from the edge of layer 1 to the edge of implant layer 1 in horizontal and vertical directions, user units. Some technologies add implant shapes to vias between the Metal1 and Diffusion layers. The implant enclosure via parameters specify the minimum width and height for the implant that surrounds the layer. If there are no layer offsets, the implant enclosures are centered on the cut relative to the layer enclosure. When layer offset is applied, it affects both the layer enclosure and the implant enclosure. (The layer rectangles and the implant rectangles always have a common center.) **Example** `list("imp1Enc" '(0.4 0.6))`
imp2Enc	yes	float	List of distances from the edge of layer 2 to the edge of implant layer 2 in horizontal and vertical directions, user units. See imp1Enc. **Example** `list("imp2Enc" '(0.4 0.6))`
layer1Offset	yes	float	List of offsets for layer 1 in horizontal and vertical directions, in user units. In cases where Layer1 and Layer2 are not aligned on the center of the cut, the layer offset via parameters specify the amount of offset between the center of the layer and the center of the cut. The offset is added to the coordinates of the corresponding rectangle, so a positive offset moves the rectangle up and to the right. **Example** `list("layer1Offset" '(0.1 0.2))`

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
layer2Offset	yes	float	List of offsets for layer 2 in horizontal and vertical directions, in user units. See layer1Offset. **Example** `list("layer2Offset" '(0.1 0.2))`
cutSpacing	yes	float	List of edge-to-edge distances between cuts in horizontal and vertical directions, in user units. When large via cuts need to be implemented as a series of small rectangles, the spacing, columns, and rows via parameters determine the number of cut rectangles and the dimensions of the entire cut grid. cutSpacing specifies the spacing between the cuts in multiple cut vias as a vector in X and Y. **Example** `list("cutSpacing" '(2.5 2.5))`
originOffset	yes	float	List of offsets for the origin of the via in horizontal and vertical directions, in user units. The origin of a via is the `(0,0)` point in the coordinates for the shapes in the via. The originOffset specifies an offset value for the origin. The effect of specifying a value for the origin offset can be viewed in two different but equivalent ways: Subtracting that value from the origin's point, which defaults to `(0, 0)`, without changing any of the coordinates of the other shapes in the via relative to the default origin. Adding that value to the coordinates of every shape within the via. **Example** `list("originOffset" '(0.5 0.5))`

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
cutLayer	yes	Integer	Name of the cut layer. **Example** `list("cutLayer" "Cont1")`
cutColumns	yes	int	Number of columns in the array of cuts. When large via cuts need to be implemented as a series of small rectangles, the spacing, columns, and rows via parameters determine the number of cut rectangles and the dimensions of the entire cut grid. **Example** `list("cutRows" 3)`
cutRows	yes	integer	Number of rows in the array of cut. See cutColumns. **Example** `list("cutRows" 3)`
cutWidth	yes	float	Width of the cut layer. The cut width, and height, via parameters determine the size of each cut rectangle. By default, the origin (the `0,0` point of the coordinates) is at the center of the cut rectangle. For vias with multiple cut rectangles, the default origin is in the center of the cut array. See originOffset to move this default. **Example** `list("cutWidth" 1.3)`
cutHeight	yes	float	Height of the cut layer. See cutWidth. **Example** `list("cutHeight" 1.4)`

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
cutPattern		string	A string of hex characters that specify the pattern. The pattern must have an even number of hex characters. Alpha hex characters may be specified either in upper or lower case. The hex string is interpreted as an array of binary numbers. Each `1` bit in the array represents the placement of a cut. The bit stream is interpreted in row, column order. Rows and columns start in the lower left corner of the cut array. If there are not enough cuts to fill the last byte of the pattern the pattern must be padded with extra `1` bits. **Example** `list("cutPattern" "4fff")` Bit pattern: `0100 1111 1111 1111` **Note:** Here the last four bits are needed to pad out the last byte because there are only 12 cuts in the array and we need 16 bits to form 2 bytes.

Custom Via Header Attributes

Attribute	Modify	Type	Description
Note: include Generic Via Header Attributes			

Standard Via Header Attributes

Attribute	Modify	Type	Description
Note: include Generic Via Header Attributes			

© 2020 All Rights Reserved.

Custom viaVariant Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
tech	no	objId	Techfile attached to the cellview of the viaVariant. If no techfile is attached, the function will work its way up the hierarchy from the cellview level to the cell level, and finally, to the library level.
name	no	string	Name of the viaVariant.
params	no	l_objId	List of viaVariant and values in the given syntax: `(("param1" value1) ("param2" value2)...)`
viaDef	no	objId	ID of the viaDef associated with the given viaVariant
viaDefName	no	objId	Name of the viaDef associated with the given viaVariant.

Standard viaVariant. Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes			
`tech`	no	objId	Techfile attached to the cellview of the viaVariant. If no techfile is attached, the function will work its way up the hierarchy from the cellview level to the cell level, and finally, to the library level.
`name`	no	string	Name of the viaVariant.
`params`	no	l_objId	List of viaVariant parameter values.
`paramslist`	no	l_objId	List of viaVariant and values in the given syntax: `(("param1" value1) ("param2" value2)...)`
`viaDef`	no	objId	ID of the viaDef associated with the given viaVariant.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
`viaDefName`	no	objId	Name of the viaDef associated with the given viaVariant.

© 2020 All Rights Reserved.

Attributes of Shapes

- Arc Attributes
- Donut Attributes
- Dot Attributes
- Ellipse Attributes
- Label Attributes
- Line Attributes
- Path Attributes
- pathSeg Attributes
- Polygon Attributes
- Rectangle Attributes
- Text Display Attributes

Arc Attributes

The database models elliptical arcs with two bounding boxes; the outer bounding box defines the ellipse and the inner bounding box selects the portion of the ellipse that is the arc. The major and minor axes of the ellipse are parallel to the x and y axes. Once created, an arc can be modified by specifying the starting and stopping angle (in radians) of the arc.

The sweep of the arc begins at *startAngle* and proceeds counterclockwise until it ends at *stopAngle.* The sweep must be no more than pi (180 degrees). *arc~>bBox* defines the

© 2020 All Rights Reserved.

bounding box of just the arc, whereas *arc~>ellipseBBox* defines the bounding box of the entire ellipse that has the arc as a piece of it.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
ellipseBBox	yes	bBox	Bounding box of the ellipse containing the arc.
startAngle	yes	float	Starting angle of the arc (in radians).
stopAngle	yes	float	Stopping angle of the arc (counterclockwise in radians).

© 2020 All Rights Reserved.

Donut Attributes

A donut is circular and is modeled using two square bounding boxes; the outer bounding box defines the outer ellipse and the inner bounding box defines the donut hole.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
innerRadius	yes	float	Distance in user units from the donut's center to the inner circle.
outerRadius	yes	float	Distance in user units from the donut's center to the outer circle (must be greater than *innerRadius*).
xy	yes	float	Point describing donut's center (in user units).

Dot Attributes

A dot consists of a point. It can be used for marking locations, such as pin location, feed through, and so forth.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
height	yes	float	Height of the dot.
width	yes	float	Width of the dot.
xy	yes	point	Point describing dot's location (in user units).

© 2020 All Rights Reserved.

Ellipse Attributes

An ellipse is modeled with a rectangular bounding box. The major and minor axes of the ellipse are parallel to the x and y axes.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			

Label Attributes

Text is used for identification, documentation, and so forth. A number of character fonts are supported.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
font	yes	string	`"euroStyle"` Eurostyle font
			`"gothic"` Gothic font
			`"math"` Mathematics font
			`"roman"` Times Roman font
			`"script"` Script font
			`"stick"` Stick font
			`"swedish"` Swedish font
hasTextDisplay	no	boolean	Indicates if the label has any associated text display.
height	yes	float	Height of the label (in user units).

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
isDrafted	yes	boolean	`t` if the label reads left to right when horizontal and bottom to top when vertical; the label is always readable. `nil` if the label can appear upside down and backwards if the orient is set properly.
isOverbar	yes	boolean	`t` if the label has a bar on top of the text. `nil` otherwise.
isVisible	yes	boolean	Label is visible if set to `t`.
justify	yes	string	`"lowerLeft"` lower left corner of the label
			`"lowerCenter"` middle of the bottom of the label
			`"lowerRight"` lower right corner of the label
			`"centerLeft"` center of the left side of the label
			`"centerCenter"` absolute center of the label
			`"centerRight"` center of the right side of the label
			`"upperLeft"` upper left corner of the label
			`"upperCenter"` middle of the top of the label
			`"upperRight"` upper right corner of the label
labelType	yes	string	`"normalLabel"` displayed without interpretation
			`"NLPLabel"` displayed with NLP interpretation
			`"ILLabel"` displayed with IL interpretation
objType	no	string	Type of object, in this case “label”

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
orient	yes	string	Orientation of this label:
			`"R0"` not rotated
			`"R90"` rotated 90 degrees
			`"R180"` rotated 180 degrees
			`"R270"` rotated 270 degrees
			`"MY"` mirrored on the y axis
			`"MYR90"` mirrored on the y axis and rotated 90 degrees counterclockwise
			`"MX"` mirrored on the x axis
			`"MXR90"` mirrored on the y axis and rotated 90 degrees counterclockwise
theLabel	yes	string	Label text
xy	yes	point	Location of the origin of a label (in user units).

Line Attributes

A connection of line segments. Unlike a path, a line has a fixed width, with no end and corner types or variable extensions.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
nPoints	no	integer	Number of points in the line.
points	yes	l_point	List of points for the line (in user units).

© 2020 All Rights Reserved.

Path Attributes

A path is a geometry defined by the coordinates of its center line. A path has width, end and corner types, and variable extensions.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
beginExt	yes	float	Extension (in user units) of the path beyond the first vertex.
endExt	yes	float	Extension (in user units) of the path beyond the last vertex.

pathStyle	yes	string	`"extendExtend"`
			`"roundRound"`
			`"truncateExtend"`
			`"varExtendExtend"`

points	yes	l_point	List of points for the path centerline (in user units).
width	yes	float	Width of the path (in user units).

© 2020 All Rights Reserved.

pathSeg Attributes

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
boundary	no	l_points	Boundary points of the pathSeg.
beginExt	yes	float	The extension of the beginning of the pathSeg in user units beyond the first vertex.
beginPt	yes	l_points	Begin point of the pathSeg.

beginStyle	yes	string	beginStyle of the pathSeg.
			`"truncate"`
			`"extend"`
			`"variable"`
			`"chamfer"`
			`"custom"`

endExt	yes	float	The extension of the end of the pathSeg in user units beyond the last vertex.
endPt	yes	l_points	End point of the pathSeg.

endStyle	yes	string	endStyle of the pathSeg.
			`"truncate"`
			`"extend"`
			`"variable"`
			`"chamfer"`
			`"custom"`

isOrthogonal	no	boolean	Boolean attribute which indicates if the pathSeg is orthogonal.
route	no	objId	ID of the route to which the pathSeg belongs.

topology	yes	string	`"none"`
			`"stripe"`
			`"standardCellWire"`
			`"IOWire"`
			`"ring"`
			`"padRing"`
			`"blockRing"`
			`"coreWire"`

width	yes	float	Width of the pathSeg.

© 2020 All Rights Reserved.

Polygon Attributes

The list of points describing a polygon, accessed by *polygon~>points*, is implicitly closed. This means that the path for a rectangular polygon contains four points, rather than the five points found in some other systems.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
nPoints	no	integer	Number of points in the polygon.
points	yes	l_point	List of points for the polygon (in user units).

Rectangle Attributes

Rectangles have parallel to the coordinate axes, a rectangle is compactly represented by its bounding box alone.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			

Text Display Attributes

Label display attributes on a per instance basis. The label text string to be displayed is defined in the instance's master label.

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes			
associate	no	objId	ID of the associate object.
font	yes	string	`"euroStyle"` Eurostyle font
			`"gothic"` Gothic font
			`"math"` Mathematics font
			`"roman"` Times Roman font

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"script"` Script font
			`"stick"` Stick font
			`"swedish"` Swedish font
height	yes	float	Height of the text display (in user units)
isDrafted	yes	boolean	`t` if the text display reads left to right when horizontal and bottom to top when vertical; the text display is always readable. `nil` if the text display can appear upside down and backwards if the orient is set properly.
isNameVisible	yes	boolean	Name portion is visible if set to `t`. Only applicable if associate is a property/attribute.
isOverbar	yes	boolean	`t` if the text display has a bar on top of the text, else `nil`.
isValueVisible	yes	boolean	Value portion is visible if set to `t`. Only applicable if associate is a property/attribute.
isVisible	yes	boolean	Text display is visible if set `to` *t.*
justify	yes	string	`"lowerLeft"` lower left corner of the label
			`"lowerCenter"` middle of the bottom of the label
			`"lowerRight"` lower right corner of the label
			`"centerLeft"` center of the left side of the label
			`"centerCenter"` absolute center of the label
			`"centerRight"` center of the right side of the label
			`"upperLeft"` upper left corner of the label

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
			`"upperCenter"` middle of the top of the label
			`"upperRight"` upper right corner of the label
orient	yes	string	Orientation of this label:
			`"R0"` not rotated
			`"R90"` rotated 90 degrees
			`"R180"` rotated 180 degrees
			`"R270"` rotated 270 degrees
			`"MY"` mirrored on the y axis
			`"MYR90"` mirrored on the y axis and rotated 90 degrees counterclockwise
			`"MX"` mirrored on the x axis
			`"MXR90"` mirrored on the y axis and rotated 90 degrees counterclockwise
owner	no	objId	ID of the owner of the text display object.
text	no	string	Text to be displayed.
xy	yes	point	Location of the origin of a text display (in user units).

© 2020 All Rights Reserved.

Technology File Attributes

The following is an example of retrieving the technology file ID from the current library and displaying technology file attributes.

```
libFileID = ddGetObj("myLibName")
techFileID = techGetTechFile(libFileID)
techFileID~>??
```

For information about SKILL functions to retrieve technology file information, see Chapter 1, Administrative SKILL Functions, in the *Virtuoso Technology Data SKILL Reference*.

© 2020 All Rights Reserved.

Generic Technology File Object Attributes

Attribute	Modify	Type	Description
groupMembers	no	l_objId	List of group members for this object
objType	no	string	The name of the object's type. The names are described in the attribute table for each object
prop	no	l_objId	List of properties on this object

Technology File Attributes

Attribute	Modify	Type	Description
createTime	no	string	Returns the time in which the technology file was created.
displays	no	l_objId	List of displays in the technology file
layers	no	l_objId	List of layers in the technology file
lib	no	objId	Library the technology file belongs to
maxPriority	no	integer	Maximum layer priority number
objType	no	string	Type of the object: `"techFile"`
purposes	no	l_objId	List of purposes in the technology file
timeStamp	yes	string	Time stamp of the technology file

Layer Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes			
blink	yes	boolean	Layer's blink attribute
brightColors	yes	l_objId	List of bright colors for the layer
displays	no	l_objId	List of displays that the layer references
fillColors	yes	l_objId	List of fill colors for the layer
fillStyle	yes	string	Layer fill style. One of the following:

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description

			`"outline"`
			Outline only and no fill
			`"solid"`
			A solid Fill
			`"X"`
			X-fill and outline
			`"stipple"`
			Stipple fill and no outline
			`"outlineStipple"`
			Stipple fill and outline

lineStyles	yes	l_objId	List of line styles for the layer
name	yes	string	Layer name
number	no	integer	Layer number
objType	no	string	Type of the object: `"layer"`
outlineColors	yes	l_objId	List of outline colors for the layer
priority	yes	integer	Layer's priority
purpose	no	string	Layer's purpose (see the purpose table) integer
selectable	yes	boolean	Layer's selectability
stipples	yes	l_objId	List of stipples for the layer
techFile	no	objId	Techfile the layer belongs to
twoLayers	no	l_objId	List of two layer relationships for the layer
visible	yes	boolean	Layer's visibility

© 2020 All Rights Reserved.

Display Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes			
colors	no	l_objId	List of colors in the display
lineStyles	no	l_objId	List of line styles in the display
maxColor	no	integer	Maximum number of colors allowed
maxLineStyle	no	integer	Maximum number of line styles allowed
maxStipple	no	integer	Maximum number of stipples allowed
name	yes	string	Display name
objType	no	string	Type of the object: `"display"`
stipples	no	l_objId	List of stipples in the display
techFile	no	objId	Techfile the layer belongs to

Color Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes			
index	no	integer	Color's index in the color table
name	yes	string	Color name
objType	no	string	Type of the object: "color"
rgb	yes	l_integer	A list that defines the red, green, and blue components of the color. The list is expressed as: `list( x_red x_green x_blue )`
			`list( x_red x_green x_blue )`
			The range of RGB values is between 0 and 1000.

© 2020 All Rights Reserved.

Stipple Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes			
height	no	integer	Derived stipple pattern height
index	no	integer	Stipple index in the stipple table
name	yes	string	Stipple name
objType	no	string	Type of the object: “stipple”
pattern	yes	see =>	List of a list of 0’s and 1’s that defines the stipple pattern. For example:
			list(list(0 0 0 0) list(0 1 1 0)
			`list(0 1 1 0) list(0 0 0 0) )`
width	no	integer	Derived stipple pattern width

Line Style Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes			
index	no	integer	Line styles’s index in the line style table
name	yes	string	Line style name
objType	no	string	Type of the object: “lineStyle”
pattern	yes	see =>	List of 0’s and 1’s that defines the stipple pattern. For example:
			list(0 0 1 1 0 0 1 1)
width	yes	integer	Line width in pixels

Two-Layer Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes			
layer1	no	objId	First of the two layers that are related

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
layer2	no	objId	Second of the two layers that are related
objType	no	string	Type of the object: “twoLayer”

© 2020 All Rights Reserved.

Reserved Values for Purpose

Purpose	Description
all	
annotate	
boundary	Object delineates a cell boundary
cell	Object is a block, no longer used
drawing	Object is a drawing
drawing1	
drawing2	
drawing3	
drawing4	
drawing5	
drawing6	
drawing7	
drawing8	
drawing9	
dynamic	Shapes are subject to automatic void shape maintenance
error	
flight	
label	
net	Object is associated with a net, no longer used
pin	Object is associated with a pin, no longer used
tool1	
tool0	
unknown	
warning	

© 2020 All Rights Reserved.

Reserved Values for Layers

Layer	Description
align	The mosaic alignment layer
annotate	The annotation layer
assemblyBoundary	A generic layer representing the boundary around a component. The boundary is needed to assemble the component onto a substrate.
axis	The axis layer
background	The background layer
beginGenericLayer	A generic layer representing the top metal layer of a board or substrate
border	Sheet borders/title blocks
changedLayer	The changed layer
designFlow	The design flow layer
device	Symbol graphics
drill	Represents drill shapes in package and PCB pad stacks
drive	The drive layer
edgeLayer	Edge assigned pins
endGenericLayer	A generic layer representing the bottom metal layer of a board or substrate
grid	The grid layer
hiz	The hiz layer
hilite	The highlighting layer
hardFence	Hard fence for compaction
instance	The instance layer
internalGenericLayer	A generic layer representing the internal metal layers of a board or substrate
marker	Error or warning markers
pin	Pin geometries in schematics or symbols
prBoundary	P&R boundary layer
resist	The resist layer

© 2020 All Rights Reserved.

Layer	Description
select	The selection layer
snap	The snap layer
solderMaskBottom	A generic layer representing the bottom passivation layer of a board or substrate
solderMaskTop	A generic layer representing the top passivation layer of a board or substrate
softFence	Soft fence for compaction
spike	The spike layer
stretch	Stretch control lines for parameterized cells
supply	The supply layer
text	Program annotative text/solid lines
unknown unset	Unknown layer
winActiveBanner	Window active banner
winAttentionText	Window attention text
winBackground	Window background
winBorder	Window border
winBottomShadow	Window bottom shadow
wirebond	A generic layer representing a wirebond connection from a die pad to a package
wirebondFingerGuide	A generic layer representing the locations where wirebond fingers should be placed
winButton	Window buttons
winError	Window error layer
winForeground	Window foreground
winInactiveBanner	Window inactive banner
winText	Window text
winTopShadow	Window top shadow
wire	Wires in schematics or symbols
y0	Waveform display and probing

© 2020 All Rights Reserved.

Layer	Description
y1	Waveform display and probing
y2	Waveform display and probing
y3	Waveform display and probing
y4	Waveform display and probing
y5	Waveform display and probing
y6	Waveform display and probing
y7	Waveform display and probing
y8	Waveform display and probing
y9	Waveform display and probing

© 2020 All Rights Reserved.

Via Spec Attributes

Attribute	Modify	Type	Description
layer1	no	objId	Database ID of the first layer of the via spec.
layer2	no	objId	Database ID of the second layer of the via spec.
defaultVal	yes	objId	Default viaDefs of the via spec.
viaSpecs	no	objId	All via specs defined in the technology file.

Via Def (standard) Attributes

Attribute	Modify	Type	Description
layer1	no	objId	Database ID of the first layer of the via def.
layer2	no	objId	Database ID of the second layer of the via def.
resistancePerCut	no	float	The resistance for a via in ohms per number of cuts.
libName	no	string	Returns the name of the library.
cellName	no	string	Returns the name of the cell.
viewName	no	string	Returns the name of the view.

Via Def (custom) Attributes

Attribute	Modify	Type	Description
layer1	no	objId	Database ID of the first layer of the via spec.
layer2	no	objId	Database ID of the second layer of the via spec.
resistancePerCut	no	float	The resistance for a via in ohms per number of cuts.
implant1	no	string	Returns the name of the first implant layer.
implant2	no	string	Returns the name of the second implant layer.
wellOrSubstrate	yes	string	Returns the name of the well or substrate layer.

© 2020 All Rights Reserved.

Attribute	Modify	Type	Description
params	no	l_params	List of via parameters.

SiteDef Attributes

Attribute	Modify	Type	Description
height	no	float	Height of the siteDef.
`name`	no	string	Name of the siteDef.
symmetricInX	yes	boolean	Indicates the symmetry in the X direction. `t` allows the instance to be placed as R0 or MX.
symmetricInY	yes	boolean	Indicates the symmetry in the Y direction. `t` allows the instance to be placed as R0 or MY.
symmetricInR90	yes	boolean	Indicates the 90 degree rotation. `t` allows the instance to be placed at a 90 degrees.
techfile	no	objId	Technology file ID that is associated with the siteDef.
type	no	string	Type of siteDef.
width	no	float	Width of the siteDef.

```
libFileID = ddGetObj("myLibName")
techFileID = techGetTechFile(libFileID)
techFileID~> siteDefs>??
```


© 2020 All Rights Reserved.

20

Design Management Functions

This chapter contains the following sections:

- Library Organization
- Attributes of Data Management Objects
- Triggering Function Calls
- Design Management Functions
- Design Data Services
- dmb Compatibility Functions
- Cell Category Functions (ddCat Interface)
- Old Cell Category Functions
- Design Editor SKILL Functions
- Registering DE Plugins
- Client Interface
- Licensing Functions
- Design Editor Support Routines
- Extended Design Editor Example

© 2020 All Rights Reserved.

Library Organization

Through data management you impose a layer of abstraction on a physical file system that lets you organize and control the physical file system in the following ways:

A design library is organized into four levels, as shown in Figure 20-1.

Figure 20-1 Library Organization

In addition, all objects below the library have a relation to the library.

© 2020 All Rights Reserved.

Version Control

You can perform version control with the design management system you are using. The library structure is independent of any design management system. You are free to use

- The basic design management functionality
- The advanced team design manager product from Cadence
- Any other system you choose
- No design management system

Concurrence Control

Data management arbitrates between users who want to do incompatible operations at the same time. The system permits maximum sharing with a minimum of conflict.

© 2020 All Rights Reserved.

Attributes of Data Management Objects

The attributes of data management objects use the following operators to access data management objects with SKILL:

(~>)	To access any attribute or property of an object
(~>?)	To see names of all attributes of an object
(~>??)	To see names and values of all attributes and properties of an object

If you ask for an attribute that is not applicable for the database object or for a property that does not exist on the object, the system returns `nil`.

In these figures, types of data returned by an attribute are shown as

b	a DDPI object
d	an identifier
f	a floating point number
g	any of the data types
l	a list
n	a number, either integer or floating point
s	a symbol
t	a character string
x	an integer

When IL is present, the system provides “~>”(squiggle-arrow) support for all objects in the new data model. Table 20-1 on page 1962 and Table 20-2 on page 1963 show the names of the attributes and relations you can get at via squiggle-arrow.

Table 20-1 Attribute Names Provided for “Object~><name>” Access

Attribute Names	Object Types Attribute is Valid For		
group	libFile	cellFile	viewFile
groupAccess	libFile	cellFile	viewFile
isReadable	libFile	cellFile	viewFile

© 2020 All Rights Reserved.

Attribute Names	Object Types Attribute is Valid For					
isWritable				libFile	cellFile	viewFile
lastModify				libFile	cellFile	viewFile
name	lib	cell	view	libFile	cellFile	viewFile
owner				libFile	cellFile	viewFile
ownerAccess				libFile	cellFile	viewFile
prop	lib	cell	view			
publicAccess				libFile	cellFile	viewFile
readPath				libFile	cellFile	viewFile
techfile	lib	cell	view	libFile	cellFile	viewFile
type	lib	cell	view	libFile	cellFile	viewFile
writePath				libFile	cellFile	viewFile

Table 20-2 Relation Names Provided for “Object~><name>” Access

Relation names	Relationship	Object Types Attributes is Valid For					
cell	1:1			view		cellFile	
cells	1:n	lib					
children	1:n	lib	cell	view			
files	1:n	lib	cell	view			
lib	1:1	lib	cell	view	libFile	cellFile	viewFile
parent	1:1	lib	cell	view	libFile	cellFile	viewFile
view	1:1						viewFile
views	1:n		cell				

© 2020 All Rights Reserved.

Triggering Function Calls

Application programs need to react when certain changes occur to library objects. The DFII software lets you register a SKILL function to be called (triggered) when certain changes either have just occurred or are about to occur for library objects.

Caution

Do not call functions from within the registered functions that might cause an infinite loop. The trigger code that calls back to the DFII environment can have serious side effects. The only operations done by a trigger such as PreObjAccess or PreAutoCheckout should be calls to services outside of the DFII environment. The capability described here is only an enabling feature; Cadence cannot take responsibility to debug any user trigger code unless the problem lies within the DFII environment that is enabling the integration.

Setting the Trigger

Use the registration function, `ddRegTrigger`, to register a function to be called for any kind of triggering event. Figure 20-2 on page 1965 lists the types of triggering events for which you can register a function. Pass `ddRegTrigger`

- Name of the SKILL function to register
- Type of event that triggers a call to the function

For example, to register the function `exampleCheckinCallback` to be called when any cellview version is checked in,

```
ddRegTrigger( "PostCheckin" 'exampleCheckinCallback )
```

Multiple Triggers for the Same Event

You can register many functions to be called when an event occurs. When several functions are registered for the same event, you can set the order called using the priority argument of `ddRegTrigger`. Triggers that have lower priorities are called back before triggers that have higher priorities. Each function registered for a single triggering event must have a unique name. If you try to register a function with the same name as one already registered, the registration function returns `nil` and the original function remains registered.

Having unnecessary functions registered can slow down performance. Un-register functions that are no longer needed. The `ddUnregTrigger` function un-registers a trigger.

© 2020 All Rights Reserved.

Setting Pre-triggers and Post-triggers

A trigger that calls registered functions before the object is changed is called a pre-trigger. A trigger that calls the registered functions after the object is changed is called a post-trigger.

Pre-triggers are created to prepare information or the user environment and are allowed to cause the function to fail, preventing execution of the body of the function. Post-triggers may not cause the function to fail and are expected to handle cleanup of user data or other bookkeeping tasks.

Functions you attach to a pre-trigger are passed the same arguments passed to the associated PI function. Functions that you attach to a pre-trigger should be written to return either `t` or `nil`. If the value is `nil`, no further trigger functions are called and the associated PI function is canceled.

Functions you attach to a post-trigger are passed the same arguments passed to the associated PI function. In addition, the last argument is the value returned by the PI function. Post-trigger returns are ignored with the exception of the First Access Lib Trigger, which must return `t` or `nil`.

Trigger Names Allowed

Each of the PI functions has a pre-trigger and a post-trigger. The following trigger names are allowed:

Figure 20-2 PI Function Trigger Names

PI function	Pre-trigger name	Post-trigger name
ddUpdateLibList()	“PreUpdateLibList”	“PostUpdateLibList”
ddGetStartup()	“PreGetStartup”	“PostGetStartup”
ddDeleteObj() ddDeleteLocal()	“PreDeleteObj”	“PostDeleteObj”
ddCreateLib()	“PreCreateLib”	“PostCreateLib”
ddGetObj()	“PreCreateObj”	’PostCreateObj” “FirstAccessLib” *
ddCheckin()	“PreCheckin”	’PostCheckin”
ddCheckout()	“PreCheckout”	“PostCheckout”
ddGetObj()	“PreObjAccess”	

© 2020 All Rights Reserved.

PI function	Pre-trigger name	Post-trigger name
ddAutoCheckout()	“PreAutoCheckout”	

* `ddGetObj` calls the “FirstAccessLib” trigger the first time it accesses a library in the library list.

Note: The names are case insensitive.

Adding a PreObjAccess Trigger to ddGetObj

The ObjAccess trigger is a pre-trigger invoked to allow preparation of the user environment. It is invoked each time any object is accessed.

Caution

The PreObjAccess should avoid calling ddGetObj() on any object because it would be recursive and dangerous.

Registering a PreObjAccess Trigger

To register a trigger to be called by `ddGetObj` whenever access is requested for an object, use the following code:

```
ddRegTrigger( "PreObjAccess" 'exampleObjAccessTrigger )
```

where *`exampleObjAccessTrigger`* is the SKILL function that the user implements to perform the desired `PreObjAccess` operations.

The PreObjAccess trigger is called with the following arguments: *`t_libName`*, *`t_cellName`*, *`t_viewName`*, *`t_fileName`*, *`b_contextid`*, and *`t_mode`*. For information on these arguments, see ddGetObj. The lib, cell, view, and file name arguments are in the name space of the calling application. The PreObjAccess trigger may need to map these to the file system name space by using the `nmpCDBAToFilesys` function. The PreObjAccess trigger must also resolve the name arguments with respect to any `contextId` argument passed to the `ddGetObj` call.

Note: The PreObjAccess trigger must return a Boolean value. If the return from the trigger is `nil`, the associated `ddGetObj` function is canceled. The PreObjAccess trigger code should avoid calling `ddGetObj()` itself on any object. You should avoid calling the function recursively as it will cause debugging problems.

© 2020 All Rights Reserved.

Adding a PreAutoCheckout Trigger to ddAutoCheckout

The PreAutoCheckout trigger is a pre-trigger invoked to allow preparation of the user environment. It is invoked each time any object is accessed.

Registering a PreAutoCheckout Trigger

To register a trigger to be called by `ddAutoCheckout` whenever checkouts are requested for an object, use the following code:

```
ddRegTrigger( "PreAutoCheckout" 'exampleAutoCheckoutTrigger )
```

where `exampleAutoCheckoutTrigger` is the SKILL function that the user implements to perform the desired PreAutoCheckout operations.

The PreAutoCheckout trigger is called with the following argument: *`l_ddId`* which is a list of the ddIds associated with the files that are being checked out. See ddAutoCheckout for details on this argument.

The PreAutoCheckout trigger must return a Boolean value. If the return from the trigger is `nil`, the associated `ddAutoCheckout` function is canceled.

The `ddAutoCheckout` function supports triggers. PreAutoCheckout is a trigger. The DDPI invokes the `ddAutoCheckout` pre-trigger in this order:

- Verifies that the `ddAutoCheckout` request includes at least one file to check out and returns `nil` if there are any initialization problems
- Verifies that the user has set the `CDS_PROMPT_CKOUT` flag
- Verifies that the requested files for membership are in a co-managed set. If you request a checkout of a member of a co-managed set (for example, a view file), the ddId of the requested file is replaced with the ddId of its parent object (for example, the ddId of the view)
- Calls the `ddAutoCheckout` pre-trigger
- Verifies if the checkout is needed for each ddId
- Executes the prompt as requested for each ddId
- Performs the checkout if needed for each ddId
- Returns an array of boolean values, matching the ddId of the objects passed in, showing `TRUE` if the object was successfully checked out, or did not need to be checked out, and `FALSE` if the checkout failed, or the user indicated no desire for checkout.

© 2020 All Rights Reserved.

Design Management Functions

ddAmUsingDM

```
ddAmUsingDM(
     b_objId
     )
     => t / nil
```

Description

Returns `t` if the library containing *`b_objId`* uses a Design Management system.

Arguments

`b_objId`	ID of an object.

Value Returned

`t`	The library containing *`b_objId`* uses a Design Management system.
`nil`	Object was not found or is not within a library under Design Management control.

Example

```
ddAmUsingDM( myViewFile ) => t
```


© 2020 All Rights Reserved.

ddAutoCheckin

```
ddAutoCheckin(
     l_ddIds
     [ g_isBatch( { t | nil } ) ]
     )
     => l_list
```

Description

Automatically checks in an object or list of objects. Objects in a temporary directory cannot be managed. The function skips objects that exist only in the temporary directory of the library and displays a warning for them.

The function:

- Prints an info message to confirm that a checkin was performed successfully.
- Generates a warning if a file set to be checked in was not checked in.
- Prints a message or displays a message box if a user aborts the checkin via the UI or the checkin is skipped. When the checkin of an object is skipped, you will no longer be prompted to check it in. The object will have to be manually checked in or canceled using library manager when appropriate.

`ddAutoCheckin` automatically prompts for all check-ins if a registered UI is in place, a specified file requires checkin, and the `CDS_PROMPT_CKIN` flag has been set. A file is deemed as needing checkin if it was checked out in this session of this binary and has not yet been checked in.

Arguments

l_ddIds	A ddId for a single object or a list of ddIds, which are the objects that should be checked in, if necessary. These objects can be a cellview ID, a file within a cellview ID, or a file at either the library or cell level. The files can be from more than one library.
g_isBatch	The batch override. Set to `t` if the application wants to force being treated as a batch application, irrespective of whether there is a registered UI or not. Can be set to `nil` by virtually all applications.

© 2020 All Rights Reserved.

Value Returned

l_list	A list of elements, all being `t` or `nil`, corresponding to each ID being checked in. The matching index will be set to `TRUE` if the checkin was done successfully or if none was needed or if the user rejected the checkin via the UI. The matching index will be set to `FALSE` if the checkin failed.

© 2020 All Rights Reserved.

ddAutoCheckout

```
ddAutoCheckout(
     l_ddIds
     [ g_isBatch( { t | nil } ) ]
     )
     => l_list
```

Description

Automatically checks out an object or list of objects. Objects in a temporary directory cannot be managed; this function skips objects that exist only in the library's temporary directory and displays a warning for them.

Arguments

l_ddIds	A ddId for a single object or a list of ddIds, which are the objects that should be checked out, if necessary. These objects can be a cellview ID, a file within a cellview ID, or a file at either the library or cell level. The files can be from more than one library.
g_isBatch	The batch override. Set to `TRUE` if the application wants to force being treated as a batch application irrespective of whether there is a registered UI or not. Can be set to `FALSE` by virtually all applications. It will be used by the SKILL "~>" operator in the case of setting a `propBag` parameter in a bag that has not been checked out.

Value Returned

l_list	A list of elements, all being `t` or `nil`, corresponding to each ID being checked out. The matching index will be set to `TRUE` if the checkout was done successfully or if none was needed. The matching index will be set to `FALSE` if either the checkout failed or a checkout was needed but the user refused to do it via the registered UI.

`ddAutoCheckout` prints a message if a checkout was performed stating what was checked out. `ddAutoCheckout` generates a warning if a file needed to be checked out but could not be. `ddAutoCheckout` does not give a message at all if either the user aborted the checkout via the UI or no checkout was needed.

© 2020 All Rights Reserved.

`ddAutoCheckout` automatically prompts for all checkouts if a registered UI is in place, a specified file requires checkout, and the `CDS_PROMPT_CKOUT` flag has been set.

© 2020 All Rights Reserved.

ddAutoCtlGetVars

```
ddAutoCtlGetVars(
     )
     => l_settings
```

Description

Returns a list of four elements in the following order:

x_prompCkout
x_promptCkin
x_AutoCkout
x_AutoCkin

Each element is a value from `0` through `3` as in `ddAutoCtlSetVars`. Each element can have one of four values: `0` - none, `1` - views, `2` - files, `3` - both views and files.

Arguments

None

Value Returned

Returns a list of four elements in the following order:

x_promptCkout	Prompt for check out for views or files.
x_promptCkin	Prompt for check in for views or files.
x_AutoCkout	Automatically check out views or files.
x_AutoCkin	Automatically check in views or files.

Example

```
ddAutoCtlGetVars() => (0 0 3 1)
```

Do not prompt for check out for any view or file, do not prompt for check in for any view or file, automatically checkout for both views and files, automatically check in views only

© 2020 All Rights Reserved.

ddAutoCtlSetVars

```
ddAutoCtlSetVars(
     x_promptCkout
     x_promptCkin
     x_AutoCkout
     x_AutoCkin
     )
     => t
```

Description

Sets prompting and automatic check-in and check-out controls for files and views. Takes a list of four integers, each of which can be one of four values: `0` - none, `1` - views, `2` - files, `3` - both views and files. The integers are applied to indicate to the system to prompt for check out, prompt for check in, automatically check out, and automatically check in respectively.

Arguments

x_promptCkout	Prompts check out for views or files.
x_promptCkin	Prompts check in for views or files.
x_AutoCkout	Automatically checks out views or files.
x_AutoCkin	Automatically checks in views or files.

Note: Each element is a value from `0` through `3` as mentioned above.

Value Returned

`t`	Sets the matching index to the values specified.

Examples

Example 1

```
ddAutoCtlSetVars (3 3 3 3)
```

Turns on prompt check out, prompt check in, automatic check out, and automatic check in for views and files as shown in Example 1.

© 2020 All Rights Reserved.

Example 2

```
ddAutoCtlSetVars (0 0 3 3)
```

Alternatively, you can also turn off prompt check out or prompt check in for files and views, but turn on automatic check out and automatic check in for files and views as shown in this example.

Example 3

```
ddAutoCtlSetVars (0 0 3 0)
```

Or, you can also turn off prompt check out and prompt check in for files and views, turn on automatic check out of files and views, but, turn off automatic check in of files and views as shown in this example. The 4.3.4 version of Design Framework II defaults to this setting.

© 2020 All Rights Reserved.

ddCatRefresh

```
ddCatRefresh(
     b_catFiles
     )
     => t / nil
```

Description

Given the DD ID of a category file, this function will purge the in-memory copy of the file and read it in again from disk. This will update the in-memory copy. Refresh is allowed only if the category was opened in read mode.

Arguments

b_catFile	The DD ID oAI665f a category file.

Value Returned

`t`	The in-memory copy is updateAI665d.
`nil`	Execution of the function failed.

Example

```
catId = ddCatOpen(libId, "mycat.Cat" "r")
ddCatRefresh(catId)
```


© 2020 All Rights Reserved.

ddCheckin

```
ddCheckin(
     g_files
     t_description
     [ t_switches ]
     )
     => t / nil
```

Description

Checks in the new version of the checked-out and modified data files. Builds a system command that invokes the `gdmci` command. `ddCheckin` makes it possible to integrate Cadence tools with other design management software.

Arguments

g_files	A single ddId or a list of ddIds of files to be checked in. The files can be from more than one library.
t_description	Description of what you changed after you checked out the file.
t_switches	Switches to pass to the `gdmci` command. The `c` switch cancels the *checkout* of the named files. Valid Value: `c`

Value Returned

`t`	The checkin succeeded.
`nil`	Execution of the function failed.

Example

```
ddCheckin( viewId "fixed short" ) => t
```


© 2020 All Rights Reserved.

ddCheckout

```
ddCheckout(
     g_files
     [ t_switches ]
     )
     => t / nil
```

Description

Checks out a version of the files. No version of a cellview can be checked out by any user until any version that is currently checked out is checked in. Builds a system command that invokes the `gdmco` command. `ddCheckout` makes it possible to integrate Cadence tools with other design management software.

Arguments

g_files	A single ddID or a list of ddIDs of files to be checked out. The files can be from more than one library.
t_switches	Checks out files as read-only (`r`) or specifies a particular version of the file to check out (`v`). You must always follow this switch with the version of the file to check out.

Value Returned

`t`	The checkout succeeded.
`nil`	Execution of the function failed.

Examples

```
ddCheckout( viewId "rv1.1" ) => t
ddCheckout( filesList ) => t
```


© 2020 All Rights Reserved.

ddClearLibUnmanaged

```
ddClearLibUnmanaged(
     b_libId
     )
     => t / nil
```

Description

Removes any DMTYPE entry from the `cdsinfo.tag` file in the library directory. This will allow the default value for DMTYPE to be used for this library.

For more information about the `cdsinfo.tag` file, see the *Cadence Application Infrastructure User Guide*.

Also see `ddSetLibUnmanaged`, `ddIsPathManagable`, `ddIsPathManaged`, `ddAmUsingDm`, `ddCheckin`.

Arguments

b_libId	The ID of a library object.

Value Returned

`t`	The DMTYPE is removed.
`nil`	The DMTYPE is not removed.

© 2020 All Rights Reserved.

ddCreateLib

```
ddCreateLib(
     t_libName
     [ t_libPath ]
     )
     => b_ddId
```

Description

Searches for the named library in the library list.

If the library does not exist, `ddCreateLib` creates the corresponding directory and adds the `DEFINE` command to the library list file (it uses `ddUpdateLibList`'s search to find the `cds.lib` file and, if that file is not in the current working directory, it creates a new `cds.lib` file in the current working directory and adds the `DEFINE` statement to it as well as a `SOFTINCLUDE` statement to include the file found by `ddUpdateLibList`'s search). Finally, if successful, it returns the ID of the new library.

If you do not include *`t_libPath`*, DDPI uses the current working directory and *`libName`* to make a path (*`cwd/libName`*). If you specify a leaf name (one without embedded separators), DDPI makes a path out of that name and the current working directory (*`cwd/libPath`*). If you specify "." or "..", DDPI does the same as for NULL.

The `cdsinfo.tag` file keeps track of which design management system is managing a library. This file resides in the library directory. It contains one string, which is the name of the design management system.

`ddCreateLib` calls back trigger functions registered on the triggers `PreCreateLib` and `PostCreateLib`. The PostCreateObj trigger is not called when a library is created with ddCreateLib.

Arguments

`t_libName`	Name of the library to create.
`t_libPath`	Absolute or relative directory name where the library is created. A relative path is interpreted relative to the current directory. If *`t_libPath`* is omitted, `nil`, or an empty string (" "), the library is created in the current working directory.

© 2020 All Rights Reserved.

Value Returned

b_ddId	The ID of the library.

Example

```
TTL7400libraryId = ddCreateLib( "TTL7400_series" )
```


© 2020 All Rights Reserved.

ddDeleteObj

```
ddDeleteObj(
     b_objectId
     )
     => t / nil
```

Description

Deletes *b_objectId* from the disk.

If *b_objectId* is a library, this function deletes the library directory from the disk and removes the library entry from the `cds.lib` file. If the library has a temporary directory associated with it, `ddDeleteObj` adds an `UNASSIGN` statement to the `cds.lib` file to unassign the `TMP` attribute from the library; it does not delete the temporary directory from the disk. Also, if the library has been defined in more than one `cds.lib`, `ddDeleteObj` does not delete the library from the disk; instead, it adds an `UNDEFINE` statement for the library to the appropriate `cds.lib` file.

If *b_objectID* is of any other type, `ddDeleteObj` removes the object from disk from either the master directory or the temporary directory, depending on which one it exists in, or from both, if it exists in both.

If the object's library is under DM control, `ddDeleteObj` will call `gdmdelete` to do the deletion. `gdmdelete` will call the appropriate DM system's calls.

If you delete a directory object, `ddDeleteObj` tries to get an exclusive lock, in non-blocking mode, on every data file in the hierarchy under that directory. If it fails to get any lock, it knows that another process is using that file. It therefore cannot delete the object, and it fails.

`ddDeleteObj` has two triggers: PreDeleteObj and PostDeleteObj. The `objectId`'s of deleted objects in virtual memory remain valid until after the return from PostDeleteObj triggers, so you can use PI functions (for cleanup) that require an ID. If you call `ddDeleteObj` on a directory object that contains children, `ddDeleteObj` calls the pre- and post-triggers once for each of the child objects that has an ID in virtual memory.

© 2020 All Rights Reserved.

Arguments

b_objectId	Object identifier returned by `ddCreateLib` or `ddGetObj`.

Value Returned

`t`	Execution of the function succeeded.
`nil`	Execution of the function failed.

Example

```
ddDeleteObj( TTL7400libraryId ) => t
```


© 2020 All Rights Reserved.

ddDeleteLocal

```
ddDeleteLocal(
     b_objectId
     )
     => t / nil
```

Description

Removes `b_objectId` from the local work area only. If the objects are under design management, they are not removed from the repository.

`ddDeleteLocal` has two triggers: PreDeleteObj and PostDeleteObj. The object IDs of deleted objects in virtual memory remain valid until after the return from PostDeleteObj triggers, so you can use PI functions (for cleanup) that require an ID. If you call `ddDeleteLocal` on a directory object that contains children, `ddDeleteLocal` calls the pre- and post-triggers once for each of the child objects that has an ID in virtual memory.

Arguments

b_objectId	Object identifier returned by `ddCreateLib` or `ddGetObj`.

Value Returned

`t`	Execution of the function succeeded.
`nil`	Execution of the function failed.

Example

```
ddDeleteLocal( TTL7400libraryId ) => t
```


© 2020 All Rights Reserved.

ddDumpLibList

```
ddDumpLibList(
     )
     => t
```

Description

Prints information about the library list. It first prints the list of library files (`cds.lib` and `lib.defs` files) and then prints information about each library in the library list. This includes the path to the library and the path to `TMP` directories, if any.

By default, information about the libraries is printed in the order in which the libraries appear in the `cds.lib` files, unless the environment variable `DD_GEN_ORDERED_LIBLIST` is set to `false`, in which case the order is random.

Arguments

None

Value Returned

`t`	Information about the library list is printed.

Example

If you have the following `cds.lib`:

```
/tmp/TESTDIR/cds.lib
```

which has the following contents:

```
DEFINE LibA ./LibA
DEFINE LibB ./LibB
```

`ddDumpLibList` prints the following information:

```
Lib Files:
1: /tmp/TESTDIR/cds.lib

Libraries defined:
1: LibA from file /tmp/TESTDIR/cds.lib (refCount 1).
        Path: /tmp/TESTDIR/LibA
2: LibB from file /tmp/TESTDIR/cds.lib (refCount 1).
        Path: /tmp/TESTDIR/LibB
```

where `refCount` is the number of `cds.lib` files that contain that library definition.

© 2020 All Rights Reserved.

ddGetCoManagedSet

```
ddGetCoManagedSet(
     g_viewID
     )
     => l_coManagedList / nil
```

Description

Returns the list of comanaged files for the specified view.

Arguments

g_viewID	A *viewId* returned by ddGetObj.

Value Returned

l_coManagedList	The list of comanaged files for the specified view.
nil	There was an error.

Example

The following:

```
viewfileId=ddGetObj("mylib" "mycell" "schematic" "sch.oa" nil "w")
viewId=ddGetObj("mylib" "mycell" "schematic")
list=ddGetCoManagedSet(viewId)
```

returns

© 2020 All Rights Reserved.

ddGetCombineValue

```
ddGetCombineValue(
     b_libID
     )
     => l_libNames / nil
```

Description

Returns a list of names of libraries in a combined library, if the library specified has the `COMBINE` attribute assigned.

For more information, see *Setting Attributes on a Library* and *Creating Combined Libraries* in the *Cadence Library Manager User Guide*.

Arguments

g_libID	The ID of the combined library.

Value Returned

l_libNames	A list of names of libraries in the combined library, if the library specified has the `COMBINE` attribute assigned.
`nil`	The library was not found or did not have the `COMBINE` attribute assigned.

Example

```
when( libNames=ddGetCombineValue(libId)
  printf("'%s' combines libraries %L\n" libId~>name libNames)
)
```


© 2020 All Rights Reserved.

ddGetDisplayValue

```
ddGetDisplayValue(
     b_libID
     )
     => t_attrName / nil
```

Description

Returns the value of the `DISPLAY` attribute assigned to a library.

For more information, see *Setting Attributes on a Library* in the *Cadence Library Manager User Guide*.

Arguments

g_libID	The ID of the library.

Value Returned

t_attrName	The value of the `DISPLAY` attribute.
`nil`	The library was not found or did not have the `DISPLAY` attribute assigned.

Example

```
when( attrName=ddGetDisplayValue(libId)
  printf("Library '%s' has DISPLAY attribute '%s' assigned\n" libId->name attrName)
)
```


© 2020 All Rights Reserved.

ddGetForcedLib

```
ddGetForcedLib()
     => t_filePath / nil
```

Description

Returns the path to the `cds.lib` file that was set by using either the `-cdslib` argument to the executable or by using `ddSetForcedLib`.

Arguments

None

Value Returned

`t_filePath`	The path to the `cds.lib` file or empty string if no forced cds.lib file was specified.
`nil`	The path to the `cds.lib` file is not found.

Example

```
ddGetForcedLib() => "/user/library/cds.lib"
```

Reference

ddGetForcedLibEx, ddSetForcedLib, ddSetForcedLibEx

© 2020 All Rights Reserved.

ddGetForcedLibEx

```
ddGetForcedLibEx(
     g_boolean
     )
     => t_filePath
```

Description

Gets the `cds.lib` file that was set with the `ddSetForcedLib` or `ddSetForcedLibEx` functions or with the `-cdslib` command line option.

Note: The `lib.defs` file is no longer supported by Virtuoso and cannot be set as a forced library definition file using `ddSetForcedLibEx` or `-libdefs`. This function will return empty string if `t` is passed to get the forced `lib.defs` file.

This function is deprecated. Use ddGetForcedLib, that can be used to get only the cds.lib file that is set.

Arguments

g_boolean	`t` or `nil`. The `t` option is deprecated. Pass `nil` to get the `cds.lib` file that is set as a forced file.

Value Returned

t_filePath	The path to the `cds.lib` file that is set (if *g_boolean* is `nil`), or an empty string (`""`) if the file is not set or if the deprecated `t` option was passed.

Example

```
ddGetForcedLibEx(nil)
```
returns `"/usr1/mnt4/cds.lib"` or `""` if it was not set.

Reference

ddSetForcedLib, ddSetForcedLibEx

© 2020 All Rights Reserved.

ddGetLibList

```
ddGetLibList(
     )
     => l_libs/ nil
```

Description

Generates a list of the libraries in the `libList`.

Arguments

None

Value Returned

`l_libs`	A list with one element for each library in the `libList`. Each element of *`l_libs`* is the ddId of a library.
`nil`	Execution of the function failed.

Examples

```
ddGetLibList( ) => (dd:2543240 dd:2611120 )
(ddGetLibList) ~> name => ("cdsDefTechLib" "l3")
```


© 2020 All Rights Reserved.

ddGetObj

```
ddGetObj(
     { t_libName / nil }
     [ t_cellName / nil ]
     [ t_viewName / nil ]
     [ t_fileName / nil ]
     [ b_contextId / nil ]
     [ t_mode ]
     )
     => b_ddId / nil
```

Description

Finds libraries, cells, views, and files, and creates cells, views, and files (you use `ddCreateLib` to create libraries).

`ddGetObj` is re-entrant; you can call if from inside a PreCreateObj or a PostCreateObj trigger function.

Any time you create a new placeholder file using `ddGetObj` and do not modify the contents, you should pair that call with a call to `ddReleaseObj` to check for and clean up the zero-size placeholder file.

If the file you want is a CDB cellview, do not use `ddGetObj`. Use `dbOpenCellViewByType` (see `dbOpenCellViewByType`), which will call `ddGetObj` for you.

`ddGetObj` can call triggers on FirstAccessLib, PreCreateObj, PostCreateObj, and PreObjAccess.

Triggers	Time invoked
FirstAccessLib	A post-trigger invoked the first time you access any object in each library. It gets invoked once per library (unless `ddUpdateLibList` changes the library's path). Posttriggers may not cause the function to fail and are expected to handle cleanup of user data or other bookkeeping tasks.
PreCreateObj	Invoked just before DDPI creates an object.
PostCreateObj	Invoked just after DDPI creates an object. The trigger will not be called when a cell is re-created or overwritten, or if a new library is being created.
PreObjAccess	A pre-trigger invoked to allow preparation of the user environment. It is invoked each time any object is accessed.

© 2020 All Rights Reserved.

DDPI defines `ddGetObj`'s behavior using the following concepts:

- A master file is a file that the writing application believes to contain source data or data that cannot be recreated without user input. There can be only one master file per view. The master file may be identified by one of the following mechanisms:
  - It is the only file belonging to the library, cell, or view.
  - There is a file belonging to the view named `master.tag` containing the name of the master file.

  Note: PostCreateObj trigger does not get called for creating `master.tag` file.
- A co-master file is a file that the writing application believes to contain source data or data that cannot be recreated without user input, but it is useless without the master. For example, an index for the master file; sheets of a multi-sheet schematic, where the index is the master; or separation of graphics and property data into separate files.
- If a file is not a master or a co-master, it is a derived file. Derived files can be recreated from master files, or master and co-master files, in the same view. Only the creating tool can distinguish between masters or co-masters and derived files.
  - To access a derived file correctly, the tool must specify *`b_contextId`* for the master. If writing or appending to the file, use the "*d*" modifier in *`t_mode`*. Use `ddMarkObjMaster` to change the master file in a view.
- A file's relative name is specified relative to a library. The name consists of [/*`cellName`*[/*`viewName`*]]/*`fileName`*. Files can belong to the library, cell, or view.
  - `ddGetObj` searches for the object under your specified library until it finds the first occurrence of the relative named object. If you requested read access, the function is done and returns an ID. It is an error if the tool does not have read access to that object.
  - If you request any other access mode, `ddGetObj` first searches the object in read mode, then proceeds differently depending upon whether or not the file exists.
- If a library has a temporary directory (defined by putting a `TMP` attribute on a library with an `ASSIGN` statement in a `cds.lib` file), the following rules apply:
  - While reading data, `ddGetObj` looks for the data in the temporary directory first. If it is not found in the temporary directory, it looks for it in the library (the library is also referred to as the master directory).
  - `ddGetObj` always saves or creates master or co-master files in the library, regardless of whether a temporary directory exists. This is because a temporary directory can only contain derived data, it cannot contain source data.

© 2020 All Rights Reserved.

- `ddGetObj` always saves or creates derived files in the temporary directory. It creates cell or view directories as needed. A temporary directory has the same directory structure (library/cell/view) as the library.

`ddGetObj` takes string names instead of ddIds as the input for its query because you use it to locate objects when you do not yet have their ddIds. You can supply a context object instead of specifying some of the names. DDPI takes the missing names from the context object for the query.

Arguments

t_libName	The library name to find the object you want. The named library must appear in the `libList`. If you do not specify *t_libName* (by specifying `nil` or `""`) but you do specify *b_contextId*, DDPI takes the `libName` from the context object.
t_cellName	The name of a cell in the path to the object you are looking for. `ddGetObj` looks for the cell in *t_libName*. If you do not specify *t_cellName* (by specifying `nil` or `""`) but you specify *b_contextId*, DDPI takes the cellname from the context object.
t_viewName	The name of a view in the path to the object you are looking for. `ddGetObj` looks for the view in *t_libName*, *t_cellName*. If you do not specify *t_viewName*, but you do specify *b_contextId*, DDPI takes the *viewName* from the context object. **Note:** *t_viewName* may be a space-separated list of view names. If you use this option, then each time DDPI visits a place to look for an object, it looks for one with successive view names from the list. DDPI returns successfully if it finds a match with any of the view names. Thus you can prioritize which view to look for first by ordering the view list. DDPI checks all the views in each place. It does not repeat the entire search for each view.
t_fileName	The name of the file object that you are looking for. `ddGetObj` looks for the file in *t_libname* only.

© 2020 All Rights Reserved.

For the convenience of applications like the design editor, `de`, (which need to find a master file without knowing the file's name), you can specify an asterisk (*) that opens the master file belonging to the view. This feature works only for view files. In addition, if you know the file's name and its extension or a pattern of trailing characters, you can specify the filename as `*pattern`, for example, `*arch.vhd`, and it will return the master file that matches that pattern.

b_contextId

You can specify a `contextId` object to take the library, cell, and/or view names from. This parameter provides way to find an object that is in the same directory or has the same parents as some other object.

t_mode

A string representing the access mode desired. Controls side effects of `ddGetObj`. Valid values:

- `r`
  Gets the ID of an existing object in the specified library. If the library has a temporary directory, looks in the temporary directory first; if the object is not found in the temporary directory, looks in the library (the master directory)

- `w`
  Gets the ID of a new object to be created. This option is for creating master data. If the object is the first file in a view, it is marked as the master object. This will fail if a master object already exists in the specified view. The object is created in the master directory regardless of whether the library has a temporary directory and whether the object already exists in the temporary directory.

- `wc`
  This option is for creating co-master data and is similar to `w` except that if the object is a view file, it is created without marking it as a master object. This will succeed even if there is a master object in the view. The object is created in the master directory regardless of whether the library has a temporary directory and whether the object already exists in the temporary directory.

© 2020 All Rights Reserved.

- `wd`
  This option is for creating derived data and is similar to `w` except that the object is created only if the object specified by *`b_contextId`* is a master. This file is co-located with the master object, unless there is a temporary directory for the library. If there is a temporary directory, the file is created in the corresponding cell/view in the temporary directory, regardless of whether it already exists in the master directory.

- `a`
  Gets the ID of an existing object in the specified library. You can open it for modification. The object will be created if it does not exist.
  The object is created regardless of whether the library has a temporary directory and whether the object already exists in the temporary directory.

- `ad`
  Gets the ID of an existing derived object in the specified library. You can open it for modification. The object will be created if it does not exist and behaves the same as `wd`.
  If the library has a temporary directory, the derived object is always saved or created in the temporary directory, regardless of whether it already exists in the master directory.

- Default: `r` (read)

`ddGetObj` creates new cells, views, and files as a side-effect to `w`, `wd`, `a` or `ad` mode requests when the object is created. It also calls the `thePreCreateObj` and `PostCreateObj` triggers, so you can know when it does this. You need to use `ddCreateLib` to create libraries.

To create a file, use `ddGetObj` to create the file ID, followed by `filePointer = outfile( ddGetObjWritePath(fileDdId))`. Register a `PostCreateObj` trigger if you want to avoid doing an *fstat* to see if the PI created an empty file for you.

To create or access:	You need to specify these fields:
a library	*`t_libName`* or *`b_contextId`*.

© 2020 All Rights Reserved.

a library's file	*t_libName** or *b_contextId* and *t_fileName*.
a cell	*t_libName** or *b_contextId* and *t_cellName*.
a cell's file	*t_libName*, *t_cellName*, and *t_fileName* or *b_contextId* (of a library), *t_cellName, t_fileName* or *b_contextId* (of an object below the cell), *t_fileName*.
a view	*t_libName**, *t_cellName**, and *t_viewName* or *b_contextId* (of a library), *t_cellName*, and *t_viewName* or *b_contextId* (of an object below the cell, and *t_viewName*.
a view's file	*t_libName,t_cellName,t_viewName,* and *t_fileName* or *b_contextId* (of a library), *t_cellName,t_viewName*, and *t_fileName* or *b_contextId* (of an object below the cell), *t_viewName*, and *t_fileName* or b*_contextId* (of a view), and *t_fileName*.

* If a *contextId* is specified, this field does not need to be specified. However, the *libName* MUST NOT be specified if a *contextId* is specified.

Value Returned

b_ddId	An ID for the requested object.
`nil`	The object could not be found.

Example

```
fileId = ddGetObj("myLib" "myCell" "myView" "myFile" nil "r")
cellId = ddGetObj("myLib" "myCell")
viewId = ddGetObj(nil nil "myView" nil cellId)
```


© 2020 All Rights Reserved.

ddGetObjAccess

```
ddGetObjAccess(
     b_objId
     t_which
     )
     => t_access / nil
```

Description

Retrieves the UNIX access rights of the person specified by *t_which* to the object designated by *b_objId*.

If the library containing the object has a temporary directory, this function looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function returns information about the library, not its temporary directory.

Arguments

b_objId	Object ID returned by `ddCreateLib` or `ddGetObj`.
t_which	The values for *which* may be `o` (owner), `g` (group), and `p` (public).

Value Returned

t_access	A string consisting of the letters `r`, `w`, and `x` corresponding to UNIX usage.
`nil`	The string is empty if there are no permissions.

Example

```
ddGetObjAccess( TTL7400libraryId "o" ) => "rwx"
```


© 2020 All Rights Reserved.

ddGetObjAccessEx

```
ddGetObjAccessEx(
     b_objId
     t_which
     x_pathType
     )
     => t_access / nil
```

Description

Retrieves the UNIX access rights of the person specified by *t_which* to the object designated by *b_objId*. This function is similar to ddGetObjAccess except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *x_pathType* argument.

© 2020 All Rights Reserved.

Arguments

b_objId	Object ID returned by `ddCreateLib` or `ddGetObj`.
t_which	One of the following values: `o` (owner), `g` (group), or `p` (public).
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory
- `1`
  Looks for the object only in the temporary directory
- `2`
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

t_access	A string consisting of the letters `r`, `w`, and `x` corresponding to UNIX usage.
`nil`	There are no permissions.

Reference

ddGetObjAccess

© 2020 All Rights Reserved.

ddGetObjChildren

```
ddGetObjChildren(
     b_objId
     )
     => l_children / nil
```

Description

Returns the children of *b_objId*. This includes all the objects in both the temporary and master directories.

Arguments

b_objId	Object identifier returned by `ddCreateLib` or `ddGetObj`.

Value Returned

l_children	The IDs of the children of *b_objId*.
`nil`	*b_objId* does not exist or does not have any children.

Example

```
l = ddGetObj( "l1" )
ch = ddGetObjChildren( l ) => (dd:2660660 dd:2660628)
ch ~> name => ("c2" "c1")
```


© 2020 All Rights Reserved.

ddGetObjChildrenEx

```
ddGetObjChildrenEx(
     b_objId
     x_pathType
     )
     => l_children / nil
```

Description

Returns the children of *b_objId*. This function is similar to `ddGetObjChildren` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *x_pathType* argument.

Arguments

b_objId	Object ID returned by `ddCreateLib` or `ddGetObj`.
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory and, if found, returns the children from the master directory only
- `1`
  Looks for the object only in the temporary directory and, if found, returns the children from the temporary directory only
- `2`
  Returns all the children in both the master and temporary directories

Value Returned

l_children	The IDs of the children of *b_objId*.
`nil`	*b_objId* does not exist or does not have any children.

Reference

ddGetObjChildren

© 2020 All Rights Reserved.

ddGetObjDMSys

```
ddGetObjDMSys(
     b_objId
     )
     => t_sysName / nil
```

Description

Returns the name of the Design Management system of *b_objId*.

Arguments

b_objId	Object identifier returned by `ddCreateLib` or `ddGetObj`.

Value Returned

t_sysName	The name of the Design Management system.
`nil`	The Design Management system name could not be found.

Example

```
ddGetObjDMSys( myView )
    => "tdm"
```


© 2020 All Rights Reserved.

ddGetObjFiles

```
ddGetObjFiles(
     b_objId
     )
     => l_files / nil
```

Description

Lists all the files found in the object. This includes files found in both the master and temporary directories.

Arguments

b_objId	Object identifier returned by `ddCreateLib` or `ddGetObj`.

Value Returned

l_files	All of the files connected to the specified object.
`nil`	No files were found.

Example

```
l = ddGetObj( "l1" )
fl = ddGetObjFiles( l ) => (dd:2660732 dd:2660692)
fl ~> name => ( "test.ckpt" "cdsinfo.tag" )
```


© 2020 All Rights Reserved.

ddGetObjFilesEx

```
ddGetObjFilesEx(
     b_objId
     x_pathType
     )
     => l_files / nil
```

Description

Lists all the files found in the object. This function is similar to `ddGetObjFiles` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *`x_pathType`* argument.

Arguments

`b_objId`	Object identifier returned by `ddCreateLib` or `ddGetObj`.
`x_pathType`	One of the following integer values:

- `0`
  Looks for the object only in the master directory and, if found, returns the list of files only from the master directory
- `1`
  Looks for the object only in the temporary directory and, if found, returns the list of files only from the temporary directory
- `2`
  Returns the list of files from both the master and temporary directories

Value Returned

`l_files`	All of the files connected to the specified object.
`nil`	No files were found.

Reference

ddGetObjFiles

© 2020 All Rights Reserved.

ddGetObjGroup

```
ddGetObjGroup(
     b_objId
     )
     => x_int / nil
```

Description

Retrieves the UNIX group ID of *b_objId* as an integer.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function returns information about the library, not its temporary directory.

Arguments

b_objId	Object identifier returned by `ddCreateLib` or `ddGetObj`.

Value Returned

x_int	The group ID as an integer.
`nil`	No group was found.

Example

```
ddGetObjGroup( l ) => 12
```


© 2020 All Rights Reserved.

ddGetObjGroupEx

```
ddGetObjGroupEx(
     b_objId
     x_pathType
     )
     => x_int / nil
```

Description

Retrieves the UNIX group ID of *b_objId* as an integer. This function is similar to `ddGetObjGroup` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *x_pathType* argument.

Arguments

b_objId	Object identifier returned by `ddCreateLib` or `ddGetObj`.
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory
- `1`
  Looks for the object only in the temporary directory
- `2`
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

x_int	The group ID as an integer.
`nil`	No group was found.

Reference

ddGetObjGroup

© 2020 All Rights Reserved.

ddGetObjLastModify

```
ddGetObjLastModify(
     b_objId
     )
     => t_time / nil
```

Description

Returns the last-modified time from the directory or file associated with *b_objId*.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function returns information about the library, not its temporary directory.

Arguments

b_objId	ID of the object.

Value Returned

t_time	The last time the object was modified.
nil	Object was not found.

Example

```
ddGetObjLastModify( l ) => "Jan 16 10:45:10 1996"
```


© 2020 All Rights Reserved.

ddGetObjLastModifyEx

```
ddGetObjLastModifyEx(
     b_objId
     x_pathType
     )
     => t_time / nil
```

Description

Returns the last-modified time from the directory or file associated with *b_objId*. This function is similar to `ddGetObjLastModify` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *x_pathType* argument.

Arguments

b_objId	Object ID returned by `ddCreateLib` or `ddGetObj`.
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory
- `1`
  Looks for the object only in the temporary directory
- `2`
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

t_time	The last time the object was modified.
`nil`	Object was not found.

Reference

ddGetObjLastModify

© 2020 All Rights Reserved.

ddGetObjLib

```
ddGetObjLib(
     b_objId
     )
     => t_lib / nil
```

Description

Finds the ID of the library object that contains it *b_objId*. If the object is a library, it returns the same ID.

Arguments

b_objId	ID of the object.

Value Returned

t_lib	The library ID.
nil	Object was not found.

Example

```
libId = ddGetObjLib( cellId ) => dd:2544064
libId ~> name => "l1"
```


© 2020 All Rights Reserved.

ddGetObjName

```
ddGetObjName(
     b_objId
     )
     => t_name / nil
```

Description

Returns the name of *b_objId*.

Arguments

b_objId	ID of the object.

Value Returned

t_name	Name of the object.
nil	Object was not found.

Example

```
ddGetObjName( l ) => "l1"
```


© 2020 All Rights Reserved.

ddGetObjOwner

```
ddGetObjOwner(
     b_objId
     )
     => t_owner / nil
```

Description

Retrieves the UNIX owner name of *b_objId*.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function returns information about the library, not its temporary directory.

Arguments

b_objId	ID of the object.

Value Returned

t_owner	The owner's name.
nil	An invalid object ID was given.

Example

```
ddGetObjOwner( l ) => "guest"
```


© 2020 All Rights Reserved.

ddGetObjOwnerEx

```
ddGetObjOwnerEx(
     b_objId
     x_pathType
     )
     => t_owner / nil
```

Description

Retrieves the UNIX owner name of *b_objId*. This function is similar to `ddGetObjOwner` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *x_pathType* argument.

Arguments

b_objId	ID of the object.
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory
- `1`
  Looks for the object only in the temporary directory
- `2`
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

t_owner	The owner's name.
`nil`	An invalid object ID was given.

Reference

ddGetObjOwner

© 2020 All Rights Reserved.

ddGetObjParent

```
ddGetObjParent(
     b_objId
     )
     => b_parent / nil
```

Description

Returns the ID of the logical (often physical as well) owner-object od *b_objId*. For a file, it is the library, cell, or view. For a view, it is the cell. For the cell, it is the library. For the library, it is the library itself.

Arguments

b_objId	ID of the object.

Value Returned

b_parent	The ID of the parent of the specified object.
nil	Object was not found.

Example

```
p = ddGetObjParent( cell1 ) => dd:2544064
```


© 2020 All Rights Reserved.

ddGetObjReadPath

```
ddGetObjReadPath(
     b_objId
     )
     => t_path / nil
```

Description

Returns the read path to *b_objId*.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object is a library, this function returns the path to the library, not to the temporary directory.

If you want to read the file, get its read path. If you want to write the file, get its write path.

Arguments

b_objId	ID of the object.

Value Returned

t_path	The read path to *b_objId*.
nil	Object was not found.

Example

```
f = ddGetObjReadPath( file1 ) => "/tmp/l1.lib/c1/v1/sch.oa"
```


© 2020 All Rights Reserved.

ddGetObjReadPathEx

```
ddGetObjReadPathEx(
     b_objId
     x_pathType
     )
     => t_path / nil
```

Description

Returns the read path to *b_objId*. This function is similar to `ddGetObjReadPath` except that you can specify whether to look for the object in the temporary directory or the master directory with the *x_pathType* argument.

Arguments

b_objId	ID of the object.
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory
- `1`
  Looks for the object only in the temporary directory
- `2`
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns the read path to the library itself (the master directory), not its temporary directory.

Value Returned

t_path	The read path to *b_objId*.
`nil`	The object was not found.

Reference

ddGetObjReadPath

© 2020 All Rights Reserved.

ddGetObjSize

```
ddGetObjSize(
     b_objId
     )
     => x_size / -1
```

Description

Returns the length of *b_objId* in bytes.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function returns information about the library, not its temporary directory.

Arguments

b_objId	ID of a file object.

Value Returned

x_size	The file length, in bytes.
-1	The object is not a file object.

Example

```
ddGetObjSize( myViewFile ) => 1705
```


© 2020 All Rights Reserved.

ddGetObjSizeEx

```
ddGetObjSizeEx(
     b_objId
     x_pathType
     )
     => x_size / -1
```

Description

Returns the length of *b_objId* in bytes. This function is similar to `ddGetObjSize` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *x_pathType* argument.

Arguments

b_objId	ID of a file object.
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory
- `1`
  Looks for the object only in the temporary directory
- `2`
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

x_size	The file length, in bytes.
`-1`	The object is not a file object.

Reference

ddGetObjSize

© 2020 All Rights Reserved.

ddGetObjType

```
ddGetObjType(
     b_objId
     )
     => s_ddType / nil
```

Description

Returns the ddType of *b_objId*.

Arguments

b_objId	ID of the object.

Value Returned

s_ddType	The ddType of the object.
nil	Object was not found.

Example

```
ddGetObjType( f ) => ddViewFileType
```

where the possible types are

Type	The ID represents
ddLibType	a library
ddCellType	a cell within a library
ddViewType	a view within a cell
ddLibFileType	a file in the library directory
ddCellFileType	a file in the cell directory
ddViewFileType	a file in the view directory

© 2020 All Rights Reserved.

ddGetObjWritePath

```
ddGetObjWritePath(
     b_objId
     )
     => t_path / nil
```

Description

Returns the write path to *b_objId*.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object is a library, this function returns the path to the library, not to the temporary directory.

If you want to read the file, get its read path. If you want to write the file, get its write path.

Arguments

b_objId	ID of the object.

Value Returned

t_path	The write path of *b_objId*.
nil	Object was not found.

Example

```
ddGetObjWritePath( f ) => "/tmp/l1.lib/c1/v1/sch.oa"
```


© 2020 All Rights Reserved.

ddGetObjWritePathEx

```
ddGetObjWritePathEx(
     b_objId
     x_pathType
     )
     => t_path / nil
```

Description

Returns the write path to *b_objId*. This function is similar to `ddGetObjWritePath` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *x_pathType* argument.

Arguments

b_objId	ID of the object.
x_pathType	One of the following integer values:

- 0
  Looks for the object only in the master directory
- 1
  Looks for the object only in the temporary directory
- 2
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns the path to the library itself (the master directory), not to its temporary directory.

Value Returned

t_path	The write path of *b_objId*.
`nil`	The object was not found.

Reference

ddGetObjWritePath

© 2020 All Rights Reserved.

ddGetPathDMSys

```
ddGetPathDMSys(
     t_path
     )
     => t_dmSysStrin / nil
```

Description

Takes a string that is the path to a file or directory and either returns the string name of the design management system that manages that path or returns `none` if no design management system is in use for that path.

Arguments

t_ path	The path name of the file or directory.

Value Returned

t_dmSysStrin	The name of the system.
`nil`	Object was not found.

Example

```
ddGetPathDMSys("/usr/home/libs/libx/cell") returns tdm
```


© 2020 All Rights Reserved.

ddGetStartup

```
ddGetStartup(
     t_fileName
     [ b_contextId ]
     )
     => t_filePath / nil
```

Description

Looks for *t_fileName* in each standard place, and if it is not there, in *stdPlace*/share/cdssetup.

The standard places are usually

1. The work area if there is one, else the current working directory
2. $HOME
3. The project (if TDM is in use)
4. $CDS_SITE, if it has been defined
5. $CDS_INST_DIR, if it has been defined
6. The installation directory

ddGetStartup calls trigger functions registered on the triggers PreGetStartup and PostGetStartup.

Arguments

t_fileName	The first filename found.
b_contextId	The function changes the working directory to the one implied by *b_contextId* (if it differs from the current working directory).

Value Returned

t_filePath	The path to the first file found.
nil	The filename was not found.

© 2020 All Rights Reserved.

Example

```
ddGetStartup( "cds.lib" )
=> "/home/guest/cadence/cds.lib"
```


© 2020 All Rights Reserved.

ddGetTmpdirWarn

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

Description

Returns the current setting used to suppress or print a warning message for an `ASSIGN` statement with attribute `TMP` that appears in `cds.lib` files. The `TMP` attribute defines a temporary storage directory for a read-only or reference library. If you set the `DD_TMPDIR_ON` environment variable to `NO`, the temporary directory functionality is turned off and DDPI skips `ASSIGN` and `UNASSIGN` statements when parsing `cds.lib` files and, by default, prints the following warning:

```
WARNING '<PATH_TO_CDSLIB_FILE>/cds.lib', Line 3: skipping: 'ASSIGN'
lib1 TMP /tmp
```

You can change the setting for printing or suppressing warnings with the ddSetTmpdirWarn function.

Note: This function does not apply to OpenAccess ddpi.

Arguments

None

Value Returned

`t`	Current setting is set to `t`, implying that the warning will be printed.
`nil`	Current setting is set to `nil`, implying that the warning will be suppressed.

Example

```
ddGetTmpdirWarn() returns t if value is set to print warning
(Example: by ddSetTmpdirWarn(t) or by default)

ddGetTmpdirWarn() returns nil if value is set to supress warning
(Example: by ddSetTmpdirWarn(nil))
```


© 2020 All Rights Reserved.

ddGetTriggerList

```
ddGetTriggerList(
     t_triggerName
     )
     => l_triggerFunc
```

Description

Returns a list of functions registered to be executed when *t_triggerName* is activated.

Arguments

t_triggerName	The pre- or post-trigger name.

Value Returned

l_triggerFunc	A list of trigger Descriptions, in the format: *function* The trigger function name. `t` / `nil` Indicating whether the function is a SKILL function (t = SKILL). *integer* The function priority.

Example

```
ddGetTriggerList( "preupdateliblist" )
=> (( "testUpdateLibListPreTrigger" t 2147483647 ))
```


© 2020 All Rights Reserved.

ddGetUpdatedLib

```
ddGetUpdatedLib(
     )
     => t_filePath
```

Description

Returns the path to the `cds.lib` file that was last used by `ddUpdateLibList`. This is the same as `ddGetForcedLib` if a forced library was in effect.

Otherwise, it is the same as `ddGetStartup("cds.lib")` would have returned when the `ddUpdateLibList` started.

Arguments

None

Value Returned

t_filePath	The path to the `cds.lib` file that was last used by `ddUpdateLibList`.

Example

```
ddGetUpdatedLib( )
=> "/install/home/cds.lib"
```


© 2020 All Rights Reserved.

ddIsId

```
ddIsId(
     b_objId
     )
     => t / nil
```

Description

Returns `t` if *`b_objId`* is an opaque ID known to the PI, and `nil` otherwise. Some `dd` functions will accept `nil` as the ID parameter, but this function returns `nil` if the argument is `nil`.

Arguments

`b_objId`	ID of the object.

Value Returned

`t`	*`b_objId`* is an opaque ID known to the PI.
`nil`	The argument is not a valid `ddId`.

Example

```
ddIsId( myCell ) => t
```


© 2020 All Rights Reserved.

ddIsHiddenCell

```
ddIsHiddenCell(
     b_cellId
     )
     => t / nil
```

Description

Determines whether the specified cell is hidden. A cell is hidden if it is a member of the *Hidden* category or a member of a subcategory of the *Hidden* category.

Arguments

b_cellId	Object identifier of the cell.

Value Returned

t	Cell is hidden.
nil	Cell is not hidden.

Example

```
when( ddIsHiddenCell(cellId)
  printf("Cell is hidden: %s\n" cellId~>name)
)
```


© 2020 All Rights Reserved.

ddIsObjLibMarked

```
ddIsObjLibMarked(
     b_objId
     )
     => t / nil
```

Description

Returns `t` if the library that owns *`b_objId`* is marked to be deleted or its path is changed.

This can only happen while `ddUpdateLibList` is in progress. Use `ddIsObjLibMarked` in callbacks you install on the PreUpdateLibList trigger. This lets your pre-trigger function see if the library ID that you got from a PI function will exist after the `ddUpdateLibList` function that is in progress commits its changes.

Use this callback to check if any of your open files or cellviews need to be saved or deleted, or if you want to cancel the update.

Arguments

`b_objId`	ID of the object.

Value Returned

`t`	The library that owns *`b_objId`* is marked to be deleted or its path is changed.
`nil`	The object could not be found, or the object is not marked.

Example

```
ddIsObjLibMarked( myCell ) => t
```


© 2020 All Rights Reserved.

ddIsObjReadable

```
ddIsObjReadable(
     b_objId
     )
     => t / nil
```

Description

Returns `t` if the object specified by *`b_objId`* can be read by the current process.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function returns information about the library, not its temporary directory.

Arguments

`b_objId`	ID of the object.

Value Returned

`t`	*`b_objId`* can be read by the current process.
`nil`	Object was not found or is not readable.

Example

```
ddIsObjReadable( myViewFile ) => t
```

Reference

ddIsObjReadable

© 2020 All Rights Reserved.

ddIsObjReadableEx

```
ddIsObjReadableEx(
     b_objId
     x_pathType
     )
     => t / nil
```

Description

Returns `t` if the object specified by *`b_objId`* can be read by the current process. This function is similar to `ddIsObjReadable` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *`x_pathType`* argument.

Arguments

`b_objId`	ID of the object.
`x_pathType`	One of the following integer values: ❑ `0` Looks for the object only in the master directory ❑ `1` Looks for the object only in the temporary directory ❑ `2` Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

`t`	*`b_objId`* can be read by the current process.
`nil`	The object was not found or is not readable.

Reference

ddIsObjReadable

© 2020 All Rights Reserved.

ddIsObjWritable

```
ddIsObjWritable(
     b_objId
     )
     => t / nil
```

Description

Returns `t` if the object specified by *`b_objId`* can be modified by the current process.

If the library containing the object has a temporary directory, this functions looks for the object in the temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function returns information about the library, not its temporary directory.

Arguments

`b_objId`	ID of the object.

Value Returned

`t`	*`b_objId`* can be modified by the current process.
`nil`	Object was not found or is not writable.

Example

```
chmod 555 libFolder
namei -m libFolder/data.dm
=> f: libFolder/data.dm
   dr-xr-xr-x libFolder
   -rwxr-xr-x data.dm

dbIsBagWritable(ddGetObj("libFolder"))
=> t

ddIsObjWritable(ddGetObj("libFolder"))
=> nil
```


© 2020 All Rights Reserved.

Reference

ddIsObjReadableEx

© 2020 All Rights Reserved.

ddIsObjWritableEx

```
ddIsObjWritableEx(
     b_objId
     x_pathType
     )
     => t / nil
```

Description

Returns `t` if the object specified by *`b_objId`* can be modified by the current process. This function is similar to `ddIsObjWritable` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *`x_pathType`* argument.

Arguments

`b_objId`	ID of the object.
`x_pathType`	One of the following integer values: ❑ `0` Looks for the object only in the master directory ❑ `1` Looks for the object only in the temporary directory ❑ `2` Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

`t`	*`b_objId`* can be modified by the current process.
`nil`	The object was not found or is not writable.

Reference

ddIsObjReadableEx

© 2020 All Rights Reserved.

ddIsPathManageable

```
ddIsPathManageable(
     t_path
     )
     => t / nil
```

Description

Returns `t` if the supplied path is manageable (is in a location that is supported by a design management system).

Arguments

t_path	The supplied path.

Value Returned

`t`	The supplied path is manageable.
`nil`	The supplied path is not manageable.

© 2020 All Rights Reserved.

ddIsPathManaged

```
ddIsPathManaged(
     t_path
     )
     => t / nil
```

Description

Returns `t` if the specified path is currently managed by a design management system. A path is managed if the file associated with the path is checked in. You can use `ddIsPathManaged` to implement a check against creating a file when another user has already created it but the local user work area is not updated to reflect the creation.

Arguments

t_path	The specified path.

Value Returned

`t`	The specified path is managed.
`nil`	The specified path is not managed.

© 2020 All Rights Reserved.

ddLibIs5X

```
ddLibIs5X(
     b_libId
     )
     => t / nil
```

Description

Determines if the library was created with the 5.X library structure.

Arguments

b_libId	ID of the library.

Value Returned

`t`	The library was created as a 5.X library.
`nil`	The library was not created as a 5.X library.

© 2020 All Rights Reserved.

ddLibHasTmpDir

```
ddLibHasTmpDir(
     b_objId
     )
     => t / nil
```

Description

Checks if the library has a temporary directory. A temporary directory for a library is defined by putting a `TMP` attribute on the library with an `ASSIGN` statement in a `cds.lib` file.

Arguments

b_objId	ID of the library.

Value Returned

`t`	The library has a temporary directory.
`nil`	The library does not have a temporary directory.

© 2020 All Rights Reserved.

ddLockDescribe

```
ddLockDescribe(
     b_lockId
     )
     => t_message / nil
```

Description

Fills a static string with a message.

Note: This function uses the Cadence locking system (CLS) instead of `cdsd`. For information about CLS, see Chapter 11 of the *Cadence Application Infrastructure User Guide*.

Arguments

b_lockId	ID of a locked file as returned from `ddLockPath`.

Value Returned

t_message	A message used by interactive programs to tell you why there is a delay in accessing a file.
`nil`	Object was not locked.

Example

```
ddLockDescribe( myLibFileLock ) => "is currently locked by user
fabio(pid:1234)@cds3028 (age:10 sec)."
```


© 2020 All Rights Reserved.

ddLockFree

```
ddLockFree(
     b_lockId
     )
     => t / nil
```

Description

Releases any existing lock on the file specified by *b_lockId*.

Note: This function uses the Cadence locking system (CLS) instead of `cdsd`. For information about CLS, see Chapter 11 of the *Cadence Application Infrastructure User Guide*.

Arguments

b_lockId	ID of a locked file as returned from `ddLockPath`.

Value Returned

`t`	Releases the lock on the file.
`nil`	Lock could not be freed.

Example

```
ddLockFree( myLibFileLock ) => t
```


© 2020 All Rights Reserved.

ddLockGetStatus

```
ddLockGetStatus(
     b_lockId
     )
     => x_sysId t_type x_value t_userName t_userHost t_fileHost t_filePath
```

Description

Retrieves frequently needed information from the lock table: The lock-system ID, type, and value and the holder's user name and hostid.

Note: This function uses the Cadence locking system (CLS) instead of `cdsd`. For information about CLS, see Chapter 11 of the *Cadence Application Infrastructure User Guide*.

Arguments

b_lockId	ID of a locked file as returned from `ddLockPath`.

Value Returned

x_sysId	Always returns `0`.
t_type	Lock-system type of the lock. Valid Values: `"Empty"` - there is no lock `"Write"` / `"Exclusive"` - for CLS these mean the same thing, the file is locked for editing. `"Unknown"` - indeterminately locked, which means that edit lock details cannot be retrieved. For example, when the edit lock file is not readable.
x_value	Always returns `0`.
t_userName	If the lock type is anything except "`Write`" or "`Exclusive`", the return value is "`UNKNOWN`".
t_userHost	User name of host running user process. If the lock type is anything except "`Write`" or "`Exclusive`", the return value is "`UNKNOWN`".
t_fileHost	Always returns "`UNKNOWN`"

© 2020 All Rights Reserved.

t_filePath	Path locked.

Example

```
ddLockGetStatus( myFileLock ) => (0 "Write" 0 "guest" "big_sun" "UNKNOWN" "/tmp/
my.file" )
```


© 2020 All Rights Reserved.

ddLockPath

```
ddLockPath(
     t_filePath
     )
     => b_lockId
```

Description

Creates a handle that allows you to lock the file and returns a lock ID for the file. Once you call `ddLockFree`, all references to that lock ID become invalid.

Note: This function uses the Cadence locking system (CLS) instead of `cdsd`. For information about CLS, see Chapter 11 of the *Cadence Application Infrastructure User Guide*.

Arguments

t_filePath	The path to the file. The path must be a file (only files need to have locks).

Value Returned

b_lockId	The lock ID of the specified file. If the path has a lock on it, any subsequent requests to lock that path get the same lock ID.

Example

```
ddLockPath( "/tmp/myFile" ) => dd:2542360
ddLockPath( ddGetObjWritePath( myViewFile ) ) => dd:2542360
```


© 2020 All Rights Reserved.

ddLockSet

```
ddLockSet(
     b_lockId
     t_mode
     [ t_block ]
     [ t_message ]
     )
     => t / nil
```

Description

Gets the type of lock in *t_mode* for the file ID. If you have already locked the file, this function changes the lock type.

Note: This function uses the Cadence locking system (CLS) instead of `cdsd`. For information about CLS, see Chapter 11 of the *Cadence Application Infrastructure User Guide*.

Arguments

b_lockId — The lock ID as returned by `ddLockPath`.

t_mode — The lock type you want
Valid values: `l`, `r`, `w`, and `e`

- `l` (link)
  while you have a copy in memory.

  Note: The new Cadence locking system (CLS) does not have link locks. Therefore, if you specify `l`, you will not get a lock.

- `r` (read)
  while you are in the process of reading from disk.

- `w` (write)
  when you want to write a file. You cannot obtain this lock if anyone else is reading from disk or has a Write lock.

- `e` (exclusive)
  when you want to delete the file. You cannot obtain this if anyone has a Link, Read, or Write lock.

© 2020 All Rights Reserved.

g_block	When set to `t` (default), `ddLockSet` does not return until you get the lock you have requested. When set to `nil`, `ddLockSet` returns at once, whether or not it gets the lock. But you need to loop repeatedly calling `ddLockSet` until the process, which has the lock and is preventing you from getting yours, frees it.
t_message	A string you can set to explain why you want the lock, so other users will see it using `cdsdAdmin`. Defaults to `NULL`.

Value Returned

`t`	The lock was obtained, or *b_blockId* was the type of lock that you already had on the file.
`nil`	The lock was not obtained.

Example

```
ddLockSet( myFileLock "r" nil "Test lock" ) => t
```


© 2020 All Rights Reserved.

ddLockSetEx

```
ddLockSetEx(
     b_lockId
     t_mode
     [ g_block ]
     [ t_message ]
     [ g_doErrMsgs]
     [ g_timeout ]
     )
     => t / nil
```

Description

An extension of ddLockSet with two additional arguments.

Arguments

b_lockId	The lock ID as returned by `ddLockPath`.
t_mode	The lock type you want. Valid values: `l`, `r`, `w`, and `e`

- `l` (link)
  while you have a copy in memory.

  Note: The new Cadence locking system (CLS) does not have link locks. Therefore, if you specify `l`, you will not get a lock.

- `r` (read)
  while you are in the process of reading from disk.

- `w` (write)
  when you want to write a file. You cannot obtain this lock if anyone else is reading from disk or has a Write lock.

- `e` (exclusive)
  when you want to delete the file. You cannot obtain this if anyone has a Link, Read, or Write lock.

© 2020 All Rights Reserved.

g_block	When set to `t` (default), `ddLockSet` does not return until you get the lock you have requested. When set to `nil`, `ddLockSet` returns at once, whether or not it gets the lock. But you need to loop repeatedly calling `ddLockSet` until the process, which has the lock and is preventing you from getting yours, frees it.
t_message	A string you can set to explain why you want the lock, so other users will see the information. The default is `NULL`.
g_doErrMsgs	When `t`, warnings are reported.
g_timeout	Time out in seconds after which the function will quit trying to get a lock on the file if the *g_block* argument is `t` and another process has the lock.

Value Returned

`t`	The lock was obtained.
`nil`	The lock was not obtained.

Example

```
view1master = ddGetObj("lib1" "cell1" "view1" "master.tag")
view1masterpath = ddGetObjReadPath(view1master)
lockId = ddLockPath(view1masterpath)
gotLock = ddLockSetEx(lockId "w" t "proc1 lock" t 60.0)
```

`ddLockSetEx()` will time out after 60 seconds if another process has the lock on file `master.tag`

© 2020 All Rights Reserved.

ddMapGetDataTypeFileName

```
ddMapGetDataTypeFileName(
     t_dataType
     )
     => t_fileName / nil
```

Description

Returns the pattern registered in the registry for *t_dataType*.

Arguments

t_dataType	A data type from the registry.

Value Returned

t_fileName	The pattern for filenames holding *t_dataType*.
nil	The data type is not found or has no registered pattern.

Example

```
ddMapGetDataTypeFileName( "schematic" )
=> "sch.oa"
```


© 2020 All Rights Reserved.

ddMapGetDataTypeList

```
ddMapGetDataTypeList(
     )
     => l_list
```

Description

Returns a list of names of the data types specified in the registry.

Arguments

None

Value Returned

l_list	A list of data type names.

Example

```
ddMapGetDataTypeList()
=> ("AHDLNetlist" "AHDLText" "AmadeusPackage" "AmadeusVerilogMap" "AsciiText"
"CDBAGraphics" "CDBANetlist" "CDBAStranger" "ComposerPIC" "ComposerSchematic"
"ComposerSymbol" "ConceptSchematic" "DesignFlow" "HierarchyDescription"
"MaskLayout" "ThermaxBoundary" "ThermaxDesign" "ThermaxProject" "ThermaxResult"
"VHDLText" "VerilogNetlist" "VerilogText")
```


© 2020 All Rights Reserved.

ddMapGetFileDataType

```
ddMapGetFileDataType(
     b_fileObjId
     )
     => t_dataType / nil
```

Description

Returns the name of the data type associated with *b_fileObjId*.

Arguments

b_fileObjId	The object file ID.

Value Returned

t_dataType	The data type name.
nil	The data type is not found.

Example

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )
ddMapGetFileDataType( vf ) => "ComposerSchematic"
```


© 2020 All Rights Reserved.

ddMapGetFileTool

```
ddMapGetFileTool(
     b_fileObjId
     )
     => t_fileTool / nil
```

Description

Looks up the data format registered in the registry for *b_fileObjId*'s file name, and returns the tool name of the preferred editor (if there is one) or the default editor which is registered for that data format.

Arguments

b_fileObjId	The object file ID.

Value Returned

t_fileTool	File tool of the file, a permanent string.
nil	The file tool is not found.

Example

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )
ddMapGetFileTool( vf ) => "Composer-Schematic"
```


© 2020 All Rights Reserved.

ddMapGetFileViewType

```
ddMapGetFileViewType(
     b_fileObjId
     )
     => t_viewType / nil
```

Description

Returns the name of the view type associated with *b_fileObjId*.

Arguments

b_fileObjId	The object file ID.

Value Returned

t_viewType	The view type name.
nil	The view type is not found.

Example

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )
ddMapGetFileViewType( vf ) => "schematic"
```


© 2020 All Rights Reserved.

ddMapGetToolNameDataType

```
ddMapGetToolNameDataType(
     t_toolName
     )
     => t_dataType / nil
```

Description

Returns the name of the first data type in the registry which is associated with *t_toolName*.

Arguments

t_toolName	The name of the tool.

Value Returned

t_dataType	The data type name.
nil	The data type is not found.

Example

```
ddMapGetToolNameDataType( "Composer-Schematic" )
=> "ComposerSchematic"
```


© 2020 All Rights Reserved.

ddMapGetToolNameList

```
ddMapGetToolNameList(
     )
     => l_toolList
```

Description

Retrieves a list of the names of the tools specified in the registry, in alphabetical order.

Arguments

None

Value Returned

l_toolList	A list of tool names.

Example

```
ddMapGetToolNameList( )
=> ( "Amadeus-Package-Editor" ... "vi" )
```


© 2020 All Rights Reserved.

ddMapGetToolNameViewType

```
ddMapGetToolNameViewType(
     t_toolName
     )
     => t_viewTypeName / nil
```

Description

Returns the name of the first view type in the registry which is associated with *t_toolName*.

Arguments

t_toolName	Name of the tool from the registry.

Value Returned

t_viewTypeName	The view type name.
nil	The view type is not found.

Example

```
ddMapGetToolNameViewType( "Composer-Schematic" )
=> "schematic"
```


© 2020 All Rights Reserved.

ddMapGetViewTypeFileName

```
ddMapGetViewTypeFileName(
     t_viewType
     )
     => t_fileName / nil
```

Description

Returns the pattern registered in the registry for a view alias or data type named *t_viewType*.

Arguments

t_viewType	A view alias or data type name.

Value Returned

t_fileName	The filename.
nil	The view type is not found.

Example

```
ddMapGetViewTypeFileName( "schematic" ) => "sch.oa"
```


© 2020 All Rights Reserved.

ddMapGetViewTypeList

```
ddMapGetViewTypeList(
     )
     => l_list
```

Description

Returns a list of names of view types defined in the registry, in alphabetical order. If a data type in the registry has a *dfII_viewType* property defined, the value of that property will be in the list. Otherwise, the data type name will be in the list.

Arguments

None

Value Returned

l_list	A list of view type names.

Example

```
ddMapGetViewTypeList( ) => ( "ConceptSchematic" ... "vhdl" )
```


© 2020 All Rights Reserved.

ddMapIsFileNameKnown

```
ddMapIsFileNameKnown(
     t_fileName
     )
     => t / nil
```

Description

Returns `t` if the file name is registered as the pattern for some data format in the registry.

Arguments

t_fileName	The file name from the registry.

Value Returned

`t`	The *t_fileName* is found in the registry.
`nil`	*t_fileName* was not found in the registry.

Example

```
ddMapIsFileNameKnown( "sch.oa" ) => t
```


© 2020 All Rights Reserved.

ddMapIsToolLegal

```
ddMapIsToolLegal(
     b_fileObjId
     t_toolName
     )
     => t / nil
```

Description

Returns `t` if *`t_toolName`* is either the preferred editor or the default editor for *`b_fileObjId`*.

Arguments

`b_fileObjId`	The object file ID.
`t_toolName`	The tool name in the registry.

Value Returned

`t`	*`t_toolName`* is either the preferred editor or the default editor for *`b_fileObjId`*.
`nil`	The tool is not one of the tools specified for *`b_fileObjId`*.

Example

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )
ddMapIsToolLegal( vf "Virtuoso" ) => nil
```


© 2020 All Rights Reserved.

ddMapIsToolNameKnown

```
ddMapIsToolNameKnown(
     t_toolName
     )
     => t / nil
```

Description

Returns `t` if *`t_toolName`* is defined in the registry.

Arguments

`t_toolName`	Name of the tool in the registry.

Value Returned

`t`	The tool is known.
`nil`	The tool name is not found.

Example

```
ddMapIsToolNameKnown( "Composer" ) => t
```


© 2020 All Rights Reserved.

ddMarkObjMaster

```
ddMarkObjMaster(
     b_objId
     )
     => t / nil
```

Description

Changes the `master.tag` file to designate the object as the new master. The `master.tag` file must be located with the file it designates, so you must be able to write a file to mark it master. The `master.tag` file is checked out of the Design Management system, if necessary.

This function returns an error if *`b_objId`* exists only in the library's temporary directory. Objects in a temporary directory are derived data and cannot be marked as master.

Arguments

`b_objId`	ID of the object, which must be a `viewFileType` object.

Value Returned

`t`	The `master.tag` file was changed.
`nil`	The `master.tag` file could not be changed.

Example

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )
ddMarkObjMaster( vf ) => t
```


© 2020 All Rights Reserved.

ddNeedCheckout

```
ddNeedCheckout(
     b_objId
     )
     => t / nil
```

Description

Enable tools that need to provide automatic checkout behavior to decide if they need to call the functions `ddCheckout` and `ddCheckin`.

Arguments

b_objId	The ddId for a file object.

Value Returned

`t`	*b_objId* is a DM-managed object that needs to be checked out.
`nil`	The library is not managed by a Design Management system, *b_objId* does not need to be checked out, or *b_objId* exists only in the library's temporary directory (objects in the temporary directory cannot be managed).

Example

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )
if ddNeedCheckout( vf ) then
    ddCheckout( vf )
```


© 2020 All Rights Reserved.

ddObjHasNamedObj

```
ddObjHasNamedObj(
     b_objId
     t_fileName
     )
     => t / nil
```

Description

Returns `t` if *`b_objId`* has a child object named *`t_fileName`*.

`b_objId` can be only a library, cell, or view identifier. Scans objects relative to the given object and looks for something that matches by name. This function looks for the object in the library's temporary directory first; if it is not found, it looks for it in the master directory. However, if the object itself is a library, this function looks for the children in the library (the master directory).

`ddObjHasNamedObj` is meant to be a "fast" lookup function (as fstat is not used in the scan). To get the `ddId`, use `ddGetObj` using the given object as the context ID and specifying *`t_fileName`* as the appropriate relative argument.

Arguments

`b_objId`	The object file ID.
`t_fileName`	The filename to search for.

Value Returned

`t`	The file was found.
`nil`	The file was not found.

Example

```
l = ddGetObj( "l1" ) => dd:2543176
ddObjHasNamedObj(l "c1") => t
ddObjHasNamedObj(l "v1") => nil
```


© 2020 All Rights Reserved.

ddObjHasNamedObjEx

```
ddObjHasNamedObjEx(
     b_objId
     t_fileName
     x_pathType
     )
     => t / nil
```

Description

Returns `t` if *`b_objId`* has a child object named *`t_fileName`*. This function is similar to `ddObjHasNamedObj` except that you can specify whether to look for the object in the temporary directory or the master directory, or both, with the *`x_pathType`* argument.

See ddObjHasNamedObj for more information.

© 2020 All Rights Reserved.

Arguments

b_objId	The object file ID.
t_fileName	The file to search for.
x_pathType	One of the following integer values:

- `0`
  Looks for the object only in the master directory
- `1`
  Looks for the object only in the temporary directory
- `2`
  Looks for the object in the temporary directory first; if it is not found, looks for it in the master directory. However, if the object is a library, returns information about the library itself (the master directory), not its temporary directory.

Value Returned

`t`	The file was found.
`nil`	The file was not found.

Reference

ddObjHasNamedObj

© 2020 All Rights Reserved.

ddObjIsInMaster

```
ddObjIsInMaster(
     b_objId
     )
     => t / nil
```

Description

Checks if the object exists in the master directory.

Arguments

b_objId	ID of the object.

Value Returned

t	The object is in the master directory.
nil	The object is not in the master directory.

Reference

ddObjIsInTmp

© 2020 All Rights Reserved.

ddObjIsInTmp

```
ddObjIsInTmp(
     b_objId
     )
     => t / nil
```

Description

Checks if the object exists in the temporary directory. A temporary directory for a library is defined by putting a `TMP` attribute on the library with an `ASSIGN` statement in a `cds.lib` file. The library itself is considered the master directory.

Arguments

b_objId	ID of the object.

Value Returned

`t`	The object is in the temporary directory.
`nil`	The object is not in the temporary directory.

Reference

ddObjIsInMaster

© 2020 All Rights Reserved.

ddProduceObjRelNames

```
ddProduceObjRelNames(
     b_objId
     g_consumer
     g_clientData
     )
     => t / nil
```

Description

Produces the names of all the objects that belong to a library, cell, or view and calls the given function with each name and any provided data structure.

The list of names includes objects in the library's temporary directory, if it exists, as well as those in the library itself (the master directory).

The objective is to generate the names of the objects underneath a library, cell, or view when the caller is only interested in the names and not what type of object the name is associated with. This avoids doing expensive fstats on the objects.

If an object is created while the producer is active, its name may not be visible *g_consumer*. If an object is deleted after the producer has been started but before you access it, *g_consumer* could get a name for an object that does not exist.

You should use `ddGetObj` to get a ddId for the object if you need one. You should check the return value of `ddGetObj` for the reasons described above.

Arguments

b_objId	The parent object.
g_consumer	The *g_consumer* function should be of the form `myClient( t_objName, g_clientData ) => t/nil` *t_objName* Is the name of the next objects (directory or file) being produced. *g_clientData* Is the client data that the producer was given.
g_clientData	Any arbitrary piece of data you want passed to the *g_consumer*.

© 2020 All Rights Reserved.

Value Returned

`t`	*`g_consumer`* returned `t` for each name passed to it.
`nil`	There are no objects or if the name production is interrupted by a `nil` return value of *`g_consumer`*.

Example

Here is how you call this procedure from SKILL:

Sample client:

```
(procedure (myClient name  arg)
    rplacd(last(arg) ncons(name)) t )
```

Note: The client always returns `t` so it gets the entire list. If it were searching for a particular name, it would return `nil` after finding it.

Sample usage:

```
plist = '( "ProducedList" )
l1 = ddGetObj( "l1" )
ddProduceObjRelNames( l1, 'myClient plist )
    => t
plist
    => ( "ProducedList" "cdsinfo.tag" "cell1" )
```


© 2020 All Rights Reserved.

ddProduceObjRelNamesEx

```
ddProduceObjRelNamesEx(
     b_objId
     g_consumer
     g_clientData
     x_pathType
     )
     => t / nil
```

Description

Produces the names of all the objects that belong to a library, cell, or view and calls the given function with each name and any provided data structure. This function is similar to `ddProduceObjRelNames` except that you can specify whether to look for the objects in the temporary directory or the master directory, or both, with the *`x_pathType`* argument.

See ddProduceObjRelNames for more information.

Arguments

`b_objId` — The parent object.

`g_consumer` — The *`g_consumer`* function should be of the form:
`myClient(` *`t_objName, g_clientData`* `) => t/nil`

- *`t_objName`*
  Is the name of the next objects (directory or file) being produced.
- *`g_clientData`*
  Is the client data that the producer was given.

`g_clientData` — Any arbitrary piece of data you want passed to the *`g_consumer`*.

© 2020 All Rights Reserved.

x_pathType	One of the following integer values: ❑ 0 Looks for *b_objId* only in the master directory and returns the names of the objects from the master directory only ❑ 1 Looks for *b_objId* only in the temporary directory and returns the names of the objects from the temporary directory only ❑ 2 Looks for *b_objID* in the temporary directory and the master directory and returns all the objects found in both the temporary and master directories

Value Returned

t	*g_consumer* returned t for each name passed to it.
nil	There are no objects or if the name production is interrupted by a nil return value of *g_consumer*.

Reference

ddProduceObjRelNames

© 2020 All Rights Reserved.

ddRegHiddenCellsFunc

```
ddRegHiddenCellsFunc(
     s_hiddenCellsFunc
     [ t_libName]
     )
     => t / nil
```

Description

Registers a user-defined SKILL function that will be called to obtain a list of hidden cell names. Note that additionally the Virtuoso environment considers a cell to be hidden if it is a member of the *Hidden* category or its subcategories.

Hidden cell functions should be registered using the .cdsinit file so that hidden cells are available during Virtuoso startup for tools such as Library Manager.

Arguments

t_hiddenCellsFunc	Name of a user-defined SKILL function to be registered. The function should accept the ddID of a library and return a list of hidden cells names. Once the hidden cells are obtained for a library using a hidden cells function, the information is cached until a refresh is done from the Library Manager or Virtuoso CIW or a ddUpdateLibList call is made.
t_libName	Name of a library. If a name is specified, the registered function applies to only the specified library. Otherwise, the function applies to all libraries in the DDPI library list.

Value Returned

t	The function was registered.
nil	The function was not registered. The function specified might not exist or be callable.

Example

Consider the `myHiddenCellsFunc` function, which hides cells that start with an underscore (_) in the `myLib` library:

© 2020 All Rights Reserved.

```
procedure(myHiddenCellsFunc(libId)
 setof(cellName libId~>cells~>name strncmp(cellName "_" 1) == 0)
)
```

Register the `myHiddenCellsFunc` function as follows:

```
ddRegHiddenCellsFunc('myHiddenCellsFunc "myLib")
```


© 2020 All Rights Reserved.

ddRegTrigger

```
ddRegTrigger(
     t_triggerName
     s_triggerFunc
     [ x_priority ]
     )
     => t / nil
```

Description

Registers a function to be called when the triggering event occurs. Adds callbacks (triggers) to PI functions that change things. Each of these PI functions has a pre-trigger and an post-trigger.

Functions you attach to a pre-trigger are passed the same arguments that were passed to the associated PI function. Functions you attach to a pre-trigger should be written to return either `t` or `nil`. If the value is `nil`, no further trigger functions are called, and the associated PI function will be canceled.

Functions that you attach to a post-trigger will be passed the same arguments that were passed to the associated PI function, and in addition, the last argument will be the value to be returned by the PI function. Post-trigger functions must be written to return void.

Arguments

t_triggerName The trigger you wish to attach a callback to.

Valid Values:

PI function	Pre-Trigger name	Post-Trigger name
ddUpdateLibList()	“PreUpdateLibList”	“PostUpdateLibList”
ddGetStartup()	“PreGetStartup”	“PostGetStartup”
ddDeleteObj()	“PreDeleteObj”	“PostDeleteObj”
ddCreateLib()	“PreCreateLib”	“PostCreateLib”
ddGetObj()	“PreCreateObj”	’PostCreateObj” “FirstAccessLib”
ddCheckin()	“PreCheckin”	’PostCheckin”
ddCheckout()	“PreCheckout”	“PostCheckout”

© 2020 All Rights Reserved.

Valid Values:

PI function	Pre-Trigger name	Post-Trigger name
ddAutoCheckout()	“PreAutoCheckout”	

Note: `ddGetObj` calls triggers when it creates an object and the first time it accesses a library in the library list.

The names are case insensitive.

`s_triggerFunc` The function you want to have called back when the trigger is activated.

`x_priority` The order in which functions are called after they have been registered as triggers. Priority is an integer in the range `( 0 <= priority <= INT_MAX )`.

Functions that have the same priority are called last-registered, first called-back. Triggers that have lower priorities are called-back before triggers that have higher priorities. The default priority is INT_MAX.

Value Returned

`t` The function was successfully registered to trigger when the event occurs.

`nil` Execution of the function failed.

Example

```
ddRegTrigger( "PreCheckin" 'checkinTrigFunc 0 ) => t
```


© 2020 All Rights Reserved.

ddReleaseObj

```
ddReleaseObj(
     b_objId
     )
     => t / nil
```

Description

When you find an object using write (`w`) or append (`a`) mode, and the object and its parents do not yet exist, `ddGetObj` creates the object and its parents in the file system. The effect of this action is to make them visible in the browser and to permit us to designate file as a Master. After you close the file, you should call `ddReleaseObj`, which then deletes the file if you did not save it yet.

DDPI looks for the file designated by *obj*. If its size is zero, DDPI deletes it. (If the file exists in both the temporary and master directories, its size must be zero in both the directories for DDPI to delete it.) After you close the file, call `ddReleaseObj`, which deletes the file if you did not save it yet. If it is a *viewFile* and it is a Master with size zero, DDPI deletes the entire view.

Arguments

b_objId	The ID of the object you want to release.

Value Returned

`t`	The object was released.
`nil`	*b_objId* was not a valid object, or one of the PreDeleteObj triggers returns `nil`.

Example

```
ddReleaseObj( viewFileId ) => t
```


© 2020 All Rights Reserved.

ddSetForcedLib

```
ddSetForcedLib(
     t_path
     )
     => t / nil
```

Description

If the given path is a valid path, the *forcedLib* is used for all future library updates/creates/deletes and `t` is returned. If the path is an empty string, the default `cds.lib` file is used for all future library updates/creates/deletes and `t` is returned. If a path is given that is not valid, `nil` is returned with no error message and the *forcedLib* will be unchanged.

Note: `ddSetForcedLib` does not automatically update the list of libraries. Call ddUpdateLibList to update the list of libraries using the forced library definition file that was set.

Arguments

t_path	The path to the `cds.lib` file you want to use.

Value Returned

`t`	The path to the `cds.lib` file is valid.
`nil`	The path to the `cds.lib` file is not valid.

Example

```
ddSetForcedLib( "/user/library/cds.lib" ) => t
```

Reference

ddSetForcedLibEx, ddGetForcedLibEx

© 2020 All Rights Reserved.

ddSetForcedLibEx

```
ddSetForcedLibEx(
     g_cdsLibFile
     [ g_libDefsFile ]
     )
     => t / nil
```

Description

Sets the `cds.lib` file to be used instead of searching for them in the default locations. If no `cds.lib` is specified but a `lib.defs` file is specified, the function will use the specified `lib.defs` as a forced `cds.lib` file. Cadence recommends you to use the ddSetForcedLib function instead.

Note: There are a few differences between `cds.lib` and `lib.defs` format. For more information on `cds.lib`, see Cadence Library Definition File in *Cadence Application Infrastructure User Guide*.

Note: The `lib.defs` are no longer supported by Virtuoso.

Arguments

g_cdsLibFile	The path to the `cds.lib` file to use.
g_libDefsFile	The path to the `lib.defs` file to use. If `cds.lib` is specified, this argument is ignored. If `cds.lib` is not specified, the specified `lib.defs` file will be handled as a `cds.lib` file. In some cases this could lead to warnings due to differences in format of `cds.lib` and `lib.defs`.

Value Returned

`t`	The file was set.
`nil`	There was an error.

Example

```
ddSetForcedLibEx("../cds.lib" nil)
```


© 2020 All Rights Reserved.

Reference

ddSetForcedLib, ddGetForcedLibEx

© 2020 All Rights Reserved.

ddSetLibUnmanaged

```
ddSetLibUnmanaged(
     b_libId
     )
     => t / nil
```

Description

Sets the DMTYPE entry of the `cdsinfo.tag` file in the library directory to `none`. This will guarantee that regardless of the default value for DMTYPE, this library will not be DM managed. If the library has a temporary directory, it will also not be managed.

See also `ddClearLibUnmanaged`, `ddIsPathManagable`, `ddIsPathManaged`, `ddAmUsingDm`, `ddCheckin`.

For more information about the `cdsinfo.tag file`, see the *Cadence Application Infrastructure User Guide*.

Arguments

b_libId	The ID of a library object.

Value Returned

`t`	The DMTYPE entry is set.
`nil`	The DMTYPE entry is not set.

© 2020 All Rights Reserved.

ddSetLibDmType

```
ddSetLibDmType(
     b_libId
     t_dmType
     )
     => t / nil
```

Description

Sets the DMTYPE entry of the `cdsinfo.tag` file in the library.

Arguments

`b_libId`	The ID of a library object.
`t_dmType`	The Design Management system name to which the DMTYPE entry in the `cdsinfo.tag` file is being set.

Value Returned

`t`	The DMTYPE entry is set.
`nil`	The DMTYPE entry is not set.

Example

```
libId = ddGetObj("lib1")
ddSetLibDmType(libId,"vs")
```

Returns `t` when you set the DMTYPE entry in the `cdsinfo.tag` file for lib1 to "*vs".* Otherwise, returns nil.

© 2020 All Rights Reserved.

ddSetTmpdirWarn

```
ddSetTmpdirWarn(
     g_doWarn
     )
     => t / nil
```

Description

Enables or disables printing of a warning message for an `ASSIGN` statement with attribute `TMP` which appears in `cds.lib` files. The `TMP` attribute defines a temporary storage directory for a read-only or reference library. If you set the `DD_TMPDIR_ON` environment variable to `NO`, the temporary directory functionality is turned off and DDPI skips `ASSIGN` and `UNASSIGN` statements when parsing `cds.lib` files. In that case, DDPI prints the following warning by default:

```
WARNING '<PATH_TO_CDSLIB_FILE>/cds.lib', Line 3: skipping: 'ASSIGN'
lib1 TMP /tmp
```

If you do not want the warning message to be printed, specify `nil` for the *`g_doWarn`* argument.

Note: This function does not apply to OA ddpi.

Arguments

`g_doWarn`	Enables or disables the printing of the warning message. Valid Values: `t` (the current value will be set to `t` to enable printing of the warning) `nil` (the current value will be set to `nil` to disable printing of the warning)

Value Returned

`t`	Printing of the warning message is enabled.
`nil`	Printing of the warning message is disabled.

© 2020 All Rights Reserved.

ddUnregHiddenCellsFunc

```
ddUnregHiddenCellsFunc(
     s_hiddenCellsFunc
     )
     => t / nil
```

Description

Unregisters a hidden cells function that was registered using ddRegHiddenCellsFunc.

Arguments

s_hiddenCellsFunc	Function to be unregistered.

Value Returned

t	The function was unregistered.
nil	The function was not unregistered. The specified function might not exist or was not registered using ddRegHiddenCellsFunc.

Example

Unregister the myHiddenCellsFunc function as follows:

```
ddUnregHiddenCellsFunc('myHiddenCellsFunc)
```


© 2020 All Rights Reserved.

ddUnregTrigger

```
ddUnregTrigger(
     t_triggerName
     s_triggerFunc
     )
     => t / nil
```

Description

Removes a callback (trigger function) from a named list of trigger functions.

Arguments

t_triggerName	Name of the trigger to remove the callback from. Refer to the ddRegTrigger definition for a list of the permissible values of *t_triggerName*.
s_triggerFunc	The function to remove from the trigger's callback list.

Value Returned

t	The function was successfully unregistered and will no longer be triggered when the event occurs.
nil	*s_triggerFunc* has not been registered as a trigger for *t_triggerName*.

Example

```
ddUnregTrigger( "preCheckIn" 'checkinTrigFunc ) => t
```


© 2020 All Rights Reserved.

ddUpdateLibList

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

Description

Establishes and re-establishes a list of libraries in virtual memory to be used by `ddGetObj`.

The order of libraries in the list is irrelevant; there is no search path behavior among libraries. This function retrieves the entries in a series of one or more `cds.lib` files to define the libraries that your tool will see. (Only the first library file needs to have the name `cds.lib`.)

If you use the `-cdslib` switch on the command line or you have called `ddSetForcedLib`, `ddUpdateLibList` checks, when it first runs, to see if you have specified a path to a `cds.lib` file. If so, it skips the `ddGetStartup` search for the `cds.lib` file and just starts with the one you asked for. If it cannot read the file, the system aborts the run.

If no `cdslib` switch was given and there is no *forcedLib* in effect, `ddUpdateLibList` loads the `cds.lib` file `ddGetStartup` finds.

The function has two triggers: PreUpdateLibList and PostUpdateLibList. The pre-trigger is called after the system analyzes the library files to be sure they are error-free and all libraries are visible.

You should register a callback function on the PreUpdateLibList trigger that will walk through all your open files. For each file ddId, call `ddIsObjLibMarked` to see of the object's library is marked as having been deleted or having changed paths. If so, you can ask the users to save their work at the old path, lose their work, or abort the update. If a user chooses to abort the update, issue a warning to the log and return `FALSE` from the pre-trigger.

`ddUpdateLibList` gets locks on the `cds.lib` and registry while accessing them, and then releases them.

Arguments

None

© 2020 All Rights Reserved.

Value Returned

`t`	The library list update was successful.
`nil`	The library list update failed due to either syntax errors in the `cds.lib` file or one of the PreUpdateLibList triggers returning `nil`.

Example

```
ddUpdateLibList( ) => t
```


© 2020 All Rights Reserved.

Design Data Services

This section describes the *ddserv* code layer. The primary reason for this code is to encapsulate the usage of the external libManager, libSelect, and hierEditor processes using Multi-Process SKILL (MPS).

Library Manager

The libManager process provides general library management utilities and calls the Library Manager tool. The Library Manager performs browsing and issues edit, read, and new commands to a registered process (in this case, a workbench binary like *icca*). The *ddserv* layer on the DFII side invokes libManager if it needs to, imports the libManager MPS services, and exports services that the libManager uses to perform several of its functions. There is a *Library Manager* menu command on the *File* pulldown on the CIW.

Hierarchy Editor

The hierEditor process and hierarchy editor tool were implemented to support the composition and editing of how a design is to be expanded. For the first time, you can use a single tool to define your design expansion rules that all applications will obey. *ddserv* also supports the MPS connection to the hierarchy editor.

Library Browser

The libSelect process is used for browsing; it is the mechanism that replaces the DFII DM DAG browser when applications provide the user with a *Browse* button on their forms. *ddserv* also provides a level of backward compatibility with prior releases by supporting a subset of the DFII DM *dm* and *dmb* functions.

The Library Browser was implemented to support the existing DFII DM capability to browse libraries and seed a form with the selected items. The *ddserv* layer supports a new PI as well as some of the old *dmb* procedures and manages the MPS interactions with the libSelect process.

Environment Variables

The `dmbLibraryBrowser` variable is no longer supported.

The following environment variables are implemented to allow you to control and preset behavior and form field values. These variables let you seed the related form by using the `~/`

© 2020 All Rights Reserved.

`.cdsenv` file.

Tool	Variable	Type	Default	Choices
ddserv.browser	onOpen	boolean	nil	
ddserv.browser	onPlace	boolean	nil	
ddserv.browser	onOther	boolean	nil	
ddserv.ciw	promptOnExit	boolean	t	
ddserv.lib	dmchoice	string	"No DM"	
ddserv.lib	newLibDir	string	""	
ddserv.lib	techChoice	cyclic	“compile”	“compile” “bind” “none” “reference”

© 2020 All Rights Reserved.

The following environment variables are implemented to control the opening of a configuration cellview. These variables let you seed the related GUI fields by using the `~/.cdsenv` file.

Tool	Variable	Type	Default	Choices	Description
ddserv.he	openConfig	cyclic	"prompt"	"prompt"	Displays the secondary Open Configuration or Top CellView form.
				"topCV"	Opens only the top cellview.
				"config"	Opens only the configuration cellview.
				"config+topCV"	Opens both top and configuration cellviews.
				"cdsenv"	Opens top and/or configuration cellviews based on the values set for `ddserv.he schEditor` and `ddserv.he hierEditor` respectively. If the values are `nil`, the secondary open form is displayed.

© 2020 All Rights Reserved.

ddserv.he	topCvMode	cyclic	"same"	"same"	Opens the top cellview in the same mode specified for opening the configuration cellview.
				"read"	Opens the top cellview in "read" mode.
				"append"	Opens the top cellview in "append" mode.

Current Library

With the new library structure, there is no concept of a current library (nothing to open, nothing to close), however, it has been requested that accessing the currently selected library in the Library Browser is desired. To support this, two new functions, `ddsGetCurrentLib` and `dmbGetCurrentLib`, have been implemented (the two are basically the same except for consistency, the `dmb` function is provided for backward compatibility). These functions return the ddId of the library that is currently selected in the Library Browser; `nil` is returned if no library is selected or if the Library Browser is not synchronized with any particular form. Code directly accessing the `dmbCurLib` variable or calling `dmGetCurLib` might no longer be necessary.

Synchronization

Applications call `ddsSyncWithForm` (`dmbSyncWithForm`) to start the synchronization and `ddsEndSyncWithForm` (`dmbLibBrowserFormDone`) to end it (the `dmb` functions are provided for backward compatibility). These calls must be paired to properly maintain the Library Browser state.

Integrating Your Own Library Browser

The function *ddserv* supports the integration of a custom Library Browser thus replacing the default Library Browser, *libSelect*. While integrating your own Library Browser in DFII, you must consider the following:

© 2020 All Rights Reserved.

- *Invocation* - The default DFII Library Browser is a separate process from DFII. Therefore, DFII manages the execution of the default Library Browser as well as communicates with it. This process is simplified if your custom Library Browser resides within the DFII process, for example, as a form or a *dag* browser. Any initialization or invocation of the custom Library Browser should be made in the replacement functionality that would be registered for `ddsSyncWithForm`.
- *Replay* - As you execute commands, they are recorded to the DFII `.log` file. You can then turn around and use the log file to replay the session. Because of the asynchronous nature of external processes, *ddserv* cannot replay the synchronizations with the DFII Library Browser. The *ddsSync* functions will do nothing if the session is in replay. However, if your custom Library Browser resides within the DFII process, it can synchronized to work if the session is in replay.
- *Indirect Recursion* - Some forms synchronized to the Library Browser may have callbacks on the registered form fields. Some of these callbacks may call `ddsUpdateSyncWithForm` to ensure that the Library Browser selections match what you may have typed in. This could lead to the Library Browser updating the form fields with its current selections, the form field callbacks firing off, which would then try to synchronize the Library Browser with its values. You must take care in either the interface layer or the Library Browser itself to avoid having the form and the Library Browser continually trying to update each other.
- *Refresh* - The custom Library Browser should react to the creation and deletion of libraries, cells, and views. There are respective Design Data Procedural Interface (*ddpi*) triggers available from which you can perform the Library Browser update.

To plug in a different Library Browser, you need to implement and register to *ddserv* the replacement functionality for `ddsSyncWithForm`, `ddsUpdateSyncWithForm`, `ddsEndSyncWithForm`, and `ddsGetCurrentLib`. You do the following:

- You register your routines that interface to your Library Browser with *`ddserv`*.
- When a custom Library Browser is so registered, the four routines `ddsSyncWithForm`, `ddsUpdateSyncWithForm`, `ddsEndSyncWithForm`, and `ddsGetCurrentLib` will invoke the registered functionality rather than interfacing with the default DFII library browser. No application code has to make any changes as their existing calls to the *ddserv* routines will buffer them from what Library Browser is being used.
- You may un-register your custom Library Browser at any time and register it again. This is for completeness and can assist in debugging (saves from having to exit the system if there are problems with the custom Library Browser integration).

© 2020 All Rights Reserved.

Registering the Custom Library Browser

To plug in a different Library Browser, four functions must be registered to *ddserv.* You must provide all four functions to effectively emulate the functionality provided by *ddserv*. There is a single registration function to call `ddsRegCustomLibSelect`.

SKILL Procedural Interface Functions

The following pages briefly describe the public SKILL procedural interface functions provided in *ddserv*.

You can synchronize the Library Browser with the user interface Open Design form, with several of the forms for placing an instance, and with various other forms. By convention, any form that can be synchronized with the Library Browser has a *Browse* button.

When the Library Browser and a form are synchronized, any information typed into the library, cell, view, or version fields of the form causes a corresponding selection in the Library Browser. Any selections made in the Library Browser are automatically put in the synchronized form.

For a form and the Library Browser to be synchronized, you must first select the *Browse* button on the form. You can also control how the Library Browser window behaves when forms are brought up and brought down by setting the `ddserv.browser` environment variable in your `.cdsenv` file (`dmbSetBrowserInteractions` is provided for backward compatibility).

Customizing CIW Menus

DFII based tools that have pulldown menus in other windows and that install menu items into the CIW can customize these pulldown and pop-up menus to use as part of your tool.

Menu files are used to define the menu entries and the pull downs in them for the CIW and your application. Menu files allows for setup, full customization by workbench owners and customers, and isolation for applications from the details of the menu item and pulldown menu implementation.

© 2020 All Rights Reserved.

ciwAddTranslatorItem

```
ciwAddTranslatorItem(
     s_tag
     t_itemText
     t_callback
     )
     => t / nil
```

Description

Adds an entry for the `Import` or `Export` commands.

Arguments

s_tag	A SKILL symbol.
	Valid Values: `'Import`, `'Export`
t_itemText	The Item text to be used as a display prompt.
t_callback	Called to execute the given translator.

Value Returned

`t`	The entry was added.
`nil`	The entry was not added.

© 2020 All Rights Reserved.

ciwCreateMenu

```
ciwCreateMenu(
     ls_menuDef
     [ x_windowNum ]
     )
     => r_menu / nil
```

Description

Creates a menu from the given menu definition. The menu created can be used as a pop-up menu. This function exists primarily for application support.

The *ls_menuDef* argument is normally read from a menu file. If *x_windowNum* is specified for example, `window->windowNum`, the menu handle and menu item symbols are uniquely created by appending the symbol in the menu definition with the window number.

Arguments

ls_menuDef	The definition of a menu, either a list defining the contents of the pulldown menu or the SKILL symbol that represents the list.
x_windowNum	Window number.

Value Returned

r_menu	A menu suitable for use as a pop-up menu.
`nil`	The list defining the contents for the pop-up or SKILL symbol could not be found.

© 2020 All Rights Reserved.

ciwCreatePulldown

```
ciwCreatePulldown(
     ls_menuDef
     [ x_windowNum ]
     )
     => r_menu / nil
```

Description

Creates a pulldown menu from the given menu definition. The *ls_menuDef* argument can either be a list defining the contents of a pulldown menu or the SKILL symbol that represents the list. This function exists primarily for application support.

If *x_windowNum* is specified, for example, `window->windowNum`, the menu handle and menu item symbols are uniquely created by appending the symbol in the menu definition with the window number.

Arguments

ls_menuDef	The definition of a menu.
x_windowNum	Window number.

Value Returned

r_menu	A menu suitable for use as a pulldown menu.
`nil`	A pulldown menu could not be created.

© 2020 All Rights Reserved.

ciwHiExit

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

Description

Exits the session.

This is the function beneath the *File –> Exit* menu in all the DFII procedural interfaces. Most of the *File –> Exit* menus in all DFII procedural interfaces can be accomplished using the `ciwHiExit` function.

By default, the command displays a dialog box asking you if you want to really exit the session. You can turn off the display of this dialog box by setting a variable in your `.cdsenv` file or by calling `envSetVal` in your `.cdsinit` file:

```
ddserv.ciw promptOnExit boolean nil
```

or

```
envSetVal ("ddserv.ciw" "promptOnExit" 'boolean nil)
```

Arguments

None

Value Returned

`t`	The session is terminated.
`nil`	The session is not terminated.

© 2020 All Rights Reserved.

ciwMenuInit

```
ciwMenuInit(
     [ ?verbose g_verbose ]
     [ ?reset g_reset ]
     )
     => t / nil
```

Description

Initializes and installs the CIW pulldown menus based on the menu files found in the hierarchy. This function will be called instead of `hiInstallCIWMenus`. It can also be called from the CIW to debug the menu file contents.

If *`g_reset`* is specified as `t`, the CIW menus are reset. Calling `ciwMenuInit` without any arguments will initialize the CIW menus only if it has not already been done.

Arguments

`?verbose` *`g_verbose`*	Specifies whether to echo extra messages during processing
`?reset` *`g_reset`*	Specifies whether to reset the CIW menus.

Value Returned

`t`	The function succeeds.
`nil`	The menu files in the hierarchy cannot be found.

Example

```
ciwMenuInit( ?verbose t )
ciwMenuInit( ?reset t )
```

When `ciwMenuInit` is called with the reset argument, the CIW menu is re-initialized and the latest customized menu is set.

© 2020 All Rights Reserved.

ciwMenuInstall

```
ciwMenuInstall(
     r_window
     ls_bannerDef
     )
     => t / nil
```

Description

Creates and installs into *r_window* a menu banner based on *ls_bannerDef*. Useful for applications that create and manage their own windows. This function exists primarily for application support.

Arguments

r_window	The window to install the menu banner into.
ls_bannerDef	The definition of the menu banner.

Value Returned

`t`	The function succeeds.
`nil`	A menu banner could not be created and installed because of the given banner menu definition.

© 2020 All Rights Reserved.

ciwMenuLoadFile

```
ciwMenuLoadFile(
     t_appName
     [ g_verbose ]
     )
     => t / nil
```

Description

Reads menu files found in the DFII hierarchy using *t_appName*.

Menu files will be read in the following order:

your_install_dir/etc/tools/menus/*appName*.menus
your_install_dir/local/menus/*appName*.menus
workOrProjectArea/menus/*appName*.menus
~/menus/*appName*.menus

Arguments

t_appName	The application name used for finding menu files.
g_verbose	Whether to echo extra messages during processing.

Value Returned

t	The function succeeded.
nil	No menu files are found.

© 2020 All Rights Reserved.

ciwRemoveTranslatorItem

```
ciwRemoveTranslatorItem(
     s_tag
     t_itemText
     t_callback
     )
     => t / nil
```

Description

Removes an entry from the `Import` or `Export` commands.

Arguments

s_tag	A SKILL symbol.
	Valid Values: `'Import`, `'Export`
t_itemText	The Item text to be used as a display prompt.

Value Returned

`t`	The entry was removed.
`nil`	The entry was not removed.

© 2020 All Rights Reserved.

ddsAboutDFII

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

Description

Opens the product information dialog box, which indicates the product, the release number, and copyright information. This dialog box is used for the custom IC design tools.

Arguments

None

Value Returned

`t`	Opens the product information dialog box.
`nil`	Unsuccessful in opening the product information dialog box.

© 2020 All Rights Reserved.

ddsBrowserAction

```
ddsBrowserAction(
     s_action
     )
     => t / nil
```

Description

Returns the Boolean value of the requested Library Browser option. This function explains how applications use the Library Browser preferences and the actions specified in the `ddsSyncWithForm` function. If you are integrating your own Library Browser, you must decide whether to support such actions; if so, you can use this function. This functionality provides applications the ability to support the user's need to see the Library Browser when a particular command is executed.

Arguments

`s_action` Action choices are *`'open`* (or *`'design`*), *`'place`*, *`'other`*, or your own symbols.

- `'open`
  (or `'design` for backward compatibility)
  If `ddsSyncWithForm` (or `dmbSyncWithForm`) is called with an action of `'open` (or `'design`), *`ddserv`* will start the Library Browser only if this option is set to `true`. If not, the Library Browser is not displayed.

- `'place`
  If `ddsSyncWithForm` (or `dmbSyncWithForm`) is called with an action of `'place`, *ddserv* will start the Library Browser only if this option is set to `true`. If not, the Library Browser is not displayed.

- `'other`
  If `ddsSyncWithForm` (or `dmbSyncWithForm`) is called with an action of `'other`, *ddserv* starts the Library Browser only if this option is set to `true`. If not, the Library Browser is not displayed.

© 2020 All Rights Reserved.

Value Returned

`t`	Any other SKILL symbol passed as an action (meaning that the Library Browser will be called).
`nil`	No library is selected or if the Library Browser is not synchronized with any particular form.

Example

If you set the preference for `'place` to be `t`, when you run the *Add-Instance* command from tools such as the layout editor or schematic editor, the Library Browser displays automatically rather than waiting for the *Browse* button to be selected.

© 2020 All Rights Reserved.

ddsCheckinPreferences

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

Description

Sets the prompt behavior you choose for the current DFII design session for checking in design data. This function displays a form to set the auto checkin prompt preferences for properties, files, and cellviews. This process controls access to the design data files that design and team members store in a project data repository.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The prompt behavior is not set.

© 2020 All Rights Reserved.

ddsCheckoutPreferences

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

Description

Sets the prompt behaviors you select for the current DFII design session for checking out design data. This function displays a form to set the auto checkout prompt preferences for properties, files, and cellviews. This process controls access to the design data files that design and team members store in a project data repository.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The prompt behavior is not set.

© 2020 All Rights Reserved.

ddsCheckSessionName

```
ddsCheckSessionName(
     t_sessionName
     )
     => t_sessionName
```

Description

Checks to see if the current MPS session is also the default session for the user. If the given session name is the same as the effective user ID's name or if there is no session name, builds a unique session name using the process ID. This implies that the current process checks the session and returns the session name to use - either the given name if it is checked or a new unique name to use to perform the checking.

Arguments

t_sessionName	The session name.

Value Returned

t_sessionName	The session name.

© 2020 All Rights Reserved.

ddsCloseLibManager

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

Description

Closes the Library Manager.

Note: Calling this function while the Library Manager is busy (with a copy, or rename, or check-in/check-out action) might have undesirable side effects.

Arguments

None

Value Returned

`t`	The Library Manager is closed.
`nil`	The Library Manager cannot be closed.

© 2020 All Rights Reserved.

ddsEditProp

```
ddsEditProp(
     t_libName
     t_cellName
     t_viewName
     )
     => propertyEditor
```

Description

Invokes the property editor on the named object. Calls `peEditProp` after getting the ID for the named object.

Arguments

t_libName	The name of the library. If *t_cellName* and *t_viewName* are blank, invokes the property editor on the library object.
t_cellName	The name of the cell or "" in the specified library. If *t_viewName* is blank, invokes the property editor on the cell object.
t_viewName	The name of the view or "" in the specified cell. Invokes the property editor on the cellview object.

Value Returned

Property Editor	Invokes the property editor on the named object.

© 2020 All Rights Reserved.

ddsEndSyncWithForm

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

Description

Terminates an application's form synchronization with the Library Browser and closes the Library Browser.

Double-check the code in your applications to be sure there is a call to this function to match your call to `ddsSyncWithForm`. Otherwise, your application might not be synchronized with the Library Browser.

Arguments

None

Value Returned

`t`	The function succeeds.
`nil`	No library is selected or the Library Browser is not synchronized with any particular form.

Reference

For additional information, refer to the browser synchronization section of the Custom Library Browser Template.

© 2020 All Rights Reserved.

ddsFileBrowseCB

```
ddsFileBrowseCB(
     g_form
     s_targetName
     [ t_filter ]
     [ s_mode ]
     [ t_caption ]
     )
     => t / nil
```

Description

Creates a file browser dialog for specifying file names through browsing.

The given name will be pasted into the target string field upon clicking the OK or Apply button. The dialog can filter files when first displayed. To navigate up or down in the directory structure, either double-click a directory name, or select it and click OK or Apply. The mode option allows checking for existing files.

Arguments

g_form	Accepts a form or a page from a tab field.
s_targetName	The full path name specified as a symbol.
t_filter	Optional string argument which filters file names by using an asterisk at the beginning. Default is no filter.
s_mode	The selection mode specified as a symbol with the following choices: ❑ `'anyFile`: selection is any file, whether it exists or not ❑ `'existingFile`: selection is a single file (default) ❑ `'directory`: selection must be a directory where both files and directories are displayed ❑ `'directoryOnly`: selection must be a directory where only directories are displayed **Note:** The `'existingFiles` mode is not supported.
t_caption	Optional argument that allows changing the title of the dialog box that is created. Default is `"Unix Browser"`.

© 2020 All Rights Reserved.

Value Returned

`t`	The dialog executes successfully.
`nil`	The dialog encounters an error during execution.

Example

Example 1

This example creates a Browse button to open an ASCII Technology file.

```
;; target file name field
tcAsciiTFStr = hiCreateStringField(
  ?name 'tcAsciiTFStr
  ?prompt "ASCII Technology File"
  ?value ""
)
;; Browse button for selecting an existing file
tcOpenBrowserBtn = hiCreateFormButton(
  ?name 'tcOpenBrowserBtn
  ?buttonText "Browse..."
  ?callback "ddsFileBrowseCB( hiGetCurrentForm()
    'tcAsciiTFStr \"*.tf\" 'existingFile )" ;open
)
```

Example 2

This example creates a browse button for selecting a library. The browse button is placed in a HI Tab field. The eval function has been used to seed the proper argument for ddsSyncWithForm.

```
libBrowseBtn = hiCreateFormButton(
   ?name 'libBrowseBtn
   ?buttonText "Browse..."
   ?callback "ddsSyncWithForm(eval(browseForm->tabs->page1) 'libname 'libname)"
  )
; Create a browse button for selecting a file. This browse button is placed in a
HI Tab field.
        field2 = hiCreateStringField( ?name 'filename ?prompt "File Name")
        fileBrowseBtn = hiCreateFormButton(
            ?name 'fileBrowseBtn
            ?buttonText "Browse..."
            ?callback "ddsFileBrowseCB(browseForm->tabs->page1 'filename)"
```


© 2020 All Rights Reserved.

```
        )

; Create a HI Tab field where the previous browse fields are placed within it.
        tabs = hiCreateTabField(
            ?name 'tabs
            ?notebookStyle 'nil
            ?tabPlacement 'top
            ?tabs list("TEST")
            ?value 1
            ?fields list( list(
                list(field1 0:0 200:0 100)
                list(libBrowseBtn 210:0 80:30)
                list(field2 0:40 200:0 100)
                list(fileBrowseBtn 210:40 80:30))
            )
        )

        hiCreateAppForm(
            ?name    'browseForm
            ?formTitle "Example Form"
            ?callback    ""
            ?dontBlock t
            ?fields    list(list(tabs 0:0 300:100))
        )
```


© 2020 All Rights Reserved.

ddsGetCurrentLib

```
ddsGetCurrentLib(
     )
     => b_ddLibId / nil
```

Description

Accesses the currently selected library in the Library Browser.

If your applications directly access the `dmbCurLib` variable or call `dmGetCurLib`, you need to review why. There is no concept of a current library. This function supports those applications that might be sharing information from a form although it might not matter what form the information is from. This function replaces `dmbGetCurrentLib`.

Arguments

None

Value Returned

b_ddLibId	The ddId of the library that is currently selected in the Library Browser if a form is registered to it.
`nil`	No library is selected or the Library Browser is not synchronized with any particular form.

Example

```
> x = ddsGetCurrentLib()
    dd:12347080
```


© 2020 All Rights Reserved.

ddsGetCustomLibSelect

```
ddsGetCustomLibSelect(
     )
     => t_name / nil
```

Description

Returns the name string passed to `ddsRegCustomLibSelect` if a custom Library Browser is registered to the system. If a custom Library Browser is not registered, `nil` is returned.

Arguments

None

Value Returned

t_name	The name as specified to the `ddsRegCustomLibSelect` function.
`nil`	Nothing is registered.

© 2020 All Rights Reserved.

ddsGetServiceName

```
ddsGetServiceName(
     )
     => t_serviceName
```

Description

Returns a string that can be used as an MPS service name. The string is the name of the invoked program plus the process ID number.

Arguments

None

Value Returned

t_serviceName	The MPS service name.

© 2020 All Rights Reserved.

ddsHiCloseData

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

Description

Displays the Close and Purge Data form that prompts you to select one or more objects to close and purge form virtual memory.

The Close and Purge Data form can display data that have lost connection to their physical files due to one of the following reasons:

- A network failure, due to which data on the shared file servers becomes inaccessible
- Deletion of local data
- Renaming of local data

The form lets you purge such data from the virtual memory. It displays relevant warnings in CIW.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The function did not succeed.

© 2020 All Rights Reserved.

ddsHiCreateLibrary

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

Description

Create new libraries from within DFII.

Arguments

None

Value Returned

`t`	A new library was created.
`nil`	No new library was created.

© 2020 All Rights Reserved.

ddsHiDefragCellView

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

Description

Displays a form that prompts you to specify the data set (library name, cell name, view name) to be de-fragmented. De-fragmentation is an operation to remove the free space in a given cellview or cellviews in a specified library to regain more disk and memory space.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The function did not succeed.

© 2020 All Rights Reserved.

ddsHiDefragLib

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

Description

Displays a form that prompts you to specify the library to be de-fragmented. Fragmentation is an operation to remove the free space in a given cellview or cellviews in a specified library to regain more disk and memory space.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The prompt behavior is not set.

© 2020 All Rights Reserved.

ddsHiMakeReadOnly

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

Description

Displays a form that prompts you to select cellviews currently open in edit mode and change them to read only mode.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The function did not succeed.

© 2020 All Rights Reserved.

ddsHiOpenCdsLibEditor

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

Description

Brings up the Library Path Editor.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The function did not succeed.

© 2020 All Rights Reserved.

ddsHiPreferences

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

Description

Invokes the Browser and CIW Preferences form.

Arguments

None

Value Returned

`t`	The function succeeded.
`nil`	The function did not succeed.

© 2020 All Rights Reserved.

ddsHiRefresh

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

Description

Opens a form listing the open technology files, properties, and cellviews that are not current with their corresponding files on disk.

Arguments

None

Value Returned

`t`	Data was refreshed from disk.
`nil`	No data was refreshed from disk.

© 2020 All Rights Reserved.

ddsLEWhatsNewAtStartup

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

Description

Brings up the What's New window for Virtuoso Layout Editor.

Arguments

None

Value Returned

`t`	The What's New window appears.
`nil`	The function failed.

© 2020 All Rights Reserved.

ddsOnLineHelp

```
ddsOnLineHelp(
     s_tag
     )
     => t / nil
```

Description

Brings up online Help documentation for DFII tools.

Arguments

s_tag The desired online Help to display in CDSHelp. Specify any of the following to display the respective documentation:

- `caiug`
  Cadence Application Infrastructure User Guide
- `main`
  CDSHelp main menu
- `cvtguide`
  Compatibility Guide
- `kpns`
  Known Problems and Solutions
- `relnote`
  Product Notes
- `window`
  Help for the currently active window, if any

Value Returned

`t`	Online Help documentation for DFII tools was opened.
`nil`	Online Help documentation for DFII tools was not opened.

© 2020 All Rights Reserved.

ddsOpenConvertToolBox

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

Description

Displays the Conversion Tool Box user interface.

Arguments

None

Value Returned

`t`	The Conversion Tool Box is displayed.
`nil`	The function failed.

© 2020 All Rights Reserved.

ddsOpenLibManager

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

Description

Starts the library manager. The separate process, library manager, is invoked and connected to using *mps*.

Note: When in replay, calls to this function will return. This is to not tie up the current process waiting for the external process to come up. The `ddserv` code is structured such that commands coming through it can be automatically replayed independent of the original external process.

Arguments

None

Value Returned

`t`	The library manager opens.
`nil`	The library manager is not opened.

© 2020 All Rights Reserved.

ddsProgramName

```
ddsProgramName(
     )
     => t_programName
```

Description

Returns the name of the program minus any `.exe` extension.

Arguments

None

Value Returned

t_programName	The program name.

© 2020 All Rights Reserved.

ddsRefresh

```
ddsRefresh(
     [ ?cellview { t | nil } ]
     [ ?cdf { t | nil } ]
     [ ?property { t | nil } ]
     [ ?techfile { t | nil } ]
     )
     => t / nil
```

Description

Refreshes open cellviews, CDFs, property bags, and technology files that are not current with their corresponding files on disk.

Note:

Arguments

`?cellview`	Determines whether cellview data needs to be refreshed. Default value: `t`
`?cdf`	Determines whether CDF data needs to be refreshed. Default value: `t`
`?property`	Determines whether property bag needs to be refreshed. Default value: `t`
`?techfile`	Determines whether technology files need to be refreshed. Default value: `nil`

Value Returned

`t`	The given cellviews, CDFs, property bags, and technology files were refreshed.
`nil`	All the cellviews, CDFs, property bags, and technology files are the same as their corresponding files on disk.

Example

```
ddsRefresh(?cellview t ?cdf t)
```


© 2020 All Rights Reserved.

In this example, all open cellviews and CDFs that are not current with their corresponding files on disk are refreshed.

© 2020 All Rights Reserved.

ddsRegPreRefreshTrigger

```
ddsRegPreRefreshTrigger(
     t_function
     )
     => t / nil
```

Description

Registers a user-defined SKILL function that is called before executing the Refresh command using *CIW - File - Refresh* or the ddsRefresh SKILL function.

Arguments

t_function	Name of the SKILL callback function. The callback function has the following two arguments: ■ *g_type* specifies the type of data being refreshed. Supported value: `'cv` ■ *l_dataList* lists IDs of the data that will be refreshed.

Value Returned

`t`	The specified function is registered successfully.
`nil`	The operation failed.

Example

```
; Registers the function myRefreshCB, which is called before the Refresh command
starts.

procedure( myRefreshCB( tag dataList )
    case( tag
        ('cv
            foreach( cv dataList
                info("**** %s %s %s will be refreshed\n"
                    cv~>libName cv~>cellName cv~>viewName)
        )
        (t
            info("**** Received unknown refresh tag - %L\n" tag)
```


© 2020 All Rights Reserved.

```
        )
    )
)
ddsRegPreRefreshTrigger( "myRefreshCB" )
```


© 2020 All Rights Reserved.

ddsRegPostRefreshTrigger

```
ddsRegPostRefreshTrigger(
     t_function
     )
     => t / nil
```

Description

Registers a user-defined SKILL function that is called after completion of the Refresh command using *CIW - File - Refresh* or the ddsRefresh SKILL function.

The user-defined SKILL function is passed a tag value of `'cv` or `'CDF` to indicate that a list of database IDs has been provided to the function for the cellview or CDF IDs, respectively.

Arguments

t_function	Name of the SKILL callback function. The callback function has the following three arguments: ■ *g_type* specifies the type of data being refreshed. Supported value: `'cv` ■ *l_good* lists IDs of the data that successfully refreshed successfully. ■ *l_bad* lists IDs the data that failed to refresh.

Value Returned

`t`	The specified function is registered successfully.
`nil`	The operation failed.

Example

```
; Registers the function myRefreshCB, which is called after the Refresh command is
complete.
procedure( myRefreshCB( tag goodList badList )
   case( tag
       ('cv
           foreach( cv goodList
               info("**** %s %s %s was refreshed\n"
```


© 2020 All Rights Reserved.

```
                    cv~>libName cv~>cellName cv~>viewName)
            )
            foreach( cv badList
                info("==== %s %s %s failed to refresh\n"
                    cv~>libName cv~>cellName cv~>viewName)
            )
        )
        (t
            info("**** Received unknown refresh tag - %L\n" tag)
        )
    )
)

ddsRegPostRefreshTrigger( "myRefreshCB" )
```


© 2020 All Rights Reserved.

ddsRegCustomLibSelect

```
ddsRegCustomLibSelect(
     t_name
     s_sync
     s_update
     s_endSync
     s_curlLib
     )
     => t / nil
```

Description

ddserv only supports one customization at a given time. If you call this function more than once, the latest set of functions are registered. The `ddsRegCustomSelect` function performs an isCallable Check on each of its arguments to insure that the functions have been loaded into the system. If not, a message is generated.

```
*Warning* ddsRegCustomLibSelect: sync routine name not callable.
*Warning* ddsRegCustomLibSelect: update routine name not callable.
*Warning* ddsRegCustomLibSelect: endSync routine name not callable.
*Warning* ddsRegCustomLibSelect: curLib routine name not callable.
```

The functions `ddsSyncWithForm`, `ddsUpdateSyncWithForm`, `ddsEndSyncWithForm`, and `ddsGetCurrentLib` will invoke the registered custom Library Browser interface functions using SKILL's *errset* functionality. If any SKILL error occurs in the interface function or in the custom Library Browser itself, SKILL generates a stack trace (if the `skill_stacktrace` variable has a value greater than 0) and control will return to the calling *ddserv* routine rather than jumping back to the SKILL top-level. This is to try to isolate the command the user is currently interacting with from problems in the Library Browser interface.

Arguments

t_name	A string to name the registration by.
s_sync	The SKILL symbol of the function that performs the `ddsSyncWithForm` action.
s_update	The SKILL symbol of the function that performs the `ddsUpdateSyncWithForm` action.
s_endSync	The SKILL symbol of the function that performs the `ddsEndSyncWithForm` action.

© 2020 All Rights Reserved.

s_curlLib	The SKILL symbol of the function that performs the `ddsCurrentLib` action.

Value Returned

`t`	All the specified routines can be called in the current DFII process.
`nil`	The specified routines cannot be called in the current DFII process.

Example

```
ddsRegCustomLibSelect( "dsb" 'dsbSyncForm 'dsbUpdateBrowsed 'dsbEndFormSync
'dbsGetCurrentLib )
```

To define these four custom functions refer to the extended example found in the Custom Library Browser Template.

© 2020 All Rights Reserved.

ddsHiSaveData

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

Description

Displays the Save Modified Data form that prompts you to select one or more objects to save from virtual memory.

Arguments

None

Value Returned

`t`	The Save Modified Data form is displayed.
`nil`	The Save Modified Data form is not displayed.

Example

Displays the Save Modified Data form.

```
ddsHiSaveData()
=>t
```


© 2020 All Rights Reserved.

ddsServSelect

```
ddsServSelect(
     t_lib
     t_cell
     t_view
     )
     => t
```

Description

Processes a select request from the browser process if and only if a form is synchronized to it. It is the call logged in the `cds.log` file. The function sets the appropriate form field with the library, cell, and view values.

Arguments

t_lib	The name of a library.
t_cell	The name of a cell.
t_view	The name of a view.

Value Returned

`t`	Always returns `t`.

© 2020 All Rights Reserved.

ddsSetLibManLCV

```
ddsSetLibManLCV(
     t_lib
     t_cell
     t_view
     )
     => t / nil
```

Description

Allows the application to set the library name, cell name, or view name entries in the library manager from a DFII session.

Arguments

t_lib	The name of a library.
t_cell	The name of a cell.
t_view	The name of a view.

Value Returned

`t`	Returns `t`, if the library manager is already open.
`nil`	Returns `nil`, if the library manager is not yet open.

Example

```
ddsSetLibManLCV( "library" "data" "layout" )
```


© 2020 All Rights Reserved.

ddsGetLibManLCV

```
ddsGetLibManLCV(
     )
     => l_libCellView
```

Description

Returns the library, cell, and view name entries currently selected in the library manager

Arguments

None

Value Returned

l_libCellViewName	Returns a list containing the current library, cell, and view selected in the library manager. If any of the entries from the library, cell, or view is not selected, the function returns an empty string (" ") for that entry. For example, if the library manager is not open, the return value will display ("" "" "").

Example

```
ddsGetLibManLCV( )
=> ("ether" "adc_sample_hold" "layout")
```


© 2020 All Rights Reserved.

ddsSyncWithForm

```
ddsSyncWithForm(
     g_form
     s_action
     s_libField
     [ s_cellField [ s_viewField [ l_viewTypes ] ] ]
     )
     => t / nil
```

Description

Registers the given form and fields of that form for synchronization with the Library Browser. The Library Browser is seeded with the current values of the library, cell, and view names from the fields in the form if they are not blank. The field value is not updated by `ddsSyncWithForm` when its enabled or editable attribute is set to `nil`. From this point until `ddsEndSyncWithForm`, selections in the Library Browser seed the given form fields.

Applications must call `ddsEndSyncWithForm` (or `dmbLibBrowserFormDone`, which is retained for compatibility) when they are done with the browser, using *Cancel* or *OK*. Not doing so could result in unusual browser interactions for the user.

Arguments

g_form Accepts a form or a page from a tab field.

s_action Specifies an action to further control starting the Library Browser

- `'design` or `'open` `ddserv.browser onOpen`: Opens automatically, when the Open Design form is displayed in the DFII software.
- `'place` `ddserv.browser onPlace`: Opens automatically, when a Place Instance form appears. (There are several types of Place Instance forms.)
- `'other` `ddserv.browser onOther`: Opens automatically, when any other type of linked form is displayed in the DFII software.

© 2020 All Rights Reserved.

s_libField	The symbol of the form text field that is to hold the name of the library selected. It is the same as the library form field name specified in `hiCreateForm`.
s_cellField	The symbol of the form text field that is to hold the cell name. It is the same as the cell form field name specified in `hiCreateForm`.
s_viewField	The symbol of the form text field that is to hold the view name. It is the same as the view form field name specified in `hiCreateForm`.
l_viewTypes	A list of viewType strings if *s_viewField* is specified. This list directs the Library Browser to display only views matching the list of view types when a cell is expanded.

Value Returned

`t`	The execution of the function succeeds.
`nil`	No library is selected or if the Library Browser is not synchronized with any particular form.

Example

Example 1

```
exampleLibName = hiCreateStringField(
        ?name 'exampleLibName
        ?prompt "Library Name"
        ?value ""
        ?callback "ddsUpdateSyncWithForm( )"
        )
exampleCellName = hiCreateStringField(
        ?name 'exampleCellName
        ?prompt "Cell Name"
        ?callback "ddsUpdateSyncWithForm( )"
        )
exampleViewName = hiCreateStringField(
        ?name 'exampleViewName
        ?prompt "View Name"
        ?callback "ddsUpdateSyncWithForm( )"
        )
exampleBrowser = hiCreateButtonBoxField(
        ?name 'exampleBrowser
        ?prompt " "
        ?choices '(" Browse ")
        ?callback list("exampleBrowser( )")
        )
hiCreateForm(
        'exampleOpenForm "Synched Form Example"
```


© 2020 All Rights Reserved.

```
        "exampleOpenForm( )"
        list( exampleLibName
              exampleCellName
              exampleViewName
                    exampleBrowser )
        ""
)
procedure( exampleBrowser( )
    ddsSyncWithForm( exampleOpenForm 'browse 'exampleLibName
             'exampleCellName'exampleViewName )
)
procedure( exampleOpenForm( )
    ddsSyncWithForm( exampleOpenForm 'other 'exampleLibName
        'exampleCellName 'exampleViewName )
    hiDisplayForm( 'exampleOpenForm )
)
    ddsEndSyncWithForm()
```

Example 2

This example creates a browse button for selecting a library. The browse button is placed in a HI Tab field. The `eval` function has been used to seed the proper argument for `ddsSyncWithForm`.

```
libBrowseBtn = hiCreateFormButton(
   ?name 'libBrowseBtn
   ?buttonText "Browse..."
   ?callback "ddsSyncWithForm(eval(browseForm->tabs->page1) 'libname 'libname)"
  )
; Create a browse button for selecting a file. This browse button is placed in a
HI Tab field.
        field2 = hiCreateStringField( ?name 'filename ?prompt "File Name")
        fileBrowseBtn = hiCreateFormButton(
            ?name 'fileBrowseBtn
            ?buttonText "Browse..."
            ?callback "ddsFileBrowseCB(browseForm->tabs->page1 'filename)"
        )

; Create a HI Tab field where the previous browse fields are placed within it.
        tabs = hiCreateTabField(
            ?name 'tabs
            ?notebookStyle 'nil
            ?tabPlacement 'top
            ?tabs list("TEST")
            ?value 1
            ?fields list( list(
                list(field1 0:0 200:0 100)
                list(libBrowseBtn 210:0 80:30)
```


© 2020 All Rights Reserved.

```
            list(field2 0:40 200:0 100)
            list(fileBrowseBtn 210:40 80:30))
        )
    )

    hiCreateAppForm(
        ?name    'browseForm
        ?formTitle "Example Form"
        ?callback    ""
        ?dontBlock t
        ?fields    list(list(tabs 0:0 300:100))
    )
```

Reference

For additional information, refer to the browser synchronization section of the Custom Library Browser Template.

© 2020 All Rights Reserved.

ddsUnregCustomLibSelect

```
ddsUnregCustomLibSelect(
     [ g_verbose ]
     )
     => t
```

Description

Un-registers the custom Library Browser interface during a session.

Argument

g_verbose	A message with the names of the functions it is registering.

Value Returned

t	Always returns t.

© 2020 All Rights Reserved.

ddsUpdateSyncWithForm

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

Description

If an application has registered a form with the Library Browser by calling `ddsSyncWithForm` or `dmbSyncWithForm`, this function updates the Library Browser with the values currently in the respective form fields.

This function provides complementary functionality when the library selector is normally seeding the form field values. Care is taken here and in the Library Browser to avoid recursive update attempts.

Arguments

None

Value Returned

`t`	The execution of the function succeeds.
`nil`	No library is selected or the Library Browser is not synchronized with any particular form.

Reference

For additional information, refer to the browser synchronization section of the Custom Library Browser Template.

© 2020 All Rights Reserved.

ddsViewDDProp

```
ddsViewDDProp(
     t_libName
     [ t_cellName [ t_viewName ] ]
     )
     => t_viewFile
```

Description

Finds information about the specified object and any properties found in its property bag.

Arguments

t_libName	The name of the library.
t_cellName	The name of the cell in the specified library.
t_viewName	The name of the view in the specified cell.

Value Returned

t_viewFile	The viewFile window to which the information is printed.

© 2020 All Rights Reserved.

ddsWhatsNew

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

Description

Brings up the What's New window.

Arguments

None

Value Returned

`t`	The What's New window appears.
`nil`	The function failed.

© 2020 All Rights Reserved.

Custom Library Browser Template

The following may be used as a template for a custom Library Browser integration (`browser` in this template refers to the Library Browser).

```
;;*************************************************************************
;;   The sample syncronization routines for the sample browser integration.
;;         1998 Cadence Design systems, Inc. - All Rights Reserved.
;;
;; The code below is for example only and is not intended for production
;; usage by customers. Users integrating their own custom browsers with DFII
;; may use this file as a template for their own integration.
;;
;; The sample browser uses the prefix "dsb" for DAG-Simple-Browser. This
;; browser only supports the expansion of a library, its cells, and views of
;; a selected cell. Category display is not supported. There is no intention
;; of extending the capabilities of this browser any further nor to make the
;; source available except for what is listed here.
;;*************************************************************************

;; The following global variable is used to designate when the browser is
;; doing a selection of a library, cell, or cellview. When 't, the code will
;; know not to try re-expand itself in case the form currently in sync has
;; field callbacks that call ddsUpdateSyncWithForm (which ends up calling
;; the dsbUpdateBrowser routine below).

dsbvIsSelecting = nil


;;*************************************************************************
;; dsbSyncForm - This is the routine registered via ddsRegCustomLibSelect
;; to syncronize a form to the dsb. It takes the same arguments as the
;; ddsSyncWithForm routine. From this call until the dsbEndFormSync routine
;; is called, the browser will seed or be seeded from the given form
;; fields. Applications are required to call ddsEndSyncWithForm when they
;; are done with a browser otherwise the user may experience strange
;; browser behavior.
;;
;; dsb does not support the filtering of cellviews by viewType so the
;; viewTypes arg from ddsSyncWithForm is ignored. If viewType filtering
;; were to be supported, the browser would only display those cellviews
;; whose viewType were in the given list. To get a cellview's viewType,
;; you would call ddGetObj to get the master file in the cellview, and
;; then ddMapGetFileViewType to get the viewType. For example,
;;
;;      id = ddGetObj(lib cell view "*") ; '*' gets the master file
;;      viewType = ddMapGetFileViewType(id)
;;
;; Some forms within DFII only syncronize the library field so it is the
;; only required field from ddsSyncWithForm. Some forms only syncronize the
;; library and cell fields (e.g., the CDF editor). This routine and the
;; updateBrowser routine must be aware of this.
;;*************************************************************************

procedure( dsbSyncForm(form action libField
                    cellField viewField _viewTypes "rssggl")
    prog((fn)

        ;; Verify that the specified fields exist on the given form.
        ;; This is extra precaution against callers passing in misstyped
```


© 2020 All Rights Reserved.

```
        ;; field names. We also check that if a viewField is given, a
        ;; valid cellField is also given.

        fn = "dsbSyncForm"
        unless(get(form libField)
            warn("%s: %L not a field on form %L\n" fn libField form->hiFormSym)
            return()
        )
        when(cellField && !get(form cellField)
            warn("%s: %L not a field on form %L\n" fn cellField form->hiFormSym)
        )
        when(viewField && !get(form viewField)
            warn("%s: %L not a field on form %L\n" fn viewField form->hiFormSym)
        )
        when(viewField && !cellField
            warn("%s: cellField required when specifying a viewField\n" fn)
            return()
        )

        ;; Validate the browser action. If the browser action is one of
        ;; 'open, 'place, or 'other and the user has set the corresponding
        ;; ddserv environment variable to 'nil, the routine will return
        ;; at this point. If the user specifies a value of 't, the code
        ;; continues on and opens the browser. This control allows some
        ;; specific applications to support bringing up the browser when
        ;; specific commands are invoked.

        unless(ddsBrowserAction(action)
            return(t)
        )

        ;; Open the browser and save the form and field symbols.
        ;; Then call the update routine to expand the browser to anything
        ;; the user may have entered in these fields on the form.
        ;;
        ;; dsbvBrowser is a global variable that contains all the
        ;; information for the dsb. It is only defined/bound when the
        ;; browser is open; it does not exist otherwise.

        dsbOpenBrowser()
        dsbvBrowser->form = form
        dsbvBrowser->lib  = libField
        dsbvBrowser->cell = cellField
        dsbvBrowser->view = viewField
        dsbUpdateBrowser()
        return(t)
    )
)

;;*************************************************************************
;; dsbEndFormSync - Terminates syncronization between the browser and the
;; form. This is called when ddsEndSyncWithForm is called. When the browser
;; is closed, the dsbvBrowser variable is purged and marked unbound; this
;; clears any form and field information as well so it is not necessary to
;; explicitly do it here. If the form and field information were stored
;; differently, we would clear out the information to signify the browser
;; wasn't sync'd to any form.
;;
;; This and the other routines below only act if a) there is a browser
;; open and b) there is a form syncronized to it.
```


© 2020 All Rights Reserved.

```
;;***************************************************************************

procedure( dsbEndFormSync()
    when( boundp('dsbvBrowser) && dsbvBrowser->form
        dsbCloseBrowser()
    )
    t
)

;;***************************************************************************
;; dsbGetCurrentLib - Returns the current library selected in the browser.
;; This is called when ddsGetCurrentLib is called. The browser code sets
;; the dsbvCurrentLib global variable whenever the library selection
;; changes. This is possible because only 1 library can be expanded at any
;; given time.
;;***************************************************************************

procedure( dsbGetCurrentLib()
    when( boundp('dsbvBrowser) && dsbvBrowser->form
        dsbvCurrentLib
    )
)

;;***************************************************************************
;; dsbUpdateBrowser - This is the workhorse routine that expands/updates the
;; browser with the values from the registered form fields. Some apps only
;; register the lib or lib & cell fields.
;;***************************************************************************

procedure( dsbUpdateBrowser()
    cond(
        ;;
        ;; If there isn't a browser around or there isn't a form registred
        ;; or we got here because the browser did a selection and a call-
        ;; back for the form is doing an update, we return.
        ;;
        (!boundp('dsbvBrowser) || !dsbvBrowser->form || dsbvIsSelecting
            t
        )
        ;;
        ;; Some apps (more historical from the 4.3.4 days) call the update
        ;; routine in their cancel callbacks. Since the form should be
        ;; going away, there really isn't anything to do as the next thing
        ;; that should happen is a ddsEndSyncWithForm call.
        ;;
        (hiIsInFieldCancel()
            t
        )
        ;;
        ;; Get any lib/cell/view values from the respective form fields.
        ;; The assumption has always been (from 4.3.4) that these fields
        ;; are string fields. If other types of fields were to be supported,
        ;; the code below would have to be able to process them.
        ;;
        ;; The routine dsbExpandFromForm is a special routine that expands
        ;; the browser to a given lib/cell/view. If all three values are
        ;; 'nil, it only expands the libraries pruning back the display
        ;; if it has to. A different browser may have to have a different
        ;; behavior, but isolating it to a single function call helps
        ;; keep it modular.
```


© 2020 All Rights Reserved.

```
        ;;
        (t
            let( (tool lib cell view cellF viewF)
                tool = dsbvBrowser
                lib  = get(tool->form tool->lib)->value
                when( lib
                    lib = (when (lib && lib != "") lib)

                    when( tool->cell && (cellF = get(tool->form tool->cell))
                        cell = (when (cellF->value && cellF->value != "")
                                        cellF->value)
                    )

                    when( cell && tool->view &&
                            (viewF = get(tool->form tool->view))
                        view = (when (viewF->value && viewF->value != "")
                                        viewF->value)
                    )

                    dsbExpandFromForm(tool lib cell view)
                )
            )
            t
        )
    )
)

;;*************************************************************************
;; dsbServeSelect - This routine is called by the browser code whenever a
;; library, cell, or cellview is selected. It is listed here along with the
;; others as an example. It's purpose is complimentary to the above
;; dsbUpdateBrowser routine; it fills in the registered form fields with
;; the selection in the browser.
;;
;; The variable dsbvIsSelecting is first set to 't before updating the
;; form field value. In the case where the form field has a callback that
;; calls ddsUpdateSyncWithForm, the update routine above will not try to
;; expand the form again since it was already expanded by the user. This
;; variable is reset once the fields have been updated.
;;
;; The code below is specific to the dsb but conceptually, it gets
;; the current library, cell, and cellview selections and sets the
;; respective form field values.
;;*************************************************************************

procedure( dsbServeSelect(obj fromWhere "ws")
    let( (tool node libF cellF viewF tmp)
        tool = dsbvBrowser
        when(obj && tool->form
            dsbvIsSelecting = t
            caseq( fromWhere
                (lib
                    libF = get(tool->form tool->lib)
                    libF->value = obj->name
                    when( (cellF = get(tool->form tool->cell))
                        node = dsbGetExpandNode(tool->expandPath
                                                tool->cellClass)
                        unless(node
                            dsbGetExpandNode(tool->expandPath
                                                 tool->cellviewClass)
```


© 2020 All Rights Reserved.

```
                        )
                        cellF->value = (if node node->name "")
                    )
                    when( (tool->view && (viewF = get(tool->form tool->view)))
                        node = dsbGetExpandNode(tool->expandPath
                                                    tool->cellviewClass)
                        unless(node
                            dsbGetExpandNode(tool->expandPath
                                                tool->cellviewClass)
                        )
                        viewF->value = (if node node->name "")
                    )
                )
                (cell
                    tmp  = dsbiGetLibParent(obj)
                    libF = get(tool->form tool->lib)
                    libF->value = tmp->name
                    when( (cellF = get(tool->form tool->cell))
                        cellF->value = obj->name
                    )
                )
                (view
                    tmp  = dsbiGetLibCellview(obj)
                    libF = get(tool->form tool->lib)
                    libF->value = car(tmp)
                    when( (cellF = get(tool->form tool->cell))
                        cellF->value = cadr(tmp)
                    )
                    when( (viewF = get(tool->form tool->view))
                        viewF->value = caddr(tmp)
                    )
                )
            )
            dsbvIsSelecting = nil
        )
    )
)
```

The following is a form example exercising the browser synchronization.

```
;;
;; This file contains a command that takes library, cell, view, and
;; mode values and performs an open using deOpen. The associated form
;; also contains a 'Browse' button to exercise the custom library
;; browser integration although it will also work with DFII's default
;; browser as well.
;;
procedure(myOpen()
    if( boundp('myOpenForm) && hiIsForm(myOpenForm) then
        hiSetCurrentField(myOpenForm 'libName)
    else
        let((itemL)
            itemL = tconc(itemL list(hiCreateStringField(
                ?name           'libName
                ?prompt         "Library"
                ?value          ""
                ?callback       "myOpenLibCB()"
            ) 5:0 200:30 60))
            itemL = tconc(itemL list(hiCreateStringField(
                ?name           'cellName
                ?prompt         "Cell"
```


© 2020 All Rights Reserved.

```
                ?value          ""
                ?callback       "myOpenCellCB()"
            ) 5:30 200:30 60))
            itemL = tconc(itemL list(hiCreateStringField(
                ?name           'viewName
                ?prompt         "View"
                ?value          ""
                ?callback       "myOpenViewCB()"
            ) 5:60 200:30 60))
            itemL = tconc(itemL list(hiCreateRadioField(
                ?name           'mode
                ?prompt         "Mode"
                ?choices        '("edit" "read")
                ?value          "read"
            ) 5:90 200:30 60))
            itemL = tconc(itemL list(hiCreateFormButton(
                ?name           'browse
                ?buttonText     "Browse"
                ?callback       "myOpenBrowseCB()"
            ) 5:120 60:25))

            hiCreateAppForm(
                ?name               'myOpenForm
                ?formTitle          "Open Cellview"
                ?fields             car(itemL)
                ?help               ""
                ?buttonLayout       'OKCancelApply
                ?formType           'nonOptions
                ?callback           '("myOpenCB('ok)" "myOpenCB('cancel)")
                ?unmapAfterCB       t
            )
        )
    )

    ddsSyncWithForm(myOpenForm 'open 'libName 'cellName 'viewName)
    hiDisplayForm(myOpenForm)
)

procedure( myOpenBrowseCB()
    ddsSyncWithForm(myOpenForm 'browse 'libName 'cellName 'viewName)
)

procedure( myOpenLibCB()
    let( (form libN)
        form  = myOpenForm
        libN  = form->libName->value

        hiHighlightField(form 'cellName hicBackground)
        hiHighlightField(form 'viewName hicBackground)
        when( libN != "" && ddGetObj(libN)
            hiHighlightField(form 'libName hicBackground)
        )
        ddsUpdateSyncWithForm()
    )
)

procedure( myOpenCellCB()
    let( (form libN cellN)
        form  = myOpenForm
        libN  = form->libName->value
        cellN = form->cellName->value
```


© 2020 All Rights Reserved.

```
        hiHighlightField(form 'libName hicBackground)
        hiHighlightField(form 'viewName hicBackground)
        when( libN != "" && cellN != "" && ddGetObj(libN cellN)
            hiHighlightField(form 'cellName hicBackground)
        )
        ddsUpdateSyncWithForm()
    )
)

procedure( myOpenViewCB()
    let( (form libN cellN viewN)
        form  = myOpenForm
        libN  = form->libName->value
        cellN = form->cellName->value
        viewN = form->viewName->value

        hiHighlightField(form 'libName hicBackground)
        hiHighlightField(form 'cellName hicBackground)
        when( libN != "" && cellN != "" && viewN != "" &&
                ddGetObj(libN cellN viewN)
            hiHighlightField(form 'viewName hicBackground)
        )
        ddsUpdateSyncWithForm()
    )
)

procedure( myOpenCB(okOrCan "s")
    prog( (form fspec libN cellN viewN id vType)

        when(eq(okOrCan 'cancel)
            ddsEndSyncWithForm()
            return()
        )
        form  = myOpenForm
        libN  = form->libName->value
        cellN = form->cellName->value
        viewN = form->viewName->value
        cond(
            (libN == ""
                hiSetCallbackStatus(form nil)
                hiSetCurrentField(form 'libName)
                hiHighlightField(form 'libName hicError)
                myOpenDBox("Need Library Name" "enter a library name.")
                return()
            )
            (cellN == ""
                hiSetCallbackStatus(form nil)
                hiSetCurrentField(form 'cellName)
                hiHighlightField(form 'cellName hicError)
                myOpenDBox("Need Cell Name" "enter a cell name.")
                return()
            )
            (viewN == ""
                hiSetCallbackStatus(form nil)
                hiSetCurrentField(form 'viewName)
                hiHighlightField(form 'viewName hicError)
                myOpenDBox("Need View Name" "enter a view name.")
                return()
            )
            (!ddGetObj(libN cellN viewN)
```


© 2020 All Rights Reserved.

```
                hiSetCallbackStatus(form nil)
                hiSetCurrentField(form 'viewName)
                hiHighlightField(form 'viewName hicError)
                myOpenDBox("Not Found"
                    sprintf(nil strcat("Cellview '%s/%s/%s' does not exist.\n"
                                       "check your entries.")
                                       libN cellN viewN))
                return()
            )
        )
        hiSetCallbackStatus(form t)
        hiFormUnmap(form)
        ddsEndSyncWithForm()

        fspec = list(nil
                    'libName        libN
                    'cellName       cellN
                    'viewName       viewN
                    'viewType       nil
                    'accessMode     form->mode->value)

        return( deOpen(fspec nil) )
    )
)

procedure( myOpenDBox(banner msg "tt")
    hiDisplayAppDBox(
        ?name               'myDBox
        ?dboxBanner         banner
        ?dboxText           msg
        ?dialogType         hicErrorDialog
        ?buttonLayout       'Close
    )
)
```


© 2020 All Rights Reserved.

dmb Compatibility Functions

These functions are provided for backward compatibility. If you are writing new code, use the dds functions from the previous section.

dmbGetCurrentLib

```
dmbGetCurrentLib(
     )
     => ddId / nil
```

Description

Returns the `ddId` of the library currently selected in the Library Browser if a form is registered to it (otherwise, the Library Browser is hidden).

Note: There is nothing in the system that supports the notion of a “current” library.

Supplied for compatibility to support those applications that may be sharing the information from a form (although it might not matter what form it is). Replace with `ddsGetCurrentLib`.

Arguments

None

Value Returned

`ddId`	The `ddId` of the library that is currently selected in the Library Browser.
`nil`	No library is selected or the Library Browser is not synchronized with any particular form.

Example

```
>   x = dmbGetCurrentLib()
    dd:12347080
```

The Library Browser is displayed and a library is selected for backward compatibility.

© 2020 All Rights Reserved.

dmbLibBrowserFormDone

```
dmbLibBrowserFormDone(
     s_formtype
     )
     => t
```

Description

Terminates the synchronization between the last form registered and the Library Browser. Applications should ALWAYS call this function when they know their Library Browser/form interaction has terminated (using *Cancel* or *OK*). Leaving it hanging could result in unpredictable behavior which would degrade the usability of the system.

Supplied for compatibility. Replace with `ddsEndSyncWithForm`.

Prerequisite

Call `dmbLibBrowserFormDone` after the `hiDisplayForm` call that brings up the DFII software form. Refer to the example under `dmbSyncWithForm`.

Arguments

`s_formtype` Identifies the type of form that is invoking the Library Browser:

- `'design`
  The Open Design form from the Design Framework II Open menu.
- `'place`
  A Place Instance form from an editor. (There are several types of Place Instance forms.)
- `'other`
  Any other type of linked form.

© 2020 All Rights Reserved.

Value Returned

`t`	Execution of the function succeeded.

Example

```
dmbLibBrowserFormDone( 'design )
```


© 2020 All Rights Reserved.

dmbOpenLibBrowser

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

Description

Invokes the library manager. The separate process, library manager, is invoked and connected to using *mps*.

Supplied for compatibility. Replace with `ddsOpenLibManager`.

Arguments

None

Value Returned

`t`	The library manager is opened.
`nil`	The library manager is not opened.

© 2020 All Rights Reserved.

dmbSetBrowserInteractions

```
dmbSetBrowserInteractions(
     [ ?openDesignOption { t | nil } ]
     [ ?openPlaceInstanceOption {t | nil} ]
     [ ?openOthersOption { t | nil } ]
     )
     => t / nil
```

Description

Specifies whether the Library Browser is automatically opened when certain types of forms appear. This function is provided for backward compatibility with only the 4.3.4 version.

You should remove any `.cdsinit` calls and replace with the following `.cdsenv` variables (in the `.cdsenv` file):

```
ddserv.browser              onOpen
ddserv.browser              onPlace
ddserv.browser              onOther
```

Or, you can explicitly call `envSetVal` in your `.cdsinit` file:

```
envSetVal( "ddserv.browser" "onOpen"  'boolean nil)
envSetVal( "ddserv.browser" "onPlace" 'boolean t)
```

Arguments

`?openDesignOption`

Defines interactions between the Open Design form and the Library Browser.

When set to `t`, the Library Browser is opened automatically when this form is opened.

`?openPlaceInstanceOption`

Defines interactions between any Place Instance form and the Library Browser.

When set to `t`, the Library Browser is opened automatically when this form is opened.

`?openOthersOption`

Defines interactions between any other type of linked form and the Library Browser.

When set to `t`, the Library Browser is opened automatically when this form is opened.

© 2020 All Rights Reserved.

Note: All the arguments for closing are no longer applicable.

Value Returned

`t`	Execution of the function succeeded.
`nil`	Execution of the function failed if any of the input parameters is invalid.

Example

```
dmbSetBrowserInteractions( ?openOthersOption t)
```


© 2020 All Rights Reserved.

dmbSyncWithForm

```
dmbSyncWithForm(
     s_form s_synctype
     s_libraryName
     s_cellName
     s_viewName
     [ s_versionNumber ]
     )
     => t / nil
```

Description

Establishes synchronization between a given form and the Library Browser. The given library, cell, and view fields are filled in as you select items in the Library Browser.

Any form in the DFII software can be linked to the Library Browser. Linking ensures that the library, cell, view, and version fields of the corresponding form and the selected expansion in the Library Browser window are synchronized. One of these two functions should be called immediately before bringing up any form that is to be synchronized with the Library Browser. The last four parameters to these functions dictate which fields in the form are synchronized with the Library Browser when the `dmbUpdateBrowser` function is invoked. You can synchronize only one form at a time with the Library Browser.

`dmbSyncWithForm` and `dmbSyncWithCellForm` are identical, except that *`s_viewName`* is required in `dmbSyncWithForm` and is optional in `dmbSyncWithCellForm`.

Supplied for compatibility. Replace both with `ddsSyncWithForm`.

Arguments

`s_form` Name of the form that is calling the Library Browser.

`s_synctype` Specifies an action to further control the use of the Library Browser. Defines how the Library Browser is called:

- `'design`
  Opens automatically when the Open Design form is displayed in the DFII software.
- `'place`
  Opens automatically when the Place Instance form appears. (There are several types of Place Instance forms.)

© 2020 All Rights Reserved.

- `'other`
  Opens automatically when any other type of linked form is displayed in the DFII software.
- `'browse`
  Opens manually by selecting the *Browse* button from one of the above three types of forms.

s_libraryName	Represents the symbol of the form text field that is to hold the name of the library selected. This is the same as the library form field name specified in `hiCreateForm`.
s_cellName	The symbol of the form text field that is to hold the cell name. It is the same as the cell form field name specified in `hiCreateForm`.
s_viewName	The symbol of the form text field that is to hold the view name. It is the same as the view form field name specified in `hiCreateForm`.
s_versionNumber	The version number form field name specified in `hiCreateForm`.

Value Returned

`t`	The function succeeds.
`nil`	No library is selected or the Library Browser is not synchronized with any particular form.

Example

```
exampleLibName = hiCreateStringField(
        ?name 'exampleLibName
        ?prompt "Library Name"
        ?value ""
        ?callback "dmbUpdateBrowser( )"
        )
exampleCellName = hiCreateStringField(
        ?name 'exampleCellName
        ?prompt "Cell Name"
        ?callback "dmbUpdateBrowser( )"
        )
exampleViewName = hiCreateStringField(
        ?name 'exampleViewName
        ?prompt "View Name"
        ?callback "dmbUpdateBrowser( )"
        )
exampleVersionNumber = hiCreateStringField(
        ?name 'exampleVersionNumber
```


© 2020 All Rights Reserved.

```
        ?prompt "Version Number"
        ?callback "dmbUpdateBrowser( )"
        )
exampleBrowser = hiCreateButtonBoxField(
        ?name ’exampleBrowser
        ?prompt " "
        ?choices ’(" Browse ")
        ?callback list("exampleBrowser( )")
        )
hiCreateForm(
        ’exampleOpenForm "Synched Form Example"
        "exampleOpenForm( )"
        list( exampleLibName
              exampleCellName
              exampleViewName
              exampleVersionNumber
              exampleBrowser )
        ""
)
procedure( exampleBrowser( )
    dmbSyncWithForm( exampleOpenForm ’browse ’exampleLibName
             ’exampleCellName’exampleViewName
     ’exampleVersionNumber )
)
procedure( exampleOpenForm( )
    dmbSyncWithForm( exampleOpenForm ’other ’exampleLibName
        ’exampleCellName ’exampleViewName
        ’exampleVersionNumber)
    hiDisplayForm( ’exampleOpenForm )
    dmbLibBrowserFormDone( ’other )
)
```


© 2020 All Rights Reserved.

dmbSyncWithCellForm

```
dmbSyncWithCellForm(
     s_form
     s_synctype
     s_libraryName
     s_cellName
     [ s_viewName ]
     [ s_versionNumber ]
     )
     => t / nil
```

Description

Establishes synchronization between a given form and the Library Browser. The given library, cell, and view fields are filled in as you select items in the Library Browser.

Any form in the DFII software can be linked to the Library Browser. Linking ensures that the library, cell, view, and version fields of the corresponding form and the selected expansion in the Library Browser window are synchronized. One of these two functions should be called immediately before bringing up any form that is to be synchronized with the Library Browser. The last four parameters to these functions dictate which fields in the form are synchronized with the Library Browser when the `dmbUpdateBrowser` function is invoked. You can synchronize only one form at a time with the Library Browser.

`dmbSyncWithForm` and `dmbSyncWithCellForm` are identical, except that *`s_viewName`* is required in `dmbSyncWithForm` and is optional in `dmbSyncWithCellForm`.

Supplied for compatibility. Replace both with `ddsSyncWithForm`.

Arguments

`s_form`	Name of the form that is calling the Library Browser.
`s_synctype`	Specifies an action to further control the use of the Library Browser. Defines how the Library Browser is called: ❑ `'design` Opens automatically when the Open Design form is displayed in the DFII software. ❑ `'place` Opens automatically when the Place Instance form appears. (There are several types of Place Instance forms.)

© 2020 All Rights Reserved.

- `'other`
  Opens automatically when any other type of linked form is displayed in the DFII software.
- `'browse`
  Opens manually by selecting the *Browse* button from one of the above three types of forms.

s_libraryName	Represents the symbol of the form text field that is to hold the name of the library selected. This is the same as the library form field name specified in `hiCreateForm`.
s_cellName	The symbol of the form text field that is to hold the cell name. It is the same as the cell form field name specified in `hiCreateForm`.
s_viewName	The symbol of the form text field that is to hold the view name. It is the same as the view form field name specified in `hiCreateForm`.
s_versionNumber	The version number form field name specified in `hiCreateForm`.

Value Returned

`t`	The function succeeds.
`nil`	No library is selected or the Library Browser is not synchronized with any particular form.

Example

```
exampleLibName = hiCreateStringField(
        ?name 'exampleLibName
        ?prompt "Library Name"
        ?value ""
        ?callback "dmbUpdateBrowser( )"
        )
exampleCellName = hiCreateStringField(
        ?name 'exampleCellName
        ?prompt "Cell Name"
        ?callback "dmbUpdateBrowser( )"
        )
exampleViewName = hiCreateStringField(
        ?name 'exampleViewName
        ?prompt "View Name"
        ?callback "dmbUpdateBrowser( )"
        )
exampleVersionNumber = hiCreateStringField(
        ?name 'exampleVersionNumber
```


© 2020 All Rights Reserved.

```
        ?prompt "Version Number"
        ?callback "dmbUpdateBrowser( )"
        )
exampleBrowser = hiCreateButtonBoxField(
        ?name 'exampleBrowser
        ?prompt " "
        ?choices '(" Browse ")
        ?callback list("exampleBrowser( )")
        )
hiCreateForm(
        'exampleOpenForm "Synched Form Example"
        "exampleOpenForm( )"
        list( exampleLibName
              exampleCellName
              exampleViewName
              exampleVersionNumber
              exampleBrowser )
        ""
)
procedure( exampleBrowser( )
    dmbSyncWithForm( exampleOpenForm 'browse 'exampleLibName
             'exampleCellName'exampleViewName
     'exampleVersionNumber )
)
procedure( exampleOpenForm( )
    dmbSyncWithForm( exampleOpenForm 'other 'exampleLibName
        'exampleCellName 'exampleViewName
        'exampleVersionNumber)
    hiDisplayForm( 'exampleOpenForm )
    dmbLibBrowserFormDone( 'other )
)
```


© 2020 All Rights Reserved.

dmbUpdateBrowser

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

Description

Updates the library browser with the values currently in the registered form *formFields*. This function lets applications update the Library Browser based on what users type into a form.

Any form in the DFII software can be linked to the Library Browser using the `dmbSyncWithForm` or `dmbSyncWithCellForm` function. After you invoke the `dmbUpdateBrowser` function for the form, you no longer need to type values on the form for the library, cell, view, and (optionally) version. Instead, you can select from graphic choices displayed in the Library Browser window, and the selected value will automatically display in the appropriate name field on the form. Conversely, typing a value in a name field on the form causes that typed name to be selected in the Library Browser window.

Supplied for compatibility. Replace with `ddsUpdateSyncWithForm`.

Arguments

None

Value Returned

`t`	Execution of the function succeeded.
`nil`	No library is selected or the Library Browser is not synchronized with any particular form.

Example

```
exampleLibName = hiCreateStringField(
        ?name 'exampleLibName
        ?prompt "Library Name"
        ?value ""
        ?callback "dmbUpdateBrowser( )"
        )
exampleCellName = hiCreateStringField(
        ?name 'exampleCellName
        ?prompt "Cell Name"
        ?callback "dmbUpdateBrowser( )"
        )
```


© 2020 All Rights Reserved.

```
hiCreateForm(
        'exampleOpenForm "Synched Form Example"
        "exampleOpenForm( )"
        list( exampleLibName
              exampleCellName )
        ""

)
procedure( exampleBrowser( )
    dmbSyncWithCellForm( exampleOpenForm 'browse 'exampleLibName 'exampleCellName)
)
procedure( exampleOpenForm( )
    dmbSyncWithCellForm( exampleOpenForm 'other
    'exampleLibName 'exampleCellName )
    hiDisplayForm( 'exampleOpenForm )    dmbLibBrowserFormDone( 'other )
)
```


© 2020 All Rights Reserved.

Cell Category Functions (ddCat Interface)

The files used to store the category information are in a special format developed for use with the Cadence Team Design Manager (TDM).

If you copy a library containing categories with the intent of using a new library name, the category information stored in the `.Cat` and `.TopCat` files under the library will require updating because the library name is stored in these files as part of the data.

These files are simple text files which can be manipulated using `vi`, `sed`, or `awk`. The `.TopCat` file lists the top level categories for the library and each `.Cat` lists category contents. Items in these files are of the form

```
library_name / cell_name type="cell"
```

or

```
library_name> / category_name.Cat type="category"
```

To update the library name to the new library name, edit the files from UNIX like this:

```
% foreach cat ( *.TopCat *.Cat )
> mv $cat $cat.tmp
> sed -e 's/^library_name/new_name/' $cat.tmp > $cat
> end
```


© 2020 All Rights Reserved.

ddCatOpen

```
ddCatOpen(
     g_ddId
     t_categoryName
     t_mode
     )
     => b_objId / nil
```

Description

Opens a category with the given name and uses the mode to get the object ID.

You can use `ddCatGetLibCats` to get the names of the categories in a library. With this list of names, you can use `ddCatGetCatMembers` to access the contents of each category. You can use subsequent calls to `ddCatOpen` and `ddCatGetCatMembers` to process hierarchical categories.

Like DFII design management, this interface does not let you create new categories if they already exist. This prevents recursive category specification within a library.

Arguments

g_ddId	The library ID or category ID (for hierarchical categories). If the object is a *libId* and the given name is " " or `nil`, the library's list of categories can be accessed or modified.
t_categoryName	The name of the category.
t_mode	The modes "`a`", "`w`", or "`r`". See ddGetObj.

Value Returned

b_objId	The ID for the opened category.
`nil`	The category does not exist or cannot be opened in the specified mode.

Example

```
cat = ddCatOpen(libId "myCat" "a")
ddCatAddItem(cat "cell1" "cell")
ddCatSave(cat)
ddCatClose(cat)
```


© 2020 All Rights Reserved.

Updates (or creates) a category and adds a cell name to it.

© 2020 All Rights Reserved.

ddCatOpenEx

```
ddCatOpenEx(
     g_ddId
     t_categoryName
     t_mode
     x_keepEmpty
     )
     => b_objId / nil
```

Description

Opens a category with the given name using the mode to get the object ID.

Similar to `ddCatOpen` except that this function has an additional fourth argument indicating that a flag should be set in the category file so that the dd interface will not remove the category file when it is empty.

Arguments

g_ddId	The library ID or category ID (for hierarchical categories). If the object is a *libId* and the given name is " " or `nil`, the library's list of categories can be accessed or modified.
t_categoryName	The name of the category.
t_mode	The modes "`a`", "`w`", or "`r`". See ddGetObj.
x_keepEmpty	When *x_keepEmpty* is `1`, the category files are not deleted, but, when *x_keepEmpty* is `0`, the category files are deleted.

Value Returned

b_objId	The ID for the opened category.
`nil`	The category does not exist or cannot be opened in the specified mode.

Note: By calling `ddCatOpenEx( g_ddId t_categoryName t_mode nil)`, you get the same behavior as in ddCatOpen.

© 2020 All Rights Reserved.

Example

```
cat = ddCatOpenEx(libId "myCat" "a" "1")
ddCatAddItem(cat "cell1" "cell")
ddCatSave(cat)
ddCatClose(cat)
```

Updates (or creates) a category and adds a cell name to it.

© 2020 All Rights Reserved.

ddCatAddItem

```
ddCatAddItem(
     g_ddId
     t_name
     t_type
     )
     => t / nil
```

Description

Adds the specified name to the given category.

Edits are not written to disk until a `ddCatSave` is done. The given category must have been opened for "`w`" (write) or "`a`" (append).

Arguments

g_ddId	The object ID of the item.
t_name	The name of the item.
t_type	The object type of the item. The caller must specify the type of the object being added.

Value Returned

`t`	The function was successful
`nil`	The function failed.

© 2020 All Rights Reserved.

ddCatSubItem

```
ddCatSubItem(
     g_ddId
     t_name
     )
     => t / nil
```

Description

Removes the named item from the given category.

If the name is the name of a category (that is, removing a subcategory from another category), the checks if it is referenced in any other category. If not, the category is moved to the library category list. This action ensures that the caller must do a `ddCatRemove` to remove category entries from a library. Changes are not written to disk until a `ddCatSave` is done.

Arguments

g_ddId	The object ID of the item.
t_name	The name of a cell or category.

Value Returned

`t`	The function was successful
`nil`	The function failed

© 2020 All Rights Reserved.

ddCatSave

```
ddCatSave(
     g_ddId
     )
     => t / nil
```

Description

Writes the category information to disk. This writes out any changes made to the category to disk.

Arguments

g_ddId	The object ID of the item.

Value Returned

`t`	The function was successful.
`nil`	The function failed.

© 2020 All Rights Reserved.

ddCatRemove

```
ddCatRemove(
     g_ddId
     )
     => t / nil
```

Description

Removes the category and its contents from disk. All references to the category are removed from the library.

Arguments

g_ddId	The object ID of the category.

Value Returned

`t`	The function was successful.
`nil`	The function failed.

© 2020 All Rights Reserved.

ddCatClose

```
ddCatClose(
     g_ddId
     )
     => t / nil
```

Description

Closes the category and purges its information from virtual memory. Any edits since the last `ddCatSave` are lost.

Arguments

g_ddId	The object ID of the category.

Value Returned

`t`	The function was successful.
`nil`	The function failed.

© 2020 All Rights Reserved.

ddCatGetCatName

```
ddCatGetCatName(
     g_ddId
     )
     => t_categoryName / nil
```

Description

Returns the category name of the given category ID.

Arguments

g_ddId	The object ID of the category.

Value Returned

t_categoryName	The category name of the given category ID.
nil	The function failed

© 2020 All Rights Reserved.

ddCatGetCatMembers

```
ddCatGetCatMembers(
     g_ddId
     )
     => l_memberList / nil
```

Description

Returns the list of members of the given category. The function does not return the members (cells) that do not exist in the library.

Arguments

g_ddId	The object ID of the category.

Value Returned

l_memberList	The list of member name and type pairs in the given category.
nil	The category has no members.

Example

```
'(("nand2" "cell") ("Models" "category"))
```

Returns the list of member names and type pairs.

© 2020 All Rights Reserved.

ddCatGetLibCats

```
ddCatGetLibCats(
     g_ddId
     )
     => l_categoryNames / nil
```

Description

Returns the list of library-level categories.

Arguments

g_ddId	The object ID of the library.

Value Returned

l_categoryNames	The list of library-level categories.
nil	The library has no categories.

© 2020 All Rights Reserved.

ddCatFindCat

```
ddCatFindCat(
     g_ddId
     t_name
     t_mode
     )
     => b_objId / nil
```

Description

Returns the ID of a category with the given name in the given mode. Traverses category hierarchies.

This function calls `ddCatOpen` to get the category ID to be returned. The caller must call `ddCatClose` to properly flush the information from virtual memory when done. The category is opened in the given mode.

Arguments

g_ddId	The library ID.
t_name	The category name.
t_mode	The modes "`a`", "`w`", or "`r`". See ddGetObj.

Value Returned

b_objId	The ID of a category with the given name in the given mode. The returned ID can be used to access the category members.
`nil`	The function failed.

© 2020 All Rights Reserved.

ddCatGetCellsNotInCat

```
ddCatGetCellsNotInCat(
     g_ddId
     )
     => l_cellNames / nil
```

Description

Returns the names of cells that are not in any category. This information is derived from an existing library.

Arguments

g_ddId	The library ID.

Value Returned

l_cellNames	List of cells that are not in any category in the specified library.
nil	All cells are listed in some category.

© 2020 All Rights Reserved.

ddCatIsObjInCat

```
ddCatIsObjInCat(
     t_name
     b_ddId
     )
     => t_type / ddcCatUnknownType
```

Description

Searches for the named object in the given category.

Arguments

t_name	The object name.
b_ddId	The category ID.

Value Returned

t_type	The type of the object found if the category contains something of that name.
ddcCatUnknownType	The category does not contain anything of that name.

© 2020 All Rights Reserved.

Old Cell Category Functions

These functions are provided for backward compatibility. If you are writing new code, use the ddCat functions from the previous section.

dmAddCellToCategory

```
dmAddCellToCategory(
     g_categoryId
     l_cellList
     )
     => t / nil
```

Description

Adds the specified cells to the category.

Supplied for compatibility. The equivalent functionality using the `ddCat` API would be

```
foreach( i cellList
    ddCatAddItem( categoryId i~>name "cell )
)
ddCatSave( categoryId )
```

Using the `ddCat` API is more efficient because the categories are implemented using check-point files that must be locked against access by other users. Although the edits are written to disk (`ddCatSave`), the process still has a lock on the category until it is freed using `ddCatClose`.

After you call `dmAddCellToCategory`, use `ddCatClose` on *`g_categoryId`*.

Note: The DFII-DM function accepted a list of cell IDs. For forward compatibility, the implementation of this function accepts a list of cell IDs or cell names.

Arguments

`g_categoryId`	ID of the category to which the specified cells are added.
`l_cellList`	List of cells to be added to the category.

Value Returned

`t`	The cells are added.

© 2020 All Rights Reserved.

`nil`	The specified cells or the category ID cannot be found.

© 2020 All Rights Reserved.

dmCreateCellCategory

```
dmCreateCellCategory(
     g_libOrCatId
     t_categoryName
     l_cellList
     )
     => t / nil
```

Description

Creates a cell category and adds all specified cells to the category.

Supplied for compatibility. The equivalent functionality using the `ddCat` API would be

```
catId = ddCatOpen( libOrCatId categoryName "w" )
when( catId
    foreach( i cellList
        ddCatAddItem( catId i~>name "cell" )
    )
    ddCatSave( catId )
    ddCatClose( catId )
)
```

Using the `ddCat` API is more efficient because the categories are implemented using checkpoint files that must be locked against access by other users. The above lines of code explicitly gets a lock (the `ddCatOpen`) and frees the lock (the `ddCatClose`).

Note: The DFII-DM function accepted a list of cell IDs. For forward compatibility, the implementation of this function accepts a list of cell IDs or cell names.

Arguments

g_libOrCatId	ID of the library or cell category to be created.
t_categoryName	Name of the category to be created.
l_cellList	List of cells to be added to the category.

Value Returned

`t`	The category is created.
`nil`	The category cannot be created.

© 2020 All Rights Reserved.

dmRemoveCellCategory

```
dmRemoveCellCategory(
     g_categoryId
     )
     => t / nil
```

Description

Removes a cell category from a library or from a parent cell category.

Supplied for compatibility. The equivalent functionality using the `ddCat` API would be

```
ddCatRemove( categoryId )
```

Arguments

g_categoryId	ID of the cell category to be removed.

Value Returned

`t`	The cell category is removed.
`nil`	The library or parent cell category cannot be found.

© 2020 All Rights Reserved.

dmRemoveCellFromCategory

```
dmRemoveCellFromCategory(
     g_categoryId
     l_cellList
     )
     => t / nil
```

Description

Removes one or more cells from a cell category.

Supplied for compatibility. The equivalent functionality using the `ddCat` API would be

```
foreach( i cellList
    ddCatSubItem( categoryId i~>name )
)
ddCatSave( categoryId )
```

After you call `dmRemoveCellFromCategory`, use `ddCatClose` on *`g_categoryId`*.

Note: The DFII-DM function accepted a list of cell IDs. For forward compatibility, the implementation of this function accepts a list of cell IDs or cell names.

Arguments

`g_categoryId`	ID of the cell category to be removed.
`l_cellList`	List of cells to be added to the category.

Value Returned

`t`	The cell category is removed.
`nil`	The library or parent cell category cannot be removed.

© 2020 All Rights Reserved.

Design Editor SKILL Functions

This section describes the SKILL level interface to the Design Editor, including

- Registering and un-registering applications
- Registering DE plugins
- Client interface
- Application trigger routines
- General support routines
- Support routines for applications

Registering and Un-registering Applications

The Design Editor requires applications to register their characteristics and entry points so that they can be called in the appropriate situations. If the user attempts to edit a design for which the view type is not registered, no window will be opened.

Registration is not the same as licensing. The application which is registered with the Design Editor still might not be licensed for use.

Both application view types and view subtypes are registered using the same procedure, defined in this section. However, because the data and application triggers can only be registered at the root (or view type) level an error is generated if these procedures are supplied for a view subtype.

© 2020 All Rights Reserved.

deRegApp

```
deRegApp(
     [ ?viewType t_viewType ]
     [ ?parentType t_parentType ]
     [ ?appName t_appName ]
     [ ?help t_help ]
     [ ?dataTrigger s_dataTrigger ]
     [ ?appTrigger s_appTrigger ]
     [ ?enableTrigger s_enableTrigger ]
     [ ?menuTrigger s_menuTrigger ]
     [ ?postInstallTrigger s_postInstallTrigger ]
     [ ?viewNameSwitchTrigger s_viewNameSwitchTrigger ]
     [ ?widgetType t_widgetType ]
     [ ?noCadenceMenu g_noCadenceMenu ]
     [ ?windowBBoxProc s_function ]
     [ ?appTitle t_appTitle ]
     [ ?exclusive g_exclusive ]
     )
     => t / nil
```

Description

Registers the characteristics and entry points of applications so that they can be called in the appropriate situations.

Arguments

`?viewType` *`t_viewType`*

View type (or view subtype, if *`t_parentType`* is not `nil`) to which the application wants to attach itself.

An application can attach itself to any number of view types, but must call `deRegApp` once for each one. The list of predefined view types currently available are listed in “Predefined Views and ViewTypes.”

All view type and view subtype names must be unique. That is, if an application subclasses two different view types, it must use two unique subTypes even if the triggers are otherwise identical.

`?parentType` *`t_parentType`*

View type or view subtype that *`t_viewType`* subclasses.

© 2020 All Rights Reserved.

An application can subclass any number of parentTypes, but must call `deRegApp` with a unique subType name for each one.

`t_parentType` does not have to be previously registered. However, an application can never be accessed until *`t_parentType`* is successfully registered.

If *`t_parentType`* is *""*, the data and application triggers must be specified, or an error results.

If *`t_parentType`* is not empty, these triggers must be `nil`, or an error results.

`?appName` *`t_appName`*

Name of the application.

This name is used for a number of purposes, including attaching a set of bindkeys to the window when the application is installed, and for clients of `deOpen` to use when they want to explicitly override the default application associated with a given view type.

When the application is registered, *`t_appName`* is registered as a bindkey prefix with the Human Interface code. Subsequently, hiSetBindKey can be called with *`t_appName`* to associate a key binding with the application.

The Human Interface code provides a mechanism for sharing bindkeys between multiple applications.

`?help` *`t_help`*

Help string for the application. Whenever the application is installed in a window, the window's help string is set to this value.

`?dataTrigger` *`s_dataTrigger`*

SKILL function that will be called when the user has finished specifying to DE the particular design data to be edited. The data trigger has the responsibility of opening the data. The data trigger is discussed in detail in the next section.

A data trigger is always associated with a view type, such as `maskLayout`. If a data trigger for an application is not executed successfully, the following warning might display:

```
WARNING (DEBASE-102084): Data trigger for
viewType %s failed.
```

`?appTrigger` *`s_appTrigger`*

© 2020 All Rights Reserved.

SKILL function that is called after the data is installed in the window. The primary function of this routine is for the application to manage any runtime context required by the application. The application trigger is discussed in detail in the next section.

`?enableTrigger` *`s_enableTrigger`*

SKILL function that is called to determine if the application is enabled for use by the current user.

The enable trigger will normally do any appropriate security checks necessary to make this determination. This function should be defined in an autoload file, and not in the application context. Otherwise, the application context will be loaded even when the security checks fail.

This trigger is optional, and if omitted, no license check is made by the Design Editor. However, if licensing or other restriction is desired, then the application code is responsible for implementing it.

`?menuTrigger` *`s_menuTrigger`*

SKILL function that is called when the Design Editor determines that menus for the application must be installed in the banner.

This routine is normally defined in the application context and returns a list of menus to be installed when this application is required. If this trigger is omitted, the application itself must take responsibility for any banner menu initialization.

`?postInstallTrigger` *`s_postInstallTrigger`*

SKILL function that is called when the DE has finished all installation, including the banner menus, and displaying the window.

This is provided for those applications that need to finish up any initialization that must occur after menus are installed.

`?viewNameSwitchTrigger` *`s_viewNameSwitchTrigger`*

DE calls this SKILL function when switching from a subview type (from_viewtype) to a different root type (to_roottype).

If a valid subview name is returned, DE uses it, and installs the application of the view type. If `nil` is returned, DE installs the new root type application.

© 2020 All Rights Reserved.

For example, DE calls this SKILL function when descending in a window or switching from `schematicXL` to a `text.v`.

However, switching from `schematicXL` to `schematic` does not call this trigger because there is no change in the root type of the application.

`?widgetType` *`t_widgetType`*

Window widget type the application needs.

Whenever the application is installed in a window, the window's widget type is set to this value. This value can only be set for root class view types.

The value must be either `text`, corresponding to plain text data, or `graphics`, corresponding to general graphically displayed data. This value defaults to `graphics` if not supplied.

`?noCadenceMenu` *`g_noCadenceMenu`*

Flag indicating whether the standard Cadence menu is desired as the first item in the menu banner.

If `nil`, the Cadence menu will be supplied automatically. If if is `t`, the Cadence menu will not appear.

The default is `nil`, so only applications which explicitly do not desire this menu need to supply it.

Design Editor clients not using this menu are advised that this menu contains all the basic Design Editor commands, and that those which apply to the application (particularly `deReturn`) should be made available to the user in some manner.

`?windowBBoxProc` *`s_function`*

Function to register. The function is passed the *`t_appName`* so it must accept a string and return a hi window bbox specification where the first coordinate is the lower left corner and the second coordinate is the upper right corner.

hi bbox's are based on a screen where (0,0) is the lower left hand corner of the screen.

`?appTitle` *`t_appTitle`*

© 2020 All Rights Reserved.

Name that will appear on the customize menu for this application.

`?exclusive` *`g_exclusive`*

An optional argument, which causes the view to be opened in a new window if its view type is flagged exclusive and there already exists a view type (ancestor or sub view type that is also exclusive) in the current session window.

Value Returned

`t`	The registration was successful.
`nil`	the registration was not successful.

Example

See Extended Design Editor Example.

© 2020 All Rights Reserved.

deUnRegApp

```
deUnRegApp(
     t_viewType
     )
     => t / nil
```

Description

Un-registers the view type.

Only one application can currently be registered for each view type. If a second registration is attempted, an error results. To re-register a view type with different procedures, the view type must first be unregistered with this routine.

Arguments

t_viewType	View type to be removed from the list of applications known to the Design Editor.

Value Returned

`t`	The view type was unregistered.
`nil`	The view type was not unregistered.

Example

```
deUnRegApp( "graphic" )
```

Removes "`graphic`" from the list of registered view types and returns `t`.

© 2020 All Rights Reserved.

deRegUserTriggers

```
deRegUserTriggers(
     t_viewType
     [ s_appTrigger [ s_menuTrigger [ s_postInstallTrigger ] ] ]
     )
     => t / nil
```

Description

Registers user application, user menu, and user post install triggers.

A user application trigger is called immediately following the application trigger. (Only root view types can have application triggers, therefore, only root view types can have user application triggers.) A user menu trigger is called before the menu trigger. A user post install trigger is called immediately following the post install trigger. The following warning might display if an issue occurs while executing a user application trigger:

```
WARNING (DEBASE-102079): A SKILL error occurred in function
<functionName>.
```

User triggers are optional and can be defaulted to `nil`; however, there is no point in registering user triggers if all are `nil`. This condition, therefore, is flagged as an error. User triggers are described in detail in the next section.

Multiple user triggers can be registered for any given view type. If a user trigger is already registered when `deRegUserTriggers` is called, a warning message displays. To remove a user trigger associated with a view type, see deUnRegUserTriggers.

Arguments

t_viewType	View type or view subtype to which *s_appTrigger*, *s_menuTrigger*, and *s_postInstallTrigger* will be attached. An application can attach user triggers to any number of view types, but must call `deRegUserTriggers` once for each one.
s_appTrigger	Symbol for a SKILL function to be called after the application trigger is called. The primary function of this routine is to allow the user to manage any runtime context that might need to be added to customize an existing application. For more information about the user application trigger, see userAppTrigger.

© 2020 All Rights Reserved.

s_menuTrigger Symbol for a SKILL function to be called before the menu trigger is called. The primary function of this routine is to allow the user to supply any menus that might need to be added to customize an existing application. For more information about the user menu trigger, see userMenuTrigger.

s_postInstallTrigger

Symbol for a SKILL function to be called after the post-install trigger is called. The primary function of this routine is to allow the user to do any processing that might be required after the application has been fully installed and displayed in the window. For more information about the user post install trigger, see userPostInstallTrigger.

Value Returned

`t`	Registration was successful.
`nil`	Registration failed.

Example

```
deRegUserTriggers("schematic" 'myAppTrigger 'myMenuTrigger 'myPostTrigger)

deRegUserTriggers("schematic" nil 'userMenuTrigger)

deRegUserTriggers("schematic" nil nil 'myPostTrigger)
```

(ICADVM20.1 Only) The following example registers trigger for Layout EXL.

```
deRegUserTriggers("maskLayoutEXL" nil nil 'trigger)
```

From ICADVM20.1 release, Layout Suite supports only the Layout XL and Layout EXL tiers. For backward compatibility, `deRegUserTriggers` continues to support `"maskLayoutGXL"`, and `"maskLayoutEAD"` as the values for *`t_viewType`* argument. However, in this software version, `"maskLayoutGXL"` is mapped to `"maskLayoutXL"`, and `"maskLayoutEAD"` is mapped to `"maskLayoutEXL"`.

© 2020 All Rights Reserved.

deUnRegUserTriggers

```
deUnRegUserTriggers(
     t_viewType
     )
     => t / nil
```

Description

Removes any user triggers associated with *t_viewType*.

Arguments

t_viewType	View type or view subtype for which the user triggers are to be removed.

Value Returned

t	The triggers were unregistered.
nil	The triggers were not unregistered.

Example

```
deUnRegUserTriggers( "graphic" )
```

Removes any user triggers associated with view type “graphic” and returns t.

© 2020 All Rights Reserved.

deInheritEnabling

```
deInheritEnabling(
     t_viewType
     )
     => t / nil
```

Description

Enables *t_viewType* to depend on the enabling of its parent type. Thus, the user must have the parent application licensed in order to use the application. *t_viewType* must be a subclass of some other view type.

Arguments

t_viewType	View subtype for which enabling is to be inherited.

Value Returned

`t`	The enabling was inherited.
`nil`	The enabling was not inherited.

Example

```
deInheritEnabling( "graphic" )
```

Sets view type “`graphic`” to inherit the enabling of its parent and returns `t`.

© 2020 All Rights Reserved.

deUnInheritEnabling

```
deUnInheritEnabling(
     t_viewType
     )
     => t / nil
```

Description

Removes the inheritance of enabling for *t_viewType*.

Arguments

t_viewType	View subtype for which the inheritance of enabling is to be removed.

Value Returned

`t`	The enabling inheritance was removed.
`nil`	The enabling inheritance was not removed.

Example

```
deUnInheritEnabling( "graphic" )
```

Removes dependence of view type “`graphic`” enabling on its parent and returns `t`.

© 2020 All Rights Reserved.

deInheritMenus

```
deInheritMenus(
     t_viewType
     )
     => t / nil
```

Description

Causes *t_viewType* to inherit the menus defined by its parent class.

When this application is installed, the *menu trigger* of its parent (which can itself inherit menus) is called, after which the *menu trigger* associated with *t_viewType* is called and the results of the latter call are appended to that of the former to construct the banner menu list. If there is no parent type, or the inheritance cannot be done, `nil` is returned.

Note: Inheriting menus implies that both application menus and user application menus are inherited.

Arguments

t_viewType	View subtype for which the inheritance of menus is desired.

Value Returned

`t`	The menus could be inherited.
`nil`	The menus could not be inherited.

Example

```
deInheritMenus( "graphic" )
```

Sets view type “`graphic`” to inherit the menus of its parent and returns `t`.

© 2020 All Rights Reserved.

deUnInheritMenus

```
deUnInheritMenus(
     t_viewType
     )
     => t / nil
```

Description

Removes the inheritance of menus for *t_viewType*.

Arguments

t_viewType	View subtype for which the inheritance of menus is to be removed.

Value Returned

t	The menu inheritance was removed.
nil	the menu inheritance was not removed.

Example

```
deUnInheritMenus( "graphic" )
```

Removes dependence of view type “graphic” menus on its parent and returns t.

© 2020 All Rights Reserved.

deRegProbeTrigger

```
deRegProbeTrigger(
     s_probeTrigger
     )
     => t / nil
```

Description

Registers a probe trigger, *s_probeTrigger*, to be called after creating a new window or installing a new design in an existing window.

The probe trigger is responsible for maintaining the probe context, which is kept for each window. The probe trigger is also called when the window is closed to delete the probe context. Only one probe trigger can be registered. If one has already been registered, an error is generated. The probe trigger is discussed in the next section.

Arguments

s_probeTrigger	SKILL function to be called when a window probe context needs to be updated because of a change in the window contents.

Value Returned

`t`	The probe trigger was installed.
`nil`	The probe trigger was not installed.

Example

```
deRegProbeTrigger( 'myProbeTrigger )
```

Installs `'myProbeTrigger` as the current global probing trigger and returns `t`.

© 2020 All Rights Reserved.

deUnRegProbeTrigger

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

Description

Removes the current probe trigger.

Arguments

None

Value Returned

`t`	The probe trigger was removed.
`nil`	The probe trigger was not removed.

Example

```
deUnRegProbeTrigger()
```

Removes the current probe trigger and returns `t`.

© 2020 All Rights Reserved.

deGetProbeTrigger

```
deGetProbeTrigger(
     )
     => s_probeTrigger / nil
```

Description

Returns the current probe trigger.

Arguments

None

Value Returned

s_probeTrigger	SKILL symbol for the current global probe trigger.
nil	No current probe trigger is registered.

Example

```
deGetProbeTrigger()
```

Assuming the previous example, returns `'myProbeTrigger`.

© 2020 All Rights Reserved.

deGetAppInfo

```
deGetAppInfo(
     t_viewType
     )
     => l_appInfo / nil
```

Description

Returns the application registration information associated with `t_viewType` and its sub-view types.

This function returns a property list with the following elements:

l_appInfo->viewType	viewType	(string)
l_appInfo->parentType	Parent type	(string)
l_appInfo->appName	Application name	(string)
l_appInfo->help	Application help string	(string)
l_appInfo->dataTrigger	SKILL data trigger	(symbol)
l_appInfo->appTrigger	SKILL application trigger	(symbol)
l_appInfo->userAppTrigList	SKILL user app trigger list	(symbol)
l_appInfo->userMenuTrigList	SKILL user menu trigger list	(symbol)
l_appInfo->userPostInstallTrigList	SKILL user post-install trigger list	(symbol)
l_appInfo->inheritEnabling	Whether or not to inherit enabling	(t/nil)
l_appInfo->inheritMenus	Whether or not to inherit menus	(t/nil)
l_appInfo->widgetType	*widgetType* of the window	(string)
l_appInfo->noCadenceMenu	Whether or not to install the Cadence Menu	(t/nil)
l_appInfo->subTypes	*subviewType* list of this *viewType*	(list)

If `t_viewType` is a root view type, then *l_appInfo->parentType* will be *""*, and `l_appInfo`*->dataTrigger* and *l_appInfo->appTrigger* will be non-`nil`. Otherwise, the opposite is true.

l_appInfo->subTypes is a list of property lists, one for each defined *subviewType* of `t_viewType`, which have the same format. Thus `l_appInfo` contains the entire class tree for `t_viewType`.

© 2020 All Rights Reserved.

- If there are no sub viewtypes, then *l_appInfo->subTypes* is `nil`.
- If `t_viewType` is itself a subType, then only the *subType* tree with `t_viewType` at its root is returned.

Note: `l_appInfo->userAppTrigger`, `l_appInfo->userMenuTrigger`, and `l_appInfo->userPostInstallTrigger` are supported for compatibility reasons only and should not be used otherwise. If multiple functions are registered for any of these triggers, only the first function of the list is returned.

Arguments

`t_viewType`	View type for which you want the information.

Value Returned

`l_appInfo`	Application information for `t_viewType`, if any.

Example

For the subviewType, maskLayout

```
deGetAppInfo("maskLayout") => (nil userMenuTrigList (m1 m2)
userPostInstallTrigList nil userPostInstallTrigger nil userAppTrigList nil
viewType "maskLayout" parentType ""
appName "Layout" apptitle nil help
"Layout" widgetType "graphics" noCadenceMenu t
dataTrigger geDataTrigger appTrigger leAppTrigger enableTrigger
leEnableTrigger menuTrigger leMenuTrigger exitTrigger nil
postInstallTrigger nil windowBBoxProc leWindowBBoxProc userAppTrigger nil
userMenuTrigger m1 inheritMenus nil
inheritEnabling nil subTypes (
    (nil viewType "maskLayoutXL" "parentType" "maskLayout"
    appName "Virtuoso XL" appTitle "Layout XL" help
    "layoutXLhelp" widgetType "graphics" noCadenceMenu t
    dataTrigger nil appTrigger nil enableTrigger
    lxiEnableTrigger menuTrigger lxiMenuTrigger exitTrigger
    lxiExitTrigger
    postInstallTrigger lxiPostInstallTrigger windowBBoxProc nil
    userAppTrigger nil userMenuTrigger nil inheritMenus nil
    inheritEnabling nil subTypes nil
    )
```


© 2020 All Rights Reserved.

deGetAppInfo and changing the application information

Caution

`deGetAppInfo()` *returns a pointer to the application information. If the application information is then changed, your local pointer to the application information also changes.*

Example

Register the user triggers.

```
deRegUserTrigger("maskLayout"
    'appTriggerCB 'menuTriggerCB 'postInstallTriggerCB
 )
```

Verify that the triggers are registered.

```
localAppInfo = deGetAppInfo("maskLayout")
localAppInfo->userAppTrigger ; Display the userAppTrigger.
     appTriggerCB
localAppInfo->userMenuTrigger ; Display the userMenuTrigger.
     menuTriggerCB
localAppInfo->postInstallTriggerCB  ; Display the postInstallTrigger
postInstallTriggerCB
```

Un-register the user triggers.

```
deUnRegUserTriggers("maskLayout")
```

Note: The `localAppInfo` properties are set to `nil` even though `localAppInfo` was not directly modified.

```
localAppInfo->userAppTrigger
     nil
localAppInfo->userMenuTrigger
     nil
localAppInfo->postInstallTriggerCB
     nil
```

Copy the application information into different variables if you want to preserve the information for future reference:

```
localAppTrigger = localAppInfo->userAppTrigger
```


© 2020 All Rights Reserved.

deHiSetDefaultApplication

```
deHiSetDefaultApplication(
     )
```

Description

Opens the *Set Default Application* form for the current cellview.

You can use the *Set Default Application* form to automatically launch an application when a cellview is opened rather than choosing the application from *Tools* menu each time. For more information, see *Set Default Application Form* in Appendix A of *Virtuoso Layout Suite L User Guide*.

Arguments

None

Value Returned

None

Example

```
deHiSetDefaultApplication()
```


© 2020 All Rights Reserved.

deSetWorkspace

```
deSetWorkspace(
     w_window
     t_workspaceName
     )
     => t / nil
```

Description

Sets a workspace name to a given window (as long as the workspace name is valid for that window).

Arguments

w_window	The given window.
t_workspaceName	The workspace name to be set for the given window (must be valid).

Value Returned

`t`	Workspace name set to window successfully.
`nil`	Workspace name not set and warning printed.

Example

```
deSetWorkspace(window(3) "Classic")
```

Sets `window(3)` to the workspace name "`Classic`".

© 2020 All Rights Reserved.

Registering DE Plugins

A design editor (DE) plugin is an add-on to a DE main application, such as VSE and VXL. The DE plugin adds extra capability to a main application, or another plugin. It can customize banner menus, toolbars,and bindkeys of other applications and/or own menus, toolbar assistants, and bindkeys. The main advantage of a plugin over a subapplication is that a plugin does not replace the running application when installed. In addition, you can simultaneously access multiple plugins.

You create your own DE plugin by defining a subclass of the DE base class, `DEPlugin` and re-implementing certain public methods of the class that can be *overridden*, if required.

Note: The public methods that can be *overridden* are named with the prefix `deo`, where the character `o` indicates *override*. You should not reimplement methods that are not prefixed with `deo`.

The following example defines a plugin subclass with the name `My Plugin` of the `DEPlugin` base class.

```
defclass( MyPlugin (DEPlugin) ())
```

There are three main types of public methods that can be *overridden* in the DE Plugin subclass. These include:

- Trigger Methods: These include deoEnableTrigger, deoMenuTrigger, deoPostInstallTrigger, and deoExitTrigger. The following example redefines the trigger methods for the plugin subclass `MyPlugin`, defined above.

```
(let()
    defmethod( deoEnableTrigger( (this MyPlugin) @optional args)
        mylicenseCheck( args))
    defmethod( deoMenuTrigger( (this MyPlugin) args)
        (if  (eq args->action 'activate)
        append( args->bannerMenus (myMenuMaker args))
        ;;else
        (when (eq args->action 'deactivate)
            (remove_my_plugin_menu args->bannerMens))))
    defmethod( deoPostInstallTrigger((this MyPlugin) args)
        (my_post_installer args)
             t )
     defmethod( deoExitTrigger((this MyPlugin) @optional args)
    (my_pre_exit_process args)
     t )
```


© 2020 All Rights Reserved.

```
    ) ;; End of MyPlugin
```

- Event Handling Methods: These include deoOnApplicationChange and deoOnTabSwitch

The following example redefines the event handling methods that can be overridden in the `MyPlugin` subclass.

```
(defmethod deoOnApplicationChange ((this MyPlugin) args)
(handle_application_change args))


(defmethod deoOnTabSwitch ((this MyPlugin) session_window into_window
outof_window)
    (handle_tab_switch session_window into_window outof_window))
```

- GUI Customization Methods: These include methods for defining and customizing new toolbars, defining and customizing bindkeys, customizing the banner menu of an application window, and working with workspaces.

To define a new toolbar for your plugin, provide a toolbar definition file for the plugin in the same way as you would provide for a DE main application. For example, to define toolbars for a plugin name `My Plugin`, the toolbar file should be installed in a CSF searchable location as follows:

```
dfII/toolbars/byApplication/My_Plugin.toolbars
```

"`My Plugin`" is the same string given as argument `appName` to `deRegPlugin`. However, the *space* in the string is replaced with an *underscore* (_).

To customize toolbars, you need to define the following methods in your DE Plugin subclass: deoRequireToolbars and deoUpdateToolbar.

The following example customizes a toolbar `View` for `My Plugin`.

```
defmethod( deoRequireToolbars ((this MyPlugin))
    cons( "View" (callNextMethod this)))


  (defmethod deoUpdateToolbar ((this MyPlugin) (name string) (toolbar hiToolBar)
args)
    (if (equal name "View")
        (if (eq args->action 'activate)
            (customize_toolbar toolbar args)
          (revert_customization toolbar args))

      (callNextMethod this name toolbar args)))
```


© 2020 All Rights Reserved.

To customize bindkey, you need to define the following methods in your DE Plugin subclass: deoGetBindkeys.

An application can re-define `deoGetBindkeys` to provide more dynamic key-bindings. For example:

```
defmethod( deoGetBindkeys((this MyPlugin))
append( (callNextMethod this) my_bindkeys))
```

To customize the banner menu of an application window, DE calls the deoMenuTrigger method defined by the plugin.

A plugin does not have its own workspace. However, a main application workspace can contain assistants and toolbars defined by a plugin. When the saved workspace is re-applied later, all plugins contained in the workspace are reinstalled.

The methods to create, register, install, and customize a plugin are described in detail in the following sections.

© 2020 All Rights Reserved.

deRegPlugin

```
deRegPlugin(
     t_appName
     s_className
     s_superClass
     [ ?appTitle t_appTitle ]
     [ ?bindkeys l_bindkeys ]
     [ ?group t_group ]
     [ ?help t_help ]
     [ ?pluggableMainApps g_pluggableMainApps ]
     [ ?pluggableViewTypes l_pluggableViewTypes ]
     [ ?shortDescription t_shortDescription ]
     [ ?excludeViewTypes l_excludeViewTypes ]
     )
     => l_appClassInfo
```

Description

Registers the characteristics of plugins.

This function issues a warning in the following situations:

- a class symbol is already used by an application
- a plugin application name is already registered, or used as an application tool name, for example, "&ADE L"

Arguments

t_appName	Specifies a name for the plugin. This name is used for a number of purposes, including attaching a set of bindkeys to the window when the plugin is installed.
s_className	Specifies the name of the SKILL++ class. If not already loaded, the class and superclass (from which this plugin class is implemented) are auto-loaded when deInstallPlugin is applied to *t_appName* or *s_className*. For information about autoloading your classes, see "Autoloading Your Classes".
s_superClass	Specifies the name of the superclass from which the *s_className* is derived

© 2020 All Rights Reserved.

`?appTitle` *`t_appTitle`*

Specifies the name that is to appear on the slider menu for this plugin. If no *`t_appTitle`* is specified, *`t_appName`* is used on the slider menu. You can also use the mnemonic ampersand (&) symbol in the title.

`?bindkeys` *`l_bindkeys`*

Specifies a list of bindkeys for the plugin that will be passed to `hiSetBindKeys`. This list does not need to include any bindkeys that are defined by the superclass(es) of the plugin. This is because when the plugin is installed, the design editor calls `hiSetBindKeys` with collective bindkeys (defined by this plugin and by the superclass(es) of this plugin.)

`?group` *`t_group`*

Specifies a name to group plugins together on the slider menu. If specified, the grouped plugins are displayed in a submenu with the group name (*`t_group`*).

`?help` *`t_help`*

Specifies a help string for the plugin. However, this value is not currently used.

`?pluggableMainApps` *`g_pluggableMainApps`*

Specifies a list of main application names, such as `list("Schematics" "Schematics XL")`, or a single string "`ALL`", on which the plugin can be installed using `deInstallPlugin`

`?pluggableViewTypes` *`l_pluggableViewTypes`*

Specifies a list of view (sub) types (string) on which the plugin can be installed using `deInstallPlugin`. If main application names are not specified, but a list of view (sub) types is given, the plugin can be installed to all applications associated with the view (sub) types.

`?shortDescription` *`t_shortDescription`*

Specifies a brief description of the plugin. If specified, the description is displayed in the prompt line at the bottom of the window when the plugin name is highlighted in the *Window – Plugins* menu.

`?excludeViewTypes` *`l_excludeViewTypes`*

Specifies a list of view types on which the plugin cannot be installed

© 2020 All Rights Reserved.

Value Returned

l_appClassInfo	Returns a copy of the registration information associated with the plugin, if the registration is successful
`nil`	Returns `nil` if the registration is not successful

Examples

```
deRegPlugin("My Plugin" 'MyPlugin 'DEPlugin ?pluggableMainApps '("Schematics"
"Schematics XL"))
```

This example registers a plugin with the application name `MyPlugin`, which can be installed on VSE, and VSE-XL application windows.

```
deRegPlugin("My Plugin" 'MyPlugin 'DEPlugin ?pluggableViewTypes '("schematic"))
```

This example registers `MyPlugin` so that it can be installed on all applications associated with view type `schematic`, such as VSE and VSE XL applications.

```
deRegPlugin("My Plugin" 'MyPlugin 'DEPlugin ?appTitle "My &Plugin"
?pluggableViewTypes '("schematic"))
```

This example registers `MyPlugin` with an `appTitle`.

```
deRegPlugin( "MyPlugin" ...
    ?pluggableMainApps '("Text Editor XL")
    ?pluggableViewTypes '("schematic")?excludeViewTypes '("text.veriloga"))
```


© 2020 All Rights Reserved.

deInstallPlugin

```
deInstallPlugin(
     w_windowID
     S_plugin
     )
     => stdobj / nil
```

Description

Installs (or activates) the specified plugin in an application window associated with the given ID. For each session window, DE makes only one instance of a plugin. `deInstallPlugin` function installs a plugin in an application window. A session window can host multiple application windows. This means that if an instance of the plugin already exists in the specified window, the instance is returned. And, if there exists an instance in another tab of the same session window, the instance is shared.

Arguments

w_windowID	Specifies the ID of an application window on which the plugin is to be installed
S_plugin	Specifies the name (string) or class name (symbol) of the plugin to be installed

Value Returned

stdobj	Returns an instance of the plugin installed in the given application window
`nil`	Returns `nil` if the named plugin (string) has not been registered with `deRegPlugin`, the class name (symbol) is not a sub-class of `DEPlugin`, or the plugin is not pluggable to the given application window (pluggable applications are specified when registering the plugin using `deRegPlugin`)

© 2020 All Rights Reserved.

deRemovePlugin

```
deRemovePlugin(
     w_window
     S_plugin
     )
     => t / nil
```

Description

Removes (or deactivates) the named plugin from the specified application window–session or tab window. If the ID is of a session window, the plugin is removed from all tab windows associated with the session window.

See also deoExitTrigger

Arguments

w_window	Specifies the ID of the window from which the plugin is to be removed.
S_plugin	Specifies the name (string) or class name (symbol) of the plugin to be removed.

Value Returned

`t`	Returns `t` if the plugin is successfully removed.
`nil`	Returns `nil` if the named plugin is not found installed in the specified window or its exitTrigger returns `nil`.

© 2020 All Rights Reserved.

deFindPlugin

```
deFindPlugin(
     S_plugin
     g_window
     )
     => stdobj / nil
```

Description

Searches for and returns an instance of the specified plugin installed in the given application window

Arguments

S_plugin	Specifies the name (string) or class name (symbol) of the plugin you want to search for.
g_window	Specifies the ID of the application window–session or tab–in which you want to search for the named plugin.

Value Returned

stdobj	Returns an instance of the named plugin you are searching for
`nil`	Returns `nil` if the named plugin is not found installed in the specified window

© 2020 All Rights Reserved.

deBeginConfigurePlugins

```
deBeginConfigurePlugins(
     w_window
     )
     => nil
```

Description

Lets applications issue a series of `deInstallPlugin` and `deRemovePlugin` calls to add or remove plugins in the specified application window. Using `deBeginConfigurePlugins` before adding or removing plugins helps in reducing calls to the `hiApplyWinConfig` SKILL function.

When `deBeginConfigurePlugins` is called, Design Editor postpones applying workspace in the specified application window until `deEndConfigurePlugins` is called in the same window.

Note: Nested calls of `deBeginConfigurePlugins` and `deEndConfigurePlugins` are not supported.

Arguments

w_window	An application window.

Value Returned

`nil`	Always returns `nil`. The value is not significant.

Example

```
(deBeginConfigurePlugins (window 2))
```

Informs the Design Editor that a series of calls for `deInstallPlugin` or `deRemovePlugin`, or both will be made in `window:2`.

© 2020 All Rights Reserved.

deEndConfigurePlugins

```
deEndConfigurePlugins(
     w_window
     => nil
```

Description

Indicates an end to a series of calls for `deInstallPlugin` and `deRemovePlugin` marked by the `deBeginConfigurePlugins` in the same application window.

Note: Nested calls of `deBeginConfigurePlugins` and `deEndConfirutePlugins` are not supported.

Arguments

w_window	Application window in which `deBeginConfigurePlugins` was called.

Value Returned

`nil`	Always returns `nil`. The value is not significant.

Example

```
(deEndConfigurePlugins (window 2)
```

Indicates the end of the series of calls for `deInstallPlugin` and `deRemovePlugin` marked by the `deBeginConfigurePlugins` in `window:2`.

© 2020 All Rights Reserved.

deFindAssistant

```
deFindAssistant(
     t_assistant
     w_sessionWindow
     [ ?search_object_name { t | nil } ]
     [ ?search_window_name { t | nil } ]
     [ ?in_active_tab { t | nil } ]
     )
     => w_dwindow / nil
```

Description

Returns the dock window ID associated with the specified assistant name in the specified session window. By default, the function returns the assistant whose window name matches *t_assistant* in the currently active tab of *w_sessionWindow*. You can optionally widen the search to include object names in all tabs of the specified session window.

Arguments

t_assistant	Name of the assistant to be searched.
w_sessionWindow	The Session window in which the assistant needs to be searched.
?search_object_name	Searches for an assistant with object name matching *t_assistant*. Default value: nil
?search_window_name	Searches for an assistant with window name matching *t_assistant*. Default value: t
?in_active_tab	Searches only in the currently active tab. Default value: t If specified nil, the function can return an assistant that does not belong to the application installed in the active tab of the *w_sessionWindow*.

© 2020 All Rights Reserved.

Value Returned

w_dwindow	The dock window ID of the assistant with the specified window or object name in the specified session window
nil	The named assistant was not found installed in the specified session window

Example

The example below shows how to use this function to map or unmap the Navigator assistant in the currently active tab of session window 2:

```
hiUnmapWindow( deFindAssistant( "Navigator" swindow(2)))
hiMapWindow( deFindAssistant( "Navigator" swindow(2)))
```


© 2020 All Rights Reserved.

deGetPlugins

```
deGetPlugins(
     w_window
     [ g_activeOnly ]
     )
     => l_stdobj / nil
```

Description

Returns a list of all plugin instances installed in the given application window

Arguments

w_window	Specifies the ID of the application window (tab window) from which you want to get a list of instances of all plugins
g_activeOnly	(Optional) If non-nil, the function returns only the instances of active plugins in a given window. The default value is `nil`.

Value Returned

l_stdobj	Returns a list of all instances of the plugins installed in the given application window
`nil`	Returns `nil` if no plugin is found installed in the given application window

© 2020 All Rights Reserved.

deGetAppClassInfo

```
deGetAppClassInfo(
     g_application
     )
     => dpl_appClassInfo
```

Description

Returns a copy of the registration information (DPL) associated with the given application

Arguments

g_application	Specifies the application name (string) of the plugin, the class name (symbol) of the plugin, or an instance (stdobj) of the plugin.

Value Returned

dpl_appClassInfo	Returns a copy of the registration information (DPL) associated with an application.
nil	Returns `nil` if the plugin is not registered by using `deRegPlugin`.

Example

```
(deGetAppClassInfo "My Plugin")
 =>(nil appType plugin appName "My Plugin" className MyPlugin superClass DEPlugin
bindkeys nil help "" pluggableMainApps ("Schematics" "Schematics XL")
pluggableViewTypes nil shortDescription "")
```

The following public methods of DE plugin can be overridden by plugin applications. These methods are named with prefix *deo* by following the Cadence SKILL programming standards, The character 'o' indicates overridable method.

Note: Methods that are not prefixed with *deo* should not be re-implemented by code groups outside DE.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

deoEnableTrigger

```
deoEnableTrigger(
     g_application
     [ l_args ]
     )
     => t / nil
```

Description

Determines if the application is enabled for use by the current user. The enable trigger will normally do any appropriate security checks necessary to make this determination. Design Editor calls the enable trigger before installing a plugin and then installs the plugin only if this trigger returns a non-nil value. The default deoEnableTrigger permits a plugin to be installed on top of an application as long as it is pluggable to the application.

If licensing or other restriction is desired, then the plugin code should define the enable trigger. A subplugin application should call next method in its deoEnableTrigger if it requires the licensing of the super plugin application.

Arguments

g_application	Specifies an instance (stdobj) of the plugin
l_args	Single argument, which is a disembodied property list. `l_args->action` property equals `'activate` in addition to the properties specified for DE main application triggers. **Note:** The trigger argument (*l_args*) for DE plugin is similar to the Trigger Argument given to DE main application triggers. Two new values are added for the `args->action` property, they are `'activate`, and `'deactivate`. These are used for installing and removing a plugin in a window, respectively.

Value Returned

`t`	Returns `t` iff the specified application passes security check.
`nil`	Returns `nil` if the application should not be installed.

Example

```
defmethod( deoEnableTrigger ((this MyPlugin) @optional args)
```


© 2020 All Rights Reserved.

```
(and (callNextMethod this args) (check_my_license this)))
```


© 2020 All Rights Reserved.

deoExitTrigger

```
deoExitTrigger(
     g_application
     [ l_args ]
     )
     => t / nil
```

Description

When a plugin is being removed from a window, Design Editor calls the exit trigger before disabling all GUI components of the plugin and removes the plugin only if this trigger returns a non-nil value. The return value of the exit trigger is ignored if the window on which the plugin is installed is being closed. The default implementation is a no-op, and always returns `t`.

Arguments

g_application	Specifies an instance (stdobj) of the plugin
l_args	(Optional) Single argument, which is a disembodied property list. *l_args->action* property equals ‘`deactivate` in addition to the properties for DE main application triggers for removing the plugin. **Note:** The trigger argument for DE plugin is similar to the Trigger Argument given to DE main application triggers. Two new values are added for the `args->action` property, they are `'activate`, and `'deactivate`. These are used for installing and removing a plugin in a window, respectively.

Value Returned

`t`	Returns `t` to permit the removal of the plugin
`nil`	Returns `nil` if the removal of the plugin cannot be successfully initiated. The return value is ignored when the window in which the plugin is running is being closed.

Example

```
defmethod( deoExitTrigger((this MyPlugin) @optional args)
    (my_pre_exit_process args)
    t )
```


© 2020 All Rights Reserved.

deoGetAssistantList

```
deoGetAssistantList(
     g_application
     )
     => l_assistNameandPlacement / nil
```

Description

Returns a list of *t_assistant_name* and *s_placement* pairs (called an association list and contains a list of key and value pairs). The default implementation returns all assistants defined by this plugin and super-plugins.

Arguments

g_application	Specifies an instance (stdobj) of the plugin

Value Returned

l_assistNameandPlacement

Returns an association list of all assistant name-placement pairs defined by this plugin and super-plugins.

`nil` Returns `nil` if no assistants are previously registered with the plugin.

Example

```
defmethod( deoGetAssistantList ((this MyPlugin))
 '(("Search" right) ( "Tree Assistant" left) ( "OI Assistant" left)))
```


© 2020 All Rights Reserved.

deoGetBindkeys

```
deoGetBindkeys(
     g_application
     )
     => l_bindkeys / nil
```

Description

Returns a copy of bindkeys defined for the plugin in the format that is acceptable by `hiSetBindKeys()`. When installing a plugin, DE calls `hiSetBindKeys` using the value of `deoGetBindkeys` method. A default implementation of `deoGetBindkeys` is provided by DE, which returns a list of bindkeys collectively defined by a plugin and its super-plugin(s) through `deRegPlugin ?bindkeys` argument. The bindkeys returned are ordered from generic to specific.

Arguments

g_application	Specifies an instance (stdobj) of the plugin.

Value Returned

l_bindkeys	Returns a collective list of bindkeys defined by the plugin and its super class.
`nil`	Returns `nil` if no bindkeys are registered with the plugin.

Example

```
(defmethod deoGetBindkeys ((this MyPlugin))
    (append (callNextMethod this) my_bindkeys))
```


© 2020 All Rights Reserved.

deoMenuTrigger

```
deoMenuTrigger(
     g_application
     l_args
     )
     => l_menus
```

Description

Returns a list of menus to be installed in the banner of a window. The menu trigger is called when the Design Editor installs and removes the plugin. Whenever there's an application switching event, `deoMenuTrigger` is called again for plugins that are pluggable to the application being switched into. For example, given a plugin whose `pluggableViewTypes` equals `schematic`, the plugin is pluggable to all schematic-based applications. So, if the plugin is currently installed in a VSE window then, if you launch VSE XL in the same window the menu trigger will be called again. Also push down, or return back into another schematic cell-view, the `deoMenuTrigger` is called again. The default `deoMenuTrigger` is a no-op. It returns *`args->bannerMenus`* as is.

Arguments

`g_application`	Specifies an instance (stdobj) of the plugin
`l_args`	Single argument, which is a disembodied property list. The trigger argument (*`l_args`*) for DE plugin is similar to the Trigger Argument given to DE main application triggers. In addition to the properties for DE main application menu triggers, two new values are added for the `args->action` property, they are `'activate`, and `'deactivate`. These are used for installing and removing a plugin in a window, respectively. `args->bannerMenus` equals the list of menus in the main window when the menu trigger is called.

Value Returned

`l_menus`	Returns a list of menus to be installed in the banner

Example

This example adds a menu `My Menu` at the end of the banner menu when `MyPlugin` is installed and removes the menu when the plugin is removed.

© 2020 All Rights Reserved.

```
defclass( MyPlugin (DEPlugin) ())

defmethod( deoMenuTrigger ((this MyPlugin) args)

  (if (eq args->action 'activate)

      (append args->bannerMenus (myMenuTrigger args))

    ;; else

    (when (eq args->action 'deactivate)

        (setof mi args->bannerMenus (nequal mi->hiMenuSym MyMenu)))))
```


© 2020 All Rights Reserved.

deoOnTabSwitch

```
deoOnTabSwitch(
     g_application
     w_sessionWin
     w_intoWin
     g_outofWin
     )
     => g_value
```

Description

An event notification function called during HI calling post tab switch procedures. The order in which calls are made to all `deoOnTabSwitch` functions is not defined.

Arguments

g_application	Specifies an instance (stdobj) of the plugin.
w_sessionWin	Specifies the ID of the session window in which the tab switch happens.
w_intoWin	Specifies the ID of the window being switched into
g_outofWin	Specifies the window being switched out. This value can be `nil`.

Value Returned

g_value	Return value is ignored.

© 2020 All Rights Reserved.

deoOnApplicationChange

```
deoOnApplicationChange(
     g_application
     l_args
     )
     => g_value
```

Description

An event notification function called when data or application changes in a window, such as when a user opens new data or installs a new application in a window, traverses down or up a design hierarchy, or opens another cellview. This function is called for all active plugins at the same time the DE change-application-trigger is called. The order in which calls are made to all `deoOnApplicationChange` functions is not defined.

Arguments

g_application	Specifies an instance (stdobj) of the plugin.
l_args	Single argument, which is a disembodied property list. The trigger argument (*l_args*) for DE plugin is very similar to the Trigger Argument given to DE main application triggers. In addition to the properties for DE main application menu triggers, two new values are added for the `args->action` property, they are `'activate`, and `'deactivate`. These are used for installing and removing a plugin in a window, respectively.

Value Returned

g_value	Return value is ignored.

© 2020 All Rights Reserved.

deoPostInstallTrigger

```
deoPostInstallTrigger(
     g_application
     l_args
     )
     => g_value
```

Description

The post-install trigger is called after all other processing is done, including installation of banner menu, toolbar, assistant, bindkeys. This is provided for those applications that need to finish up any initialization that must occur after GUI is installed.

Arguments

g_application	Specifies an instance (stdobj) of the plugin
l_args	Single argument, which is a disembodied property list. The trigger argument (*l_args*) for DE plugin is very similar to the Trigger Argument given to DE main application triggers. In addition to the properties for DE main application menu triggers, two new values are added for the `args->action` property, they are `'activate`, and `'deactivate`. These are used for installing and removing a plugin in a window, respectively.

Value Returned

g_value	Returns `non-nil` value to indicate success

Example

```
let(()
    defmethod( deoEnableTrigger( (this MyPlugin) @optional args)
        mylicenseCheck( args))
    defmethod( deoMenuTrigger( (this MyPlugin) args)
        (if  (eq args->action 'activate)
        append( args->bannerMenus (myMenuMaker args))
        ;;else
        (when (eq args->action 'deactivate)
        (remove_my_plugin_menu args->bannerMenu))))
```


© 2020 All Rights Reserved.

```
    defmethod( deoPostInstallTrigger((this MyPlugin) args)
        (my_post_installer args)
        t )
    defmethod( deoExitTrigger((this MyPlugin) @optional args)
        (my_pre_exit_process args)
        t )
) ;; End of MyPlugin
```


© 2020 All Rights Reserved.

deoRequireToolbars

```
deoRequireToolbars(
     g_plugin
     )
     => l_toolbarNames
```

Description

Returns a list of toolbar names (string) that are to be customized by the plugin. A plugin can customize toolbars of a main application, or those of another plugin. For a plugin to declare which toolbars need customization, re-define method `deoRequireToolbars`.

Arguments

g_plugin	Specifies an instance (stdobj) of the plugin

Value Returned

l_toolbarNames	Returns a list of toolbar names that will be customized by the plugin

Example

```
(defclass MyPlugin (BasePlugin) ())

  (defmethod deoRequireToolbars ((this MyPlugin))
    (cons "View" (callNextMethod this)))
```


© 2020 All Rights Reserved.

deoUpdateToolbar

```
deoUpdateToolbar(
     g_plugin
     t_name
     o_toolbar
     l_args
     )
     => g_value
```

Description

Customizes a toolbar that is requested by `deoRequireToolbars`. DE calls this method during installing and removing a plugin.

Arguments

g_plugin	Specifies an instance (stdobj) of the plugin.
t_name	Specifies the name of the toolbar.
o_toolbar	Specifies a handle to the toolbar.
l_args	Single argument, which is a disembodied property list. The trigger argument (*l_args*) for DE plugin is very similar to the Trigger Argument given to DE main application triggers. In addition to the properties for DE main application menu triggers, two new values are added for the `args->action` property, they are `'activate`, and `'deactivate`. These are used for installing and removing a plugin in a window, respectively.

Value Returned

g_value	Return value is ignored.

Example

```
(defmethod deoUpdateToolbar ((this MyPlugin) (name string) (toolbar hiToolBar)
args)
    (if (equal name "View")
        (if (eq args->action 'activate)
            (customize_toolbar toolbar args)
          (revert_customization toolbar args))
```


© 2020 All Rights Reserved.

```
(callNextMethod this name toolbar args)))
```


© 2020 All Rights Reserved.

deoOnEvent

```
deoOnEvent(
     g_app
     s_event
     @rest args
     )
```

Description

This event handler can be reimplemented in a subclass to receive various events.

Arguments

g_app	Name of the application for which `deoOnEvent` needs to be called.
s_event	Name of an event specified as a symbol.
`@rest` *args*	Arguments specific to the event.

Value Returned

None

© 2020 All Rights Reserved.

Client Interface

This section describes routines that implement the basic operations of the Design Editor, such as opening a window, pushing into a new design, returning, and so forth. Lower level routines used in the trigger implementations are described in later sections.

deFileNew

```
deFileNew(
     )
     => None
```

Description

Opens the *Create New File* form.

`deFileNew` differs from `deNew` in that it ensures that the *new tab*, *current tab*, and *new window* radio buttons are displayed on the form.

Arguments

None

Value Returned

None

© 2020 All Rights Reserved.

deOpen

```
deOpen(
     [ l_fileSpec [ g_winSpec [ t_accessMode [ t_appName g_sessionWindow
     g_saveSessionHistory] ] ] ]
     )
     => w_window / nil
```

Description

Takes a design specification and a window specification and attempts to install the design in the specified window.

The following environment variables set the defaults for the *Open File* form:

```
graphic defaultNewCellName
graphic defaultNewLibName
graphic defaultNewViewName
graphic defaultOpenCellName
graphic defaultOpenLibName
graphic defaultOpenViewName
```

If a configuration cellview is given as an argument, a SKILL symbol named `'hierEditor` is returned. The value for `'hierEditor` varies based on the type of cellview opened.

- When a configuration cellview as well as a top level schematic is opened, the return value of `hierEditor` is the window ID of the open top level schematic.
- When only a top level schematic is opened (without opening a configuration cellview), the value of `hierEditor` is the window ID of the open top level schematic.
- When only a configuration cellview is opened in the hierarchy editor (a top level schematic is not opened) the value of `hierEditor` is `t`.
- When neither a configuration nor a schematic is opened, the value of `hierEditor` is `nil`.

The cdsenv variable `openConfig` controls the behavior for opening a configuration cellview. The cdsenv variable `topCvMode` controls the access mode for the top cellview of the configuration cellview, that is, the schematic cellview. For more information, refer to Environment Variables.

Note: If the open failed, `nil` is returned.

© 2020 All Rights Reserved.

Arguments

l_fileSpec Disembodied property list which can be `nil` or contain one or more of the following elements:

`l_fileSpec->libName`	Cellview library name	(string)
`l_fileSpec->libId`	Cellview library ID	(dbId)
`l_fileSpec->cellName`	Cellview cell name	(string)
`l_fileSpec->viewName`	Cellview view name	(string)
`l_fileSpec->viewId`	Cellview view ID	(dbId)
`l_fileSpec->viewNameList`	*viewName* switch list	(string)
`l_fileSpec->accessMode`	Desired access mode	(string)
`l_fileSpec->instanceName`	Instance name specification	(string)
`l_fileSpec->hierarchy`	Instance hierarchy specification	(string)
`l_fileSpec->configL`	Hierarchy manager configuration	(list)

If enough information is supplied for DE to verify the existence of the design without further interaction, it will do so. If not, then DE displays a form asking the user to supply the missing arguments. Either the library name or library ID can be supplied. If both are supplied, they must be consistent or a warning is given. Similarly, either the view name or the view ID can be supplied, and if both are supplied, they must be consistent.

The view name list field is optional, and if supplied, is used to determine the view switched into. The view name list is a string which contains a list of view names. The first one in the list for which a corresponding cellview can be found in the given library is used to open the cell.

© 2020 All Rights Reserved.

`l_fileSpec->hierarchy` is an optional property which describes a hierarchy to be pushed on top of the opened data. `l_fileSpec->cellName`, etc., thus specifies the top-level data and `l_fileSpec->hierarchy` specifies successive levels. All levels are pushed in one command, thus making this a convenient way to open a hierarchy and put each level onto the stack. All triggers for each level of hierarchy are called as the window stack is built. For levels which are not the final edit level, the triggers will be called with `l_args->action == "install"` and `l_args->pushed == t`. If `deOpen`, `deSwap`, or `dePush` was executed, then all levels are push edited. If `deEditInPlace` was executed, then all levels are edited in place. There is currently no way to mix edit in place with push operations with this feature.

`l_fileSpec->hierarchy` lets the client open a hierarchy to any arbitrary level. The top-level installed is defined by the other file spec fields. The hierarchy field specifies a list of items to be pushed on top of that item. The syntax of the hierarchy string is designed to allow the maximum flexibility in installing items in the window stack.

`l_fileSpec->instanceName` is an optional property which is used for `dePush` and `deEditInPlace` operations. It specifies an instance name, in the same format as an individual hierarchy item, that refers to the library, cell, and view name properties of the file spec. The instance name is the name of an instantiation of that cellview in the edit cellview of the window which is supplied to the `dePush` or `deEditInPlace` command. When *`instanceName`* is supplied, the library, cell, and cellview properties are optional, and the *Open File* form will only appear if the instance cannot be found in the current edit cellview in the supplied window. The *`instanceName`* property is ignored if the window has no current edit cellview (that is, has an empty stack) or if the supplied *`g_winSpec`* is not a valid Design Editor window.

© 2020 All Rights Reserved.

The syntax of `l_fileSpec->hierarchy` is as follows:

```
HierarchySpecification ::= '/' | '/' HierItemSpec |
                                                    '/'
     HierItemSpec HierarchySpecification
     HierItemSpec ::= InstItemSpec | InstItemSpec
     ModeSpec
     InstItemSpec ::= InstMosaicSpec | InstMosaicSpec '('
     ViewList ')' | DMSpec
     ModeSpec ::= ':' ModeName
     ModeName ::= 'r' | 'w' | 'a' | 'read' | 'overwrite'
     | 'edit'
     InstMosaicSpec ::= InstSpec | MosaicInstSpec
     InstSpec ::= InstName | InstName IterationSpec
     IterationSpec ::= '<' IterationList '>'
     IterationList ::= IterationItem | IterationItem ','
     IterationList
     IterationItem ::= Number | Number ':' Number
     MosaicInstSpec ::= MosaicName '[' Number ',' Number
     ']'
     DMSpec ::= '{' DMItem '}'
     DMItem ::= LibName ' ' CellName ' ' ViewNameSpec |
     LibName ' ' CellName ' ' ViewNameSpec ' ' VersionName
     ViewNameSpec ::= ViewName | '(' ViewNameList ')'
     ViewNameList ::= ViewName | ViewName ' '
     ViewNameList
```

The `InstName` is the name given to an instance when it is created in the parent cellview. All instance names in the hierarchy list are relative to the immediate parent. Mosaic instances are described by the name of the containing mosaic and the row and column of the instance in question. The iteration list for an instance is designed to capture any subset of the valid iterations of an iterated instance.

The `DMSpec` lets the client supply a complete description of the hierarchy item desired, independent of any instance relationship with the parent. The format of a `DMSpec` item lets the user supply a *`viewNameList`* as part of the specification. It should be noted that when a `DMSpec` is supplied to open a hierarchy item, programs which depend on the instantiation relationship between that item and its parent will not work because a specific instance is not supplied as part of the specification.

© 2020 All Rights Reserved.

Each item in the list can have its own *accessMode* explicitly stated. If an item has not explicit *accessMode*, it is defaulted to that supplied with the top-level file spec or `deOpen` call.

The `configL` is a member of the `fileSpec` that defines a Hierarchy Manager configuration. It is a list of properties (*`t_libName`*, *`t_cellName`*, *`t_viewName`*, *`t_accessMode`*) describing the cellview of the configuration.

Note: *`t_accessMode`* is ignored and the configuration is always opened in read mode.

The `dePush` and `deEditInPlace` commands can be called with a file spec that contains only a hierarchy item, if a valid window is passed that contains at least one valid stack item. The hierarchy is pushed on the valid stack item as if that stack item were the primary specification in the file spec. If no valid stack item exists on the window, (or `deOpen` or `deSwap` is called with only a hierarchy specification) then the first item in the hierarchy list must be a `DMSpec` item. Otherwise, the *Open File* form is opened, and the user must specify the top cellview explicitly. In this case, the first item on the hierarchy specification is pushed onto this top cellview. If the first item in the hierarchy specification is an instance, it must be present in the current item on the window stack.

In the process of opening the hierarchy, if errors are encountered, the *Open File* form is opened to allow the user to correct these errors. Cancelling this form aborts the process, leaving the stack with all items successfully opened to that point.

The *Open File* form contains fields for the library name, cell name, and view name. Once this form comes up, the user can change any of these fields as desired, with the result that the design opened might have no relationship to the values supplied in the original call to `deOpen`. At any time the user can decide to cancel the operation, with the result that no design is opened.

The *Open File* form also contains a command button to open the Library Browser window. The *Open File* form is synchronized with the browser so that *Select* operations in the Library Browser window update the form.

`g_winSpec`

Determines whether to use an old window or create a new one, and if a new one, what its creation parameters are.

© 2020 All Rights Reserved.

w_window	Uses the given window.
nil	Creates a window in a default location.
l_bBox	Creates a window with the given *bBox.*
'interactive	Creates a window, user specifies *bBox.*
	If an existing window is given, the current contents will be removed after the data trigger returns successfully. Then the new design as is installed as the top-level design. If the stack could not be successfully cleared, the data trigger will be called again with action “close” to close the data file. If a new window is created, it will not be displayed until the end of the process. This minimizes the amount of work involved in the creation and display of the various parts of the window.
	If *g_winSpec* is defaulted (or nil is passed), the view is checked for the property defaultWindowBox. If this property exists, that box is used for the window. If that property does not exist, the Design Editor chooses its own box.
t_accessMode	Access mode desired for the design, and will be edit or read. (These correspond, respectively, to the UNIX modes a and r, which can also be used.) If the user does not have a license for the application required to edit the design, the access mode passed to the trigger might differ. In this case, the user is given a chance to abort the open sequence.
	The access mode can be supplied as an optional parameter after *g_winSpec* or as part of *l_fileSpec*. This is for compatibility with older code. If both exist and are different, *t_accessMode* overrides l_fileSpec->accessMode. If neither is specified, then the access mode defaults to *edit*.
t_appName	Name of an application sub type which is to be the application installed when the window is opened. *t_appName* must be the name of some sub type in the sub type tree registered with the view type of the specified view.
	If it is found, the root class registration is used to open the data and create the context, but the sub type hierarchy is used to determine licensing and bindkeys, and the application’s menu trigger is called to generate the banner menus.
g_sessionWindow	If the session window is specified, tabs are used instead of new windows.

© 2020 All Rights Reserved.

g_saveSessionHistory

If set to `t`, the function stores the "top level opens" to history. The opens are saved to history in the following scenarios:

(a) When the function is called with `l_fileSpec->libName/cellName/viewName`, and without `l_fileSpec->hierarchy` and `l_fileSpec->hierList`. This is similar to the case when a cellview is opened from Library Manager or Hierarchy Editor.

(b) When the function is called without `l_fileSpec->libName/cellName/viewName`, but with `l_fileSpec->hierarchy` such that only a single level of hierarchy is specified.

Note: In both (a) and (b) above, if the `l_fileSpec` argument is not specified or is partially specified as shown in the following examples, the *File—Open* form will be launched, however, the history will be saved.

```
l_fileSpec <- '(nil libName "amsPLL" cellName
"vco")

l_fileSpec <- '(nil hierarchy "/{amsPLL vco}")
```

Note: If the `l_fileSpec` argument is not specified, is empty (`""`), or `nil`; the value of the `g_saveSessionHistory` argument is not considered, and the `deOpen` function will store all opens to history.

`deOpen` returns `nil` if a window could not be created or a design could not be opened. Otherwise it returns the window passed in or created.

Value Returned

w_window The name of the window opened. If a window was supplied as input, then it can be returned unless it could not be used for some reason.

Examples

```
deOpen()
```


© 2020 All Rights Reserved.

Brings up the *Open File* form, asking for the user to specify the library, cell, and view names of a cellview to edit. When this form is answered, a window is created to contain the cellview.

```
myFileSpec = list( nil 'libName "test" 'cellName "ff" 'viewName "graphic" )
deOpen( myFileSpec list( 200:200 800:800 ) )
```

Opens a window, which is given the screen box (200:200 800:800), and installs the cellview `test ff graphic` in it.

```
myFileSpec = list( nil 'hierarchy "/{test ff graphic}:
a/I4:r" )
deOpen( myFileSpec 'interactive )
```

Prompts the user to specify a screen box for a window, then installs the cellview "`test ff graphic`" in it in edit mode, pushes into instance `I4` in read mode, and finally displays the window.

```
deOpen( myFileSpec winSpec "r" "Layout Viewer" )
```

(ICADVM20.1 Only) Opens the cellview in `Layout Viewer`.

In ICADVM20.1, when the fourth argument, *`appName`* is specified as "`Layout Viewer`" the Viewer is used to open the design. If *`appName`* is specified as "`Layout`", application used to open the design depends on the *`accessMode`*. If *`accessMode`* is specified as `"r"`, the Viewer is used, otherwise Layout XL is used.

© 2020 All Rights Reserved.

deFileOpen

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

Description

Similar to `deOpen()`. Radio buttons on the Open form are displayed that allow selection between opening the cellview in a new tab, current tab, or a new window.

Arguments

None

Value Returned

w_window	The name of the window opened.

Example

```
deFileOpen()
```


© 2020 All Rights Reserved.

deNew

```
deNew(
     [ l_fileSpec ]
     [ g_winSpec ]
     [ t_accessMode ]
     [ t_appName ]
     )
     => w_windowId / nil
```

Description

Creates a new cellview and opens it in a window.

If the library name, cell name, view name, and cellview type are not specified in *l_fileSpec*, or if *l_fileSpec* is not specified, the New File form opens, partially filled out with the information specified in *l_fileSpec*. The New File form requires a library name, cell name, view name, and application.

The following environment variables set the defaults for the New File form:

```
graphic defaultNewCellName
graphic defaultNewLibName
graphic defaultNewViewName
graphic defaultOpenCellName
graphic defaultOpenLibName
graphic defaultOpenViewName
graphic defaultToolName
graphic viewNameToTypeList
```

The environment variable graphic `viewNameToTypeList` maps view names to view types in the New File form. This variable contains the user's mappings, which are in addition to, and take precedence over, the built-in mappings (listed below). This variable is automatically updated (in memory) each time a new *viewName*/*viewType* is used. You can save new mappings with the *Options – Save Defaults* command.

The view name to view type mapping is stored in memory and the view type-to-tool-mapping is stored in the registry.

When you enter a view name, the view names to view type mapping is used to determine the view type and then the registry is used to set the tool cyclic. When a tool is chosen, the registry is read to determine the view type and then the view name to view type mapping is used to set the view name.

This automatic setting of tool or view name happens until you manually change the updated field. This means if you choose a tool and then change the view name, the tool is not changed and vice-versa.

© 2020 All Rights Reserved.

If you enter a new view name which does not yet map to a view type a dialog box appears that lets you select another tool or keep the current one.

The following are the built-in view name to view type mappings:

```
      Name                    Type
( "abstract"              "maskLayout")
( "cmos.sch"              "schematic")
( "compacted"             "maskLayout")
( "extracted"             "maskLayout")
( "layout"                "maskLayout")
( "schematic"             "schematic")
( "symbol"                "schematicSymbol")
( "symbolic"              "maskLayout")
( "config"                "config")
```

The following are the built-in view type to view name mappings:

```
      Type                      Name
( "schematic"                 "schematic")
( "maskLayout"                "layout")
( "schematicSymbol"           "symbol")
( "config"                    "config")
```

If a configuration cellview is given as an argument, a SKILL symbol named `'hierEditor` is returned. The value of `hierEditor` is as follows:

When a configuration cellview is opened in the hierarchy editor, the value of `hierEditor` is `t`. If a cellview is not open or fails to open, the value of `hierEditor` is `nil`.

Note: When opening a new configuration cellview the matching top level schematic is not defined so it is not possible to return a window ID for the value of `'hierEditor`.

Note: The API also stores the history for all new cellviews.

Arguments

`l_fileSpec` Symbol with the following property list:

- *`libName`* is the library name
- *`cellName`* is the cell name
- *`viewName`* is the view name
- *`viewType`* is the type of data in view directory.
  To view a list of valid values, enter `deGetAllViewTypes()` in the CIW.

© 2020 All Rights Reserved.

	accessMode is the access mode. Valid values: "a" (edit), "r" (read), and "w" clears the old data and edit from scratch.
g_winSpec	Creates a window. The valid values are: ■ nil creates window using .Xdefaults ■ interactive creates a window letting user specify location ■ *l_bBox* creates a window using *l_bBox* as screenBox ■ *w_window* use *w_window*
t_accessMode	File access mode. Should be "w" to allow editing, start from scratch.
t_appName	Overrides default application if supplied. To view a list of valid values, enter deGetAppInfo *viewType* in the CIW.

Value Returned

w_windowId	The ID of the window that opened.
nil	No window opened.

Example

```
deNew(list(nil
?libName "test"
?cellName "top"
?viewName "layout"
?viewType "maskLayout"))
```

Opens a mask layout window for editing that contains the layout view of the cell called top from the test library. Returns the ID of the window.

© 2020 All Rights Reserved.

deNewCellView

```
deNewCellView(
     t_libName
     t_cellName
     t_viewName
     t_viewType
     g_winSpec
     )
     => w_windowId / nil
```

Description

Creates a new cellview and opens it in a window.

If a configuration cellview is given as an argument, a SKILL symbol named `'hierEditor` is returned. The value of `hierEditor` is as follows:

When a configuration cellview is opened in the hierarchy editor, the value of `hierEditor` is `t`. If a cellview is not open or fails to open, the value of `hierEditor` is `nil`.

Note: When opening a new configuration cellview the matching top level schematic is not defined so it is not possible to return a window ID for the value of `'hierEditor`.

Arguments

t_libName	The name of the library in which the cellview will be created.
t_cellName	The cell in which the new cellview will be created.
t_viewName	The view name of the new cellview.
t_viewType	The type of the new cellview.
g_winSpec	Creates a window. The valid Values are: `nil` creates window using `.Xdefaults` `'interactive` creates window letting user specify location *l_bBox* creates window using *l_bBox* as screenBox *w_window* uses *w_window*

© 2020 All Rights Reserved.

Value Returned

w_windowId	The window Id if the new cellview is created.
nil	The new cellview was not created.

Example

```
deNewCellView("test" "top" "layout" "maskLayout" list(list(0 0) list(500 600)))
```

Opens a mask layout window for editing that contains the `layout` view of the cell called `top` from the *test* library and created with the specified bounding box. Returns the ID of the window.

© 2020 All Rights Reserved.

deOpenCellView

```
deOpenCellView(
     t_libName
     t_cellName
     t_viewName
     t_viewType
     g_winSpec
     t_accessMode
     )
     => w_windowId / nil
```

Description

Opens an existing cellview in a window.

If a configuration cellview is given as an argument, a SKILL symbol named `'hierEditor` is returned. The value for `'hierEditor` varies based on the type of cellview opened.

- When a configuration cellview as well as a top level schematic is opened, the return value of `hierEditor` is the window ID of the open top level schematic.
- When only a top level schematic is opened (without opening a configuration cellview), the value of `hierEditor` is the window ID of the open top level schematic.
- When only a configuration cellview is opened in the hierarchy editor (a top level schematic is not opened) the value of `hierEditor` is `t`.
- When neither a configuration or schematic is opened, the value of `hierEditor` is `nil`.

Note: If the open failed, `nil` is returned.

Arguments

t_libName	The name of the library in which the cellview is opened.
t_cellName	The cell to open.
t_viewName	The view name of the cellview.
t_viewType	The type of the cellview.
g_winSpec	The window in which to open the cellview.
t_accessMode	The mode in which to open the cellview. "`a`" (edit), "`r`" (read), and "`w`" (clears the old data and edit from scratch).

© 2020 All Rights Reserved.

Value Returned

w_windowId	The cellview opens.
nil	The cellview did not open.

Example

```
deOpenCellView("test" "top" "layout" "maskLayout" list(list(0 0) list(500 600))
"a")
```

Opens a mask layout window for editing that contains the `layout` view of the cell called `top` from the `test` library and created with the specified bounding box. Returns the ID of the window.

© 2020 All Rights Reserved.

deBack

```
deBack(
     [w_window [ numSteps] ]
     )
     => w_window / nil
```

Description

This API allows you to go back to a particular window event. It takes two optional integer arguments, `w_window` and `numSteps`. If `deBack` is called without any arguments it will open and display the cellview that was displayed prior to the current cellview in the current window.

Arguments

`w_window`	If the `w_window` argument is specified the command will work on the specified window. The default value for `w_window` is the current window.
`numSteps`	If the value of `numSteps` is greater than or equal to 1, you will go directly back to the cellview that would be opened if you executed the `deBack(window)` command `numSteps` times. If the value of `numSteps` equals 0, this will display the *Go Back* form. The default value for `numSteps` is 1.

Value Returned

`w_window`	The window value which you have gone back to.
`nil`	If the value of `numSteps` is more than the number of levels you can go back to, the API will return `nil` and warn that you cannot go that far back.

Example

```
deBack( window(3) 1)
```

Goes back 1 step on window (3).

© 2020 All Rights Reserved.

deForward

```
deForward(
     [w_window [ numSteps] ]
     )
     => w_window / nil
```

Description

This API allows you to go forward to a particular window event. It takes two optional integer argument; `w_window` and `numSteps`. If the command is called without any arguments it will open and display the cellview that was displayed prior to the use of the last deBack command in the current window.

Arguments

w_window	If `w_window` is specified the command works on the specified window. The default value for `w_window` is the current window.
numSteps	If the `numSteps` is greater than or equal to 1, you will go directly forward to the cellview that would be opened by executing the `deForward(window)` command `numSteps` times. If the value of `numSteps` is `0`, this will display the *Go Forward* form. The default value for `numSteps` is 1.

Value Returned

w_window	The window value which you have gone forward to.
`nil`	If the value of `numSteps` is more than the number of levels you can go forward to, the API will return `nil` and warn that you cannot go that far forward.

Example

```
deForward( window(3) 1)
```

Goes forward 1 step on window (3).

© 2020 All Rights Reserved.

dePush

```
dePush(
     [ l_fileSpec [ g_winSpec [ t_accessMode [ t_appName ] ] ] ]
     )
     => w_window / nil
```

Description

The same as `deOpen`, except that if the window already exists, the current contents are not removed. Instead the new design is pushed on top of the existing one.

Arguments

l_fileSpec	File specification for the cellview to be pushed. See `deOpen` for a detailed explanation of the format for this argument.
g_winSpec	Window specification for the window in which to do the push. See `deOpen` for a detailed description of this argument.
t_accessMode	Access mode desired for the new design. Must be either `edit` or `read`.
t_appName	Name of the application to install after the data is placed on the stack. The application must be a part of the view type subclass tree that is indicated by the data.

Value Returned

w_window	Window in which the push is done. Normally this is the same as the window passed in, but if necessary a new window is created and returned

Example 1

```
dePush( nil window(3) )
```

Brings up the *Open File* form, asking for the library, cell, and view names of a cellview to edit. When this form is completed, the cellview opened is pushed on the stack in `window(3)`.

© 2020 All Rights Reserved.

Example 2

Assume *I4* is an instance of a symbol in *window(3)*. Also assume that the schematic view represented by that symbol exists in some library on the current library search path.

```
myFileSpec = list( 'instanceName "I4" 'viewNameList "schematic symbol")
dePush( myFileSpec window(3) "read" "Timing Analysis" )
```

Switches into the schematic master corresponding to instance *I4* in the edit cellview in *window(3)* for reading, then starts up the Timing Analysis application in that window.

© 2020 All Rights Reserved.

deSetHistoryOpenMode

```
deSetHistoryOpenMode(
     t_openMode
     )
     => t / nil
```

Description

Controls whether designs are opened from history in a new tab or a new window. The setting applies equally to opening a design from the CIW as well as from a session window.

Arguments

t_openMode Valid values: `newTab`, `newWindow`, `default`.

- `newTab`: The design selected from history would open in a new tab of the last active session window.
- `newWindow`: The design selected from history would open in a new session window.
- `default`: The design selected from CIW history would open in a new session window and the design selected from session window history would open in new tab.

Value Returned

`t`	The design was successful opened.
`nil`	The opening of the design failed.

Example

```
deSetHistoryOpenMode("newTab")
```

Opens the chosen design in a new tab of the last active session window and returns `t`.

© 2020 All Rights Reserved.

deSwap

```
deSwap(
     [ l_fileSpec [ g_winSpec [ t_accessMode [ t_appName] ] ] ]
     )
     => w_window / nil
```

Description

This function is same as `deOpen`, except that if the window already exists, the current contents are not removed. Instead the current stack item is replaced with the new design.

This corresponds to a trigger argument action of '`swap` only if the view types of the old and new contents are the same. Otherwise, the old contents are explicitly closed and the new contents explicitly opened and installed. If the current contents are being edited in place, the new contents are installed in edit-in-place. If the new contents being swapped consist of a hierarchy, the entire hierarchy is installed in edit-in-place. In this latter case, the '`swap` action will not be passed even when the *viewTypes* are the same.

Arguments

l_fileSpec	File specification for the cellview to be swapped into the given window. See `deOpen` for a detailed explanation of the format for this argument.
g_winSpec	Window specification for the window in which to do the swap. See `deOpen` for a detailed description of this argument.
t_accessMode	Access mode desired for the new design. Valid values: `edit`, `read`, `overwrite`, `a`, `r`, or `w`.
t_appName	Name of the application to install after the data is placed on the stack. The application must be a part of the view type subclass tree that is indicated by the data.

Value Returned

w_window	Window in which the swap is done. Normally this is the same as the window passed in, but if necessary a new window is created and returned.

© 2020 All Rights Reserved.

Example 1

```
deSwap( nil window(3) )
```

Brings up the *Open File* form, asking for the library, cell, and view names of a cellview to edit. When this form is completed, the cellview opened replaces the top of the stack in `window(3)`.

Example 2

```
deSwap( myFileSpec window(3) )
```

Swaps the top of the stack in `window(3)` with the cellview "`test ff graphic`," opening that in read mode, then pushes into instance `I4` in edit mode.

© 2020 All Rights Reserved.

deToggleAssistants

```
deToggleAssistants(
     [ w_sessionWindow ] )
     => t / nil
```

Description

Toggles the visibility of the current assistants off or on to maximize the size of the design window.

Arguments

w_sessionWindow	Session window in which to toggle the assistants.

Value Returned

t	Visibility of the assistants toggles off or on.

Example

```
deToggleAssistants(hiGetSessionWindow(hiGetCurrentWindow))
```


© 2020 All Rights Reserved.

deToggleAssistantVisibility

```
deToggleAssistantVisibility(
     t_assistantName
     [ w_sessionWindow ]
     )
     => t / nil
```

Description

Toggles the visibility of the given assistant in the given session window.

Arguments

t_assistantName	The given assistant name.
w_sessionWindow	The given window. If none is specified, the current session window is taken into consideration.

Value Returned

`t`	The visibility of the given assistant toggles successfully.
`nil`	The visibility of the given assistant does not toggle.

Example

```
deToggleAssistantVisibility("World View")
```


© 2020 All Rights Reserved.

deToggleToolbars

```
deToggleToolbars(
     [ g_sessionWindow ] )
     => l_toolbar / nil
```

Description

Toggles the currently visible toolbars off. If no visible toolbars are found, toggles the previously hidden toolbars on.

Arguments

g_sessionWindow	Session window in which to toggle the toolbars. If not specified, defaults to the session window that has the current window.

Value Returned

l_toolbar	A list of toolbars for which the visibility state has changed.
nil	The visibility state of the toolbars has not changed.

Example

```
deToggleToolbars(hiGetCurrentWindow())
```


© 2020 All Rights Reserved.

deFindToolbar

```
deFindToolbar(
     S_name
     w_sessionWindow
     )
     => o_instance / nil
```

Description

Searches for and returns an instance of the specified toolbar installed in the given session window. This function also supports the Virtuoso design editor toolbars, such as Workspaces, Go, and Bookmarks.

Arguments

S_name	Specifies the design editor toolbars, the value of `text` property (a string), or the `name` property (a symbol) of the toolbar as specified in the toolbar definition file. Unless *S_name* is specified as the name(symbol) of a toolbar, the toolbar is searched for in the active tab window.
w_sessionWindow	Specifies the ID of the session window in which you want to search for the named toolbar

Value Returned

o_instance	Returns an instance of the specified toolbar in the given session window.
`nil`	Returns `nil` if the toolbar you are searching for is not found.

Example

In this example, the *S_name* values for the Schematics toolbar in the toolbar definition file are as follows:

```
(nil
    name schFileToolbar
    text "File"
    items (
         ...
```


© 2020 All Rights Reserved.

You can search for the Schematics toolbar in a session window, say 1, using any of the function calls listed below:

```
(deFindToolbar 'schFileToolbar (swindow 1))
(deFindToolbar "File" (swindow 1))
```

The following example illustrates how you can show the Workspaces toolbar and hide the Bookmarks toolbar in the session window 2:

```
hiShowToolbar( deFindToolbar( "Workspaces" swindow(2)))
hiHideToolbar( deFindToolbar( "Bookmarks" swindow(2)))
```


© 2020 All Rights Reserved.

deManageToolbar

```
deManageToolbar(
     w_sessionWindow
     o_toolbar
     [ g_appName ]
     )
     => t / nil
```

Description

Puts the toolbar handle on the given session window (under DE management) for the specified application. A managed toolbar is displayed as a menu item in *Window–Toolbars* and in the context menu of the session window.

Arguments

w_sessionWindow	Specifies the ID of the session window in which you want to put the given toolbar.
o_toolbar	Specifies a handle of the toolbar that is to be put on the given session window (under DE management).
g_appName	(Optional) Specifies the application name in which the toolbar needs to be put. If no *appName* is given, the application running in the active-tab-window is used.

Value Returned

`t`	Returns `t` if the specified toolbar is successfully put under DE management.
`nil`	Returns `nil` if the specified toolbar is not put under DE management.

Example

```
(deManageToolbar (swindow 2) 'schFileToolbar_2 (hiGetAppType (window 3)))
```


© 2020 All Rights Reserved.

deCustomizeToolbars

```
deCustomizeToolbars(
     [ ?app t_appName ]
     [ ?toolbar t_toolbar ]
     [ ?toolbarItem t_toolbarItem ]
     )
     => t / nil
```

Description

Invokes the Toolbar Manager graphical user interface for customizing toolbars in the Virtuoso environment.

Arguments

`?app` *`t_appName`*

The application ID that must be selected in Toolbar Manager.

`?toolbar` *`t_toolbar`*

The toolbar name that must be selected in Toolbar Manager.

`?toolbarItem` *`t_toolbarItem`*

The toolbar item that must be selected in Toolbar Manager.

Value Returned

`t`	Toolbar Manager is launched.
`nil`	Toolbar Manager is not launched.

Example

The following command opens Toolbar Manager with *Layout* as the selected application, *Edit* as the selected toolbar, and *Edit Properties* as the selected item.

```
deCustomizeToolbars(?app "Layout" ?toolbar "leEditToolbar" ?toolbarItem
"leEditToolbarProps")
```


© 2020 All Rights Reserved.

deUnmanageToolbar

```
deUnmanageToolbar(
     w_sessionWindow
     s_toolbar
     [ g_appName ]
     )
     => t / nil
```

Description

Removes the toolbar from the given session window (under DE management) for the specified application. A toolbar should be removed using the `deUnmanageToolbar` function before it can be deleted.

Arguments

w_sessionWindow	Specifies the ID of the session window from which you want to remove the given toolbar.
s_toolbar	Specifies the name of the toolbar, which was set when the toolbar was put in the given session window.
g_appName	(Optional) Specifies the application name from which the toolbar needs to be removed. If no *appName* is given, the application running in the active-tab-window is used.

Value Returned

`t`	Returns `t` if the specified toolbar is successfully removed from DE management.
`nil`	Returns `nil` if the specified toolbar is not removed from DE management.

Example

```
(deUnmanageToolbar swindow(1) 'mytoolbar)
```


© 2020 All Rights Reserved.

deEditInPlace

```
deEditInPlace(
     w_window
     [ l_fileSpec [ t_accessMode [ t_appName ] ] ]
     )
     => w_window / nil
```

Description

Requires that *w_window* exists and has at least one stack element.

If the window has no stack, this command is the same as `deOpen`*(l_fileSpec w_window t_accessMode)*. Otherwise, the triggers are called with *l_args->eip* equal to `t`. If a hierarchy is specified and the window either does not exist or has no stack, the top level of the hierarchy is not opened in edit in place. However, all other levels of the hierarchy are opened in edit in place.

Arguments

w_window	Window in which to do the edit in place. This must be a valid Design Editor window with at least one stack element.
l_fileSpec	File specification for the cellview to be pushed. See `deOpen` for a detailed explanation of the format for this argument.
t_accessMode	Access mode desired for the new design. Valid values: `edit`, `read`, `overwrite`, `a`, `r`, or `w`.
t_appName	Name of the application to install after the data is placed on the stack. The application must be a part of the view type subclass tree that is indicated by the data. In ICADVM20.1, for a Layout window, if the application name is not specified, the design environment uses the same application that is currently installed in the given Window for the edit-in-place operation.

Value Returned

w_window	Window passed in is returned.

© 2020 All Rights Reserved.

Example

```
deEditInPlace( window(3) nil)
```

Brings up the *Open File* form, asking for the library, cell, and view names of a cellview to edit. When this form is completed, the cellview is pushed on the stack in `window(3)`. The cellview currently being displayed does not change, only the cell within its hierarchy which is being edited changes.

© 2020 All Rights Reserved.

deReturn

```
deReturn(
     w_window [ x_level ]
     )
     => x_level_new / nil
```

Description

Pops the window stack one or more levels.

If *x_level* is defaulted, the stack is popped one level. Otherwise, the stack is popped as many times as necessary to leave the stack with *x_level* elements on it. This routine returns `nil` if the current level could not be popped. Otherwise it returns the resulting edit level, which can be greater than that passed in if one or more levels could not be popped. If *x_level* is -1, everything is automatically popped and the window is closed.

Arguments

w_window	Window in which a stack element is to be popped.
x_level	Stack level to return to. If not specified, the stack is popped one level, which also means that if you are already at the top level, calling `deReturn` without specifying `x_level` will close the window.

Value Returned

x_level_new	Stack level after the return completes.
`nil`	The current level could not be popped.

Example 1

```
deReturn( window(3) )
```

Pops one level off the stack of `window(3)`. Assuming that previously, the window is left in the same state as it was before the call to `deEditInPlace`.

Example 2

```
deReturn( window(3) -1 )
```


© 2020 All Rights Reserved.

Equivalent to `hiCloseWindow( window(3) )`.

© 2020 All Rights Reserved.

deCopy

```
deCopy(
     [ w_window [ g_winSpec ] ]
     )
     => w_windowCopy / nil
```

Description

Makes a complete copy of the given window, including its stack and the context data stored on it.

w_windowCopy is identical to, but distinct from, *w_window*. During the copy process, the data trigger will be called as if you had executed the corresponding sequence of `deOpen/dePush/deEditInPlace` operations. The application trigger will be called with the `'copy` operation and is responsible for initializing the destination context with the data from the source context. If this argument has an ADE L, XL, GXL application installed, no copy operation is performed and a warning message is displayed.

If *g_winSpec* is an existing window, its contents are removed before the copy begins. If this operation fails, the copy does not take place. If g_winSpec is a session window then the new window, which is a copy, is made inside the session window.

In the following example, deCopy copies the current window in a new tab in the same session window:

```
(let ((cw (hiGetCurrentWindow))) (deCopy cw (hiGetSessionWindow cw)))
```

The current edit level of the destination window is set to be the same as that of the source window; if this is not the maximum edit level in the source window, it will not be in the destination window.

Arguments

w_window	Window which is to be copied. Defaults to the current window.
g_winSpec	Window specification for the copy. See `deOpen` for a detailed description of this argument.

© 2020 All Rights Reserved.

Value Returned

w_windowCopy	Copy of *w_window* which is created.
nil	The copy is not made.

Example

The following example returns a window containing the same cellview as n.

```
deCopy( window(3) )
```


© 2020 All Rights Reserved.

Trigger Arguments

The data, application, and user triggers are passed a single argument, which is a disembodied property list. The list can be updated at various times during the process as necessary. All properties will be present in the list at the time the trigger is called; if any property can't be determined, then the operation is a failure and will be aborted before calling any triggers. The list of properties is as follows:

l_args->libName	Cellview library name	(string)
l_args->libId	Cellview library ID	(dbId)
l_args->configName	Configuration name	(string)
l_args->cellName	Cellview cell name	(string)
l_args->viewName	Cellview view name	(string)
l_args->viewId	Cellview view ID	(dbId)
l_args->viewNameList	View name switchlist	(string)
l_args->accessMode	Desired access mode	(string)
l_args->viewType	*viewType* or *viewSubType* of the application	(string)
l_args->rootType	Root *viewType* of the view	(string)
l_args->appName	*appName* of the application being installed	(string)
l_args->help	Help string of the application being installed	(string)
l_args->action	Current action (see below)	(string)
l_args->window	Window in which to install the data	(window)
l_args->srcContext	Source context ("copy" only)	(any data type)
l_args->oldCellView	Previously installed cellview ("swap" only)	(dbId)
l_args->eip	*t* if this is an edit in place, *nil* otherwise	
l_args->pushed	Context push state	(t/nil)
l_args->config	The Hierarchy Manager `configId`, if any, associated with the current window.	
l_args->action	Describes the situation in which the data is being installed in the window. Not all actions apply to all triggers, and when an action doesn't apply, the trigger is not called. The possible actions are:	
	'open Open cellview (data triggers only)	

© 2020 All Rights Reserved.

	’install	Create context for display or stacking
	’close	Stack item going away - close cellview / delete context
	’push	Cellview being pushed underneath another
	’pop	Cellview being popped from underneath another
	’change	Access mode for cellview changed
	’reset	Discard edits executed for this cellview
	’save	Cellview was saved to disk
	’copy	Copy window - stack and contexts are all copied
	’copyInstall	Install after copy
	’swap	Replace the current cellview with a new one of same type
	’activate	Install a plugin in a window
	’deactivate	Remove the plugin from the window
l_args->viewType	The *viewType* or *viewSubType* corresponding to the application which is being installed. *l_args->rootType* is the *viewType* of the root of the class tree containing *l_args->viewType*.	

The triggers return `nil` if the operation fails. For successful returns, in some cases data is returned; in others, `t`.

© 2020 All Rights Reserved.

dataTrigger

```
dataTrigger(
     l_args
     )
     => g_handle / nil
```

Description

The data trigger is called after the DE has collected and verified all of the trigger arguments, checked for the proper application registration, and opened or validated a window for the design.

Arguments

l_args The property *l_args->action* can be `open`, `swap`, or `close`.

- `'open`
  Passed when a new window has been created, or when the user has called `deOpen` with an existing window. In the latter case, the Design Editor will remove the current contents of the window before installing the new design. Thus the new design always becomes the bottom level of the stack in the window. In the process of removing the current contents of the window, triggers for designs on its stack will be called and might necessitate interaction with the user. The data trigger should abort and return `nil` if the specified design cannot be opened.

  To install or stack a design in the window requires that the application create a context and link it to the window. The application trigger is responsible for all context creation and initialization.

© 2020 All Rights Reserved.

- `'swap`
  Passed when the design is being replaced on the window stack by another design of the same view type. `'swap` is equivalent to `'close` on the current design followed by `'install` on the new design, except that because the view type is not changing, the data trigger and app trigger need only be called once, and the context can be overwritten instead of destroyed and recreated.

- `'close`
  Passed when the design is being removed from the window stack. The data trigger can then close the design data.

Value Returned

`g_handle`	Identifier for the opened file.
`nil`	The operation failed and the process is aborted.

© 2020 All Rights Reserved.

appTrigger

```
appTrigger(
     l_args
     )
     => g_context / nil
```

Description

The application trigger is responsible for any necessary banner setup (except for menus), installing the window popup menu, and other application-specific initialization. It is also responsible for managing the application context.

The application trigger is responsible for setting the window interface procedures and otherwise connecting itself to the window. This is referred to as “installing” the application into the window. This will only need to be done when the current application is to be the displayed application; that is, for `'install` and `'pop` actions.

After the data trigger has returned successfully from opening the data, its handle is stored in the window and the application trigger is called with the same argument list, with perhaps a different action. The application trigger creates a context and installs its window procedures. The application trigger returns this context (which is a SKILL data structure), and it will be placed in the stack along with the data handle.

If the application trigger fails, the Design Editor operation in progress is aborted, with a potentially partial result.

Arguments

`l_args->action`

Valid values: `install, push`, `pop`, `close`, `change`, `reset`, `save`, `swap`, `copy`, `copyInstall`

- `'install`
  Indicates this design is about to be installed and displayed in the window. The trigger should create and initialize the context as necessary, including any application specific initialization, and install its popup menu and any bindkeys, etc. If successful, *`g_context`* is returned, at which time it is attached to the stack.

© 2020 All Rights Reserved.

If the item is to go on the stack, but is going to be immediately pushed, then *l_args->pushed* will be `t` and the application need not install pop-ups, etc. The context must still be created and initialized but is not installed in the window. The context should be created in the pushed state, as if the operation `'push` had been called with it.
If this design is being edited in place *l_args->eip* will be `t`. In this case, the application can assume that there is at least one item already stacked. *EditInPlace* stack elements are marked by the Design Editor. If *l_args->pushed* is *nil*, the application might want to reinstall pop-ups, and so forth, but this is not necessary.

- `'push`
  Indicates this design was displayed and about to be pushed underneath a new design. The popup should be removed, and that portion of the context which is to remain around while the design is underneath should be saved. The trigger should return `nil` if this level cannot be pushed, thus aborting the stacking operation. Otherwise, *`g_context`* will replace the context in the associated stack element. If the trigger intends to replace the existing context with a new handle, it is responsible for deleting the existing context.

  If *l_args->eip* is `t`, then the new design was generated by an edit in place. (This is not the same as when the current design was there as a result of an edit-in-place.) In this case, the application might want to do something different with the context.

© 2020 All Rights Reserved.

- `'pop`
  Indicates this design was pushed and the design on top of it was just removed. This action is the same as `'install` but assumes that the context already exists in a pushed state and that some work might be necessary to restore it. If successful, *`g_context`* is the popped context. Otherwise `nil` is returned. If the design was pushed in an edit-in-place operation, *l_args->eip* will be `t`.

- `'close`
  Indicates this design is about to be removed from the stack. All menus, etc. should be removed unless *l_args->pushed* is `t`, in which case this is not necessary because this stack element was not the one displayed. The application context should be deleted. If the design was installed in an edit-in-place operation, *l_args->eip* will be `t`. This returns `t` if successful, `nil` otherwise.

- `'change`
  Indicates the design's access mode was changed by the user. Any context changes (such as disable/enable menu items) that might be required to keep in sync should be done. The return value is ignored.

- `'reset`
  Indicates that the design was just refreshed from disk and edits were discarded. Any context changes (such as deleting references to objects which might have gone away) that might be required should be done. The return value is ignored.

- `'save`
  Indicates that the design was just saved to disk. Any context changes (such as changing the name frame) that might be required should be done. The return value is ignored.

© 2020 All Rights Reserved.

- `'swap`
  Indicates that a new design has replaced the current stack element, and the *app* trigger must re-initialize the context according to the new design. “swap” is equivalent to “close” on the old stack element and “open” on the new one, except that by definition, the same trigger is called in both cases. Hence, the trigger only need be called once, and it will not normally need to reinstall the banner/popup menus, bindkeys, etc. It will have to re-initialize the current context to the new design. `g_context` is returned, and might or might not be the same as the original handle. If not, the *app* trigger is responsible for deleting the previous version. The old cellview is passed as *args->oldCellView*, and the *appTrigger* is responsible for closing this cellview.

- `'copy`
  Indicates that the user is copying the window and the context needs to be copied. In this case, *l_args->srcContext* is the context associated with this stack element in the source window. *l_args->window* is the destination window. *l_args->eip* is the source stack item *eip* state. *l_args->pushed* is `t` if the source stack frame was a pushed frame and `nil` if it was the displayed frame. The application trigger returns `g_context`, which is the copy of the source context, or `nil` if a copy could not be made.

- `'copyInstall`
  Indicates that the context has already been copied, and all that is necessary is to install it in the window.

© 2020 All Rights Reserved.

userAppTrigger

```
userAppTrigger(
     l_args
     )
     => t / nil
```

Description

After the application trigger is called, the user application trigger is called with the same argument list. The user application trigger allows clients to subclass existing application triggers, and thus add additional context to that already managed by the application. The user application trigger's return value is ignored, but should indicate `t` for success and `nil` for failure.

Arguments

`l_args`	List of arguments.

Value Returned

`t`	The execution of the function succeeded.
`nil`	The execution of the function failed.

© 2020 All Rights Reserved.

enableTrigger

```
enableTrigger(
     [ l_args ]
     )
     => t / nil
```

Description

The enable trigger is a special routine loaded separately from the application context, normally by an autoload file.

The enable routine normally takes no arguments, and returns `t` or `nil` depending on whether the application passes or fails, respectively, the security checks required before the user can edit designs with it. This routine must not call any code in the application context. Therefore any routines needed should be included in the autoload file along with the enable trigger itself.

The enable trigger and its associated routines are placed in an autoload file rather than the application context because it will be called to determine whether an application is available to the user. If the application is not enabled, the application context is not needed. Thus putting this code in a separate autoload file eliminates the need to load the application context when the application is not enabled.

When registered with a view type, the enable trigger is called when trying to open a cellview of that view type. If it returns `nil`, the cellview will not be opened, and the Design Editor operation which initiated it fails. When registered in a customizer, the enable trigger is called when the customizer menu is built, thus causing the associated items to be disabled. In this manner, the user will not have access to the tool.

The enable trigger is also called when the user attempts to install a new application either by executing a customizer menu item or by calling `deInstallApp`. In this case, the enable trigger is passed the full set of trigger arguments. The enable trigger can use this argument list to do additional checking after the license check.

Arguments

l_args	List of arguments.

Value Returned

`t`	The execution of the function succeeded.
`nil`	The execution of the function failed.

© 2020 All Rights Reserved.

menuTrigger

```
menuTrigger(
     l_args
     )
     => l_menuList / nil
```

Description

The menu trigger is called when an application is about to be installed as the current application in a window. At the time the menu trigger is called, all other triggers have already been called, and the window is about to be displayed (or if the window was a previously existing window, about to be refreshed).

The menu trigger does not install the menus itself; this is done by the Design Editor. The menu trigger merely returns a list of the menus that are to be installed when needed. The menu trigger is passed the trigger arguments in case it desires to customize its menu list according to any data in `l_args`. When the menu trigger for a parent view type is called due to menu inheritance, those elements of `l_args` derived directly from the *viewType* (*args->viewType* and *args->appName*) will be those derived from the parent type, not the original view type.

Arguments

`l_args`	List of arguments.

Value Returned

`l_menuList`	A list of the menus that are to be installed when needed.
`nil`	The execution of the function failed.

© 2020 All Rights Reserved.

userMenuTrigger

```
userMenuTrigger(
     l_args
     )
     => l_menuList / nil
```

Description

The user menu trigger is called just before the application menu trigger is called and performs the same function. This trigger allows the user to attach menus to an application. Although it is called prior to the application menu trigger, the menu list it returns is appended after the application menus.

The user menu trigger does not install the menus itself; this is done by the Design Editor. The user menu trigger merely returns a list of the menus that are to be installed when needed. The user menu trigger is passed the trigger arguments in case it desires to customize its menu list according to any data in `l_args`. When the user menu trigger for a parent view type is called due to menu inheritance, those elements of `l_args` derived directly from the *viewType* (*args->viewType* and *args->appName*) will be those derived from the parent type, not the original view type.

Arguments

`l_args`	List of arguments.

Value Returned

`l_menuList`	A list of the menus that are to be installed when needed.
`nil`	The execution of the function failed.

© 2020 All Rights Reserved.

postInstallTrigger

```
postInstallTrigger(
     l_args
     )
     => t / nil
```

Description

The post-install trigger is called after all other processing is done, including installation of banner menus and the display of the window. It is available for applications to do any processing that cannot be done before this time, such as adding menu items to an existing menu.

Arguments

l_args	List of arguments.

Value Returned

`t`	The execution of the function succeeded.
`nil`	The execution of the function failed.

© 2020 All Rights Reserved.

userPostInstallTrigger

```
userPostInstallTrigger(
     l_args
     )
     => t / nil
```

Description

Called immediately after the post-install trigger. It serves the same function but is intended for use only by the customer.

Arguments

l_args	List of arguments.

Value Returned

`t`	The execution of the function succeeded.
`nil`	The execution of the function failed.

© 2020 All Rights Reserved.

probeTrigger

```
probeTrigger(
     w_window
     s_action
     [ w_source ]
     )
     => t / nil
```

Description

The probe trigger is called when the Design Editor has installed a new design in a window, whether an existing one or a new one. It is also called when the window is closed. This trigger allows the probing mechanism to keep its data structures consistent without requiring each application to call it separately.

Arguments

w_window	The window being operated on.
s_action	When a new window is created, and a design is installed, the probe trigger is called after the installation is complete with *s_action* == 'install. When a window is closed, the probe trigger is called with *s_action* 'close before any other actions are taken to dismantle the window. When an existing window is having its design changed, the probe trigger will be called twice. First, when the Design Editor calls deClearStack, the probe trigger will be called with *s_action* == 'close before any other action is taken. Then, after the new design is installed, it will be called again with *s_action* == 'install after the installation is complete, just as with the creation of a new window. For *s_action* == 'install or 'close, the third argument *w_source* is nil, and should be ignored.
w_source	If deCopy is used to make a copy of a window, the probe trigger is called with *s_action* == 'copy. In this case, the third argument, *w_source*, is not nil and is the source window of the copy. Thus the probe trigger can copy the probe context to the new window rather than creating a new context. The probe trigger is responsible for setting and maintaining the probe context. Routines exist for setting and retrieving this context, which is otherwise opaque to the Design Editor.

© 2020 All Rights Reserved.

Value Returned

`t`	The execution of the function succeeded.
`nil`	The execution of the function failed.

© 2020 All Rights Reserved.

Licensing Functions

lbuiGetCheckedOutLicenses

```
lbuiGetCheckedOutLicenses(
     )
     => l_list
```

Description

Returns a list of the licenses consisting of each license feature name and the number of licenses (tokens) checked out.

Arguments

None

Value Returned

l_list	A list of the checked out licenses.

Example

(IC6.1.8 Only)

```
lbuiGetCheckedOutLicenses()
=>
("Virtuoso_Schematic_Editor_XL" 1
"Virtuoso_Layout_Suite_GXL" 18)
```

(ICADVM20.1 Only)

```
lbuiGetCheckedOutLicenses()
=>
("Virtuoso_Schematic_Editor_XL" 1
"Virtuoso_Layout_Suite_EXL" 18)
```


© 2020 All Rights Reserved.

lbuiPrintLicenseUseReport

```
lbuiPrintLicenseUseReport(
     [ ?products g_productID ]
     [ ?filename ltx_fileName ]
     )
     => g_report
```

Description

Displays the license usage report (for example: user logins, host names, display time, number of licenses, and license version) in the CIW or the specified output log file.

Arguments

`?products` *`ltx_productID`*

A list of integers or strings representing DFII product license numbers.

`?filename` *`t_fileName`*

A string representing the full path to the output log file. If you do not specify `?filename`, the report is printed in the CIW.

Value Returned

`g_report` The license usage report.

Example

```
lbuiPrintLicenseUseReport(?products (list 95011 111))
```

The above function prints the license usage report for license 95011 and 111 to the CIW, as given below.

```
*WARNING* (lbuiPrint-3) The product 95011 is not a DFII product or a license.
******************************************************************************
  License 111:
******************************************************************************
Server: 5280@noiflex  Total: 440000  Available: 439876
User     Host         Display            How many  Minutes  Version  Capability
----     ----         -------            --------  -------  -------  ----------
aras     cosmic2016    rlblr-aras:1.0          1      30383    6.16
```


© 2020 All Rights Reserved.

```
angupta   cosmic2024    rlblr-angupta:2.0       1      29147    6.15
chaitanm  cosmic2026    rlblr-chaitanm:1        1      28948    6.15
```

```
lbuiPrintLicenseUseReport(?products (list 95310) ?filename "/home/<usr2>/
licenseusage.log")
```

The above function prints the license usage report for license 95310 to the file named `licenseusage.log`, as given below.

```
*******************************************************************************
  License Virtuoso_Layout_Suite_XL:
*******************************************************************************
   Server: 5280@noiflex  Total: 20000  Available: 19995
User      Host          Display          How many  Minutes  Version  Capability
----      ----          -------          --------  -------  -------  ----------
apparao   blrfxe001     rlblr-apparao:1.0     1       286     6.15
kpriya    blrfxe001     rlblr-kpriya:3.0      1       142     6.15
pratikj    cosmic2040      rlblr-pratikj:1.0      1         85       6.16
asanwal    rlno-asanwal    :4                     1         28       6.17
```


© 2020 All Rights Reserved.

iclicCheckOutAndLock

```
iclicCheckOutAndLock(
     t_feature
     )
     => t_rfeature / nil
```

Description

Checks out the specified layout license, layout option license, or an appropriate higher-tier license and locks it after successful checkout.

Note: This function will be deprecated in a future release. Use eliCheckOutAndLock instead.

Arguments

t_feature	A string representing the license feature name that needs to be checked out.

Value Returned

t_rfeature	The license feature name of the checked-out license.
nil	License checkout is not successful.

Example

```
iclicCheckOutAndLock("Virtuoso_Layout_Suite_L")
=> "Virtuoso_Layout_Suite_XL"
```


© 2020 All Rights Reserved.

eliCheckOutAndLock

```
eliCheckOutAndLock(
     t_licenseName
     )
     => t / nil
```

Description

Checks out and locks the specified license feature. Once the specified license feature is locked, it cannot be checked in until the session ends. Use this function in a scripted batch process to reduce runtime due to licensing.

Arguments

t_licenseName	Name of the license feature to be checked out.

Value Returned

`nil`	If the license checkout failed.
`t`	If the license feature was checked out and locked successfully.

Example

A short script to check out and lock the `Virtuoso_Layout_Suite_L` license:

```
if( eliCheckOutAndLock( "Virtuoso_Layout_Suite_L" ) then
    launchScriptWork()
else
    info( "Cannot acquire Virtuoso_Layout_Suite_L license." )
)
```


© 2020 All Rights Reserved.

Design Editor Support Routines

This section describes routines available to support all applications. Those routines that alter Design Editor data structures (those with `Set` in their names) are intended for use inside the triggers and should not be used as standalone operations.

deGetCellView

```
deGetCellView(
     [ w_window [ x_level ] ]
     )
     => d_cellview / nil
```

Description

Returns the cellview for the item on the window stack corresponding to *x_level*. A value of `nil` is returned if the stack is empty, *x_level* is not a valid stack level in *w_window*, or *x_level* is -1. If *x_level* is defaulted or `nil` then the current item on the stack is used.

Arguments

w_window	Window that contains the cellview that is desired. Defaults to the current window.
x_level	Stack level to retrieve the cellview from.

Value Returned

d_cellview	The cellview at the requested stack level.
`nil`	The stack is empty, *x_level* is not a valid stack level in *w_window*, or *x_level* is -1.

Example

```
deGetCellView( window(3) 0 )
```

Equivalent to *window(3)->topCellView*.

© 2020 All Rights Reserved.

deGetEditCellView

```
deGetEditCellView(
     [ w_window ]
     )
     => d_cellview / nil
```

Description

Returns the cellview for the item on the window stack corresponding to the current edit level. `nil` is returned if the stack is empty.

This function is equivalent to

```
deGetCellView( w_window deGetEditLevel( w_window ))
```

but is more efficient.

Arguments

w_window	Window that contains the desired cellview. Defaults to the current window.

Value Returned

d_cellview	The cellview at the current edit stack level.
`nil`	The stack is empty.

Example

```
deGetEditCellView( window(3) )
```

Equivalent to *window(3)->editCellView*.

© 2020 All Rights Reserved.

deSetCellView

```
deSetCellView(
     [ w_window [ d_cellview [ x_level ] ] ]
     )
     => t / nil
```

Description

Replaces the cellview for the current item on the window stack. `nil` is returned if the stack is empty. This routine is intended for use by data triggers executing the `'swap` operation. It should never be called by an application outside of a data trigger.

Arguments

w_window	Window in which to set the new cellview. Defaults to the current window.
d_cellview	New cellview to be stored in the window stack. Defaults to `nil`.
x_level	Level of the window stack at which to store the new cellview. Defaults to the current edit level.

Value Returned

`t`	Stored the cellview at the given level.
`nil`	The stack is empty.

Example

```
deSetCellView( window(3) window(4)->editCellView 0)
```

Sets the top-level cellview in `window(3)` to be the edit cellview from `window(4)`.

© 2020 All Rights Reserved.

deGetViewType

```
deGetViewType(
     w_window
     [ x_level ]
     )
     => t_viewType / nil
```

Description

Returns the view type of the item at *x_level* in *w_window*. `nil` is returned if the stack is empty. If *x_level* is not specified, the current level is assumed.

Arguments

w_window	Window in which the view type is desired. Defaults to the current window.
x_level	Stack level to retrieve the view type from.

Value Returned

t_viewType	View type at the requested stack level.
`nil`	The stack is empty.

Example

```
deGetViewType( window(3) 2)
```

Returns "`graphic`," assuming data of this view type is at level 2 in `window(3)`.

© 2020 All Rights Reserved.

deGetEditViewType

```
deGetEditViewType(
     w_window
     )
     => t_viewType / nil
```

Description

Returns the *t_viewType* of the item at the current edit level in *w_window*.

This function is equivalent to

```
deGetViewType( w_window deGetEditLevel( w_window ))
```

but is more efficient.

Arguments

w_window	Window in which the view type is desired. Defaults to the current window.

Value Returned

t_viewType	View type at the current edit stack level.
nil	The stack is empty.

Example

```
deGetEditViewType( window(3))
```

Returns "graphic," assuming data of this view type at the current edit level in window(3).

© 2020 All Rights Reserved.

deGetRootType

```
deGetRootType(
     [ w_window [ x_level ] ]
     )
     => t_viewType / nil
```

Description

Returns the root class *t_viewType* of the item at *x_level* in *w_window*. `nil` is returned if the stack is empty. If *x_level* is not specified, the current level is assumed.

Arguments

w_window	Window in which the root view type is desired. Defaults to the current window.
x_level	Stack level to retrieve the root view type from.

Value Returned

t_viewType	Root view type at the requested stack level.
`nil`	The stack is empty.

Example

```
deGetRootType( window(3) 2)
```

Returns "`graphic`," assuming data of this view type at level 2 in `window(3)`.

© 2020 All Rights Reserved.

deGetEditRootType

```
deGetEditRootType(
     [ w_window ]
     )
     => t_viewType / nil
```

Description

Returns the root class view type of the item at the current edit level in *w_window*.

This function is equivalent to

```
deGetRootType( w_window deGetEditLevel( w_window ))
```

but is more efficient.

Arguments

w_window	Window in which the root view type is desired. Defaults to the current window.

Value Returned

t_viewType	Root view type at the current edit stack level.
nil	The stack is empty.

Example

```
deGetEditRootType( window(3) 2)
```

Returns "graphic," assuming data of this view type at the current edit level in window(3).

© 2020 All Rights Reserved.

deGetContext

```
deGetContext(
     [ w_window [ x_level ] ]
     )
     => g_context / nil
```

Description

Returns the context for the item on the window stack corresponding to *x_level*. A value of `nil` is returned if the stack is empty or *x_level* is `-1`. If *x_level* is not specified or is `nil`, the current item on the stack is used.

Arguments

w_window	Window in which the context is desired. Defaults to the current window.
x_level	Stack level to retrieve the context from.

Value Returned

g_context	Application context at the given level.
`nil`	The stack is empty or *x_level* is `-1`.

Example

```
deGetContext( window(3) 1)
```

Assuming `window(3)` contains a cellview at level 1, returns the graphics editor context at that level.

© 2020 All Rights Reserved.

deGetEditContext

```
deGetEditContext(
     [ w_window ]
     )
     => g_context / nil
```

Description

Returns the context for the item on the window stack corresponding to the current edit level.

This function is equivalent to

```
deGetContext( w_window deGetEditLevel( w_window ))
```

but is more efficient.

Arguments

w_window	Window in which the context is desired. Defaults to the current window.

Value Returned

g_context	Application context at the current edit level.
nil	The stack is empty.

Example

```
deGetEditContext( window(3))
```

Assuming `window(3)` contains a cellview at the current edit level, returns the graphics editor context at that level.

© 2020 All Rights Reserved.

deIsEditInContext

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

Description

Tells whether the current edit cellview in the window is in the context of a Hierarchy Manager configuration.

Arguments

w_windowId	The window in question. Defaults to the current window.

Value Returned

t	The edit is in context.
nil	Either the edit is out of context or there is no configuration on the window.

© 2020 All Rights Reserved.

deSetContext

```
deSetContext(
     [ w_window [ g_context [ x_level ] ] ]
     )
     => t / nil
```

Description

Replaces the context for the item at the specified level on the window stack. `nil` is returned if the stack is empty. This routine is intended for use by data triggers executing the `'swap` operation. It should never be called by application code outside of a data trigger.

Arguments

w_window	Window in which to set the new context. Defaults to the current window.
g_context	New context to be stored in the window stack. Defaults to `nil`.
x_level	Level of the window stack at which to store the new context. Defaults to the current edit level.

Value Returned

`t`	The context was successfully set.
`nil`	The context was not successfully set

Example

```
deSetContext( window(3) 1 myUserContext )
```

Sets the context for level 1 in `window(3)` to `myUserContext`.

© 2020 All Rights Reserved.

deGetMaxInContextLevel

```
deGetMaxInContextLevel(
     [ w_windowId ]
     )
     => n_level
```

Description

Gets the maximum level of the given window that is being edited in the context of the Hierarchy Manager configuration ID associated with the window. If the window has no configuration ID, 0 is returned.

Arguments

w_windowId	The window in question. Defaults to the current window.

Value Returned

n_level	The maximum level of the given window which is being edited in the context of the Hierarchy Manager configuration ID, or `0` if there is no configuration ID.

© 2020 All Rights Reserved.

deGetConfigId

```
deGetConfigId(
     [ w_windowId ]
     )
     => g_hdbConfigId / nil
```

Description

Gets the Hierarchy Manager configuration ID associated with the given window.

Arguments

w_windowId	The window in question. Defaults to the current window.

Value Returned

g_hdbConfigId	The configuration ID on the window.
nil	The window has no configuration ID.

© 2020 All Rights Reserved.

deSetConfigId

```
deSetConfigId(
     w_windowId
     g_hdbConfigId
     )
     => t / nil
```

Description

Sets the Hierarchy Manager configuration ID associated with the given window. Also creates a path vector and sets the window path vector. If *g_hdbConfigId* is `nil`, the window path vector is destroyed, and the window configuration ID is set to `nil`.

Arguments

w_windowId	The window in question. Defaults to the current window.
g_hdbConfigId	The configuration ID of the window.

Value Returned

`t`	The operation is successful.
`nil`	The operation was not successful

Examples

```
deSetConfigId( window(2) myConfigId)
```

Sets the configuration ID for window 2 and creates and sets the path vector.

```
deSetConfigId( window(2) nil)
```

Destroys the path vector and clears the configuration ID for window 2.

© 2020 All Rights Reserved.

deGetVector

```
deGetVector(
     [ w_windowId ]
     )
     => g_hdbPathVectorId / nil
```

Description

Gets the Hierarchy Manager path vector associated with the given window.

Arguments

w_windowId	The window in question. Defaults to the current window.

Value Returned

g_hdbPathVectorId	The vector on the window
nil	The window has no vector.

© 2020 All Rights Reserved.

deGetEIP

```
deGetEIP(
     [ w_window [ x_level ] ]
     )
     => t / nil
```

Description

Returns the edit-in-place state for the item on the window stack corresponding to *x_level*. The edit-in-place state is *t* if that stack level was created by the `deEditInPlace` function and `nil` otherwise. f `nil` is returned if the stack is empty or *x_level* is `-1`. If *x_level* is not specified or is `nil` then the current item on the stack is used.

Arguments

w_window	Window in which the edit-in-place state is desired. Defaults to the current window.
x_level	Stack level to retrieve the edit-in-place status from.

Value Returned

`t`	The given level was edited-in-place.
`nil`	The given level was not edited-in-place, the stack is empty, or *x_level* is `-1`.

Example

```
deGetEIP( window(3) 1)
```

Assuming an edit-in-place operation on an instance in the top-level cellview of `window(3)`, returns `t`.

© 2020 All Rights Reserved.

deGetPushed

```
deGetPushed(
     [ w_window [ x_level ] ]
     )
     => t / nil
```

Description

Returns the pushed state for the item on the window stack corresponding to *x_level*. The pushed state is `t` if that *x_level* is any level other than the current edit level and `nil` otherwise. `nil` is returned if the stack is empty or *x_level* is `-1`. If *x_level* is defaulted or is `nil` then the current item on the stack is used.

Arguments

w_window	Window in which the pushed state is desired. Defaults to the current window.
x_level	Stack level to retrieve the pushed status from.

Value Returned

`t`	The given level was pushed.
`nil`	The given level was the current edit level, the stack is empty, or *x_level* is `-1`.

Example

```
deGetPushed( window(3) 1)
```

Assuming three levels of hierarchy in `window(3)`, returns `t`.

© 2020 All Rights Reserved.

deGetProbeContext

```
deGetProbeContext(
     [ w_window ]
     )
     => g_probeContext / nil
```

Description

Returns the probe context for the window.

Arguments

w_window	Window from which to retrieve the probe context. Defaults to the current window.

Value Returned

g_probeContext	Probe context stored in the given window.
nil	There is no probe context.

Example

```
deGetProbeContext( window(3) )
```

Returns the probe context associated with `window(3)`.

© 2020 All Rights Reserved.

deSetProbeContext

```
deSetProbeContext(
     [ w_window [ g_probeContext ] ]
     )
     => t / nil
```

Description

Sets the probe context for the window. This is normally executed only by the probing implementation in the probe trigger.

Arguments

w_window	Window in which to set the probe context. Defaults to the current window.
g_probeContext	Probe context for the given window.

Value Returned

`t`	The probe context was successfully stored.
`nil`	The probe context was not successfully stored

Example

```
deSetProbeContext( window(3) myProbeContext )
```

Sets the probe context for `window(3)` to `myProbeContext`.

© 2020 All Rights Reserved.

deGetEditLevel

```
deGetEditLevel(
     [ w_window ]
     )
     => x_level / nil
```

Description

Returns the current edit level of *w_window*. If the stack is empty, the routine returns `-1`. The current edit level is the level of the stack which is currently being displayed. This is normally the maximum edit level, unless `deGoToLevel` was executed.

Arguments

w_window	Window in which current edit level is desired. Defaults to the current window.

Value Returned

x_level	Current edit level of *w_window*.
`nil`	The function failed.

Example

```
deGetEditLevel( window(3) )
```

Returns the current edit level of `window(3)`.

© 2020 All Rights Reserved.

deGetMaxEditLevel

```
deGetMaxEditLevel(
     [ w_window ]
     )
     => x_level / nil
```

Description

Returns the highest edit level of *w_window*. If the stack is empty, returns `-1`.

Arguments

w_window	Window in which maximum edit level is desired. Defaults to the current window.

Value Returned

x_level	Maximum edit level of *w_window*.
`nil`	The function failed.

Example

```
deGetMaxEditLevel( window(3) )
```

Returns the highest level of `window(3)`.

© 2020 All Rights Reserved.

deEnableApp

```
deEnableApp(
     w_window
     t_appName
     )
     => t / nil
```

Description

Enables the application registered with *t_appName* in *w_window*. It affects only the customizer menu in *w_window*. It is available to allow certain applications to be enabled depending on run-time state associated with a particular design.

Arguments

w_window	Window in which the specified application is to be enabled. Defaults to the current window.
t_appName	Name of the application to be enabled.

Value Returned

`t`	The application was successfully enabled.
`nil`	The application was not successfully enabled

Example

```
deEnableApp( window(3) "Timing Analysis" )
```

Enables the application "`Timing Analysis`" in `window(3)`.

© 2020 All Rights Reserved.

deDisableApp

```
deDisableApp(
     w_window
     t_appName
     )
     => t / nil
```

Description

Enables the application registered with *t_appName* in *w_window*. It affects only the customizer menu in *w_window*. It is available to allow certain applications to be disabled depending on run-time state associated with a particular design.

Arguments

w_window	Window in which the specified application is to be disabled. Defaults to the current window.
t_appName	Name of the application to be enabled.

Value Returned

`t`	The application was successfully disabled.
`nil`	The application was not disabled.

Example

```
deDisableApp( window(3) "Timing Analysis" )
```

Disables the application "`Timing Analysis`" in `window(3)`.

© 2020 All Rights Reserved.

deInstallApp

```
deInstallApp(
     w_window
     t_appName
     )
     => t / nil
```

Description

Switches the current application in *w_window* to *t_appName*. The enable trigger is called before switching, and if it returns `nil`, the switch is aborted. Otherwise, the application menus and bindkeys are installed as if the user had executed the corresponding customizer item.

The enable trigger is called with the standard trigger argument, unlike other situations in which the trigger is called. The trigger argument reflects the application which corresponds to *t_appName*, not the one currently installed in the window.

Arguments

w_window	Window in which the specified application is to be installed as the current application. Defaults to the current window.
t_appName	Name of the application to be installed.

Value Returned

`t`	The application was successfully installed.
`nil`	The application was not successfully installed.

Example

```
deInstallApp( window(3) "Timing Analysis" )
```

Switches to the application "`Timing Analysis`" in `window(3)`.

© 2020 All Rights Reserved.

deGetAllApplicationNames

```
deGetAllApplicationNames(
     t_rootViewType
     )
     => l_app / nil
```

Description

Returns a list of all applications that can be associated with the given root view type.

Arguments

t_rootViewType	Root view type.

Value Returned

l_app	List of all applications associated with the root view type.
nil	No application name found for the given root view type.

Example

(IC6.1.8 Only) The following example **r**returns a list of all applications that can be associated with the root view type `maskLayout`.

```
deGetAllApplicationNames("maskLayout")
=>
("ADE L" "ADE XL" "ADE GXL" "Dracula Interactive" """Layout L" "Layout XL" "Layout
GXL" "Layout CE" "Other" "Parasitics"   "Pcell" "Spectre" "UltraSim" "Verilog-XL"
"Voltage Storm")
```

(ICADVM20.1 Only) The following example returns a list of all applications that can be associated with the root view type `maskLayout`.

```
deGetAllApplicationNames("maskLayout")
=>
("ADE L" "ADE XL" "ADE GXL" "Dracula Interactive" "Layout Viewer""Layout XL"
"Layout EXL" "Layout CE" "Other" "Parasitics"   "Pcell" "Spectre" "UltraSim"
"Verilog-XL" "Voltage Storm")
```


© 2020 All Rights Reserved.

deGetAllRootTypes

```
deGetAllRootTypes(
     )
     => l_rootTypes / nil
```

Description

Returns a list of all currently registered root view types.

Arguments

None

Value Returned

l_rootTypes	List of all registered root view types.
nil	The function failed.

Example

```
deGetAllRootTypes()
```

Returns (“graphic” “maskLayout” “schematic”).

Note: “TimingAnalysis” is not in the list because it is a subtype of “schematic”.

© 2020 All Rights Reserved.

deGetAllViewTypes

```
deGetAllViewTypes(
     )
     => l_viewTypes / nil
```

Description

Returns a list of all fully registered view types and view sub types. Any view sub type for which some parent has not been registered will not appear in this list.

Arguments

None

Value Returned

l_viewTypes	List of all fully registered view types.
nil	The function failed.

Example

```
deGetAllViewTypes()
```

Returns (“graphic” “maskLayout” “schematic” “TimingAnalysis”).

© 2020 All Rights Reserved.

deGetRootViewTypeList

```
deGetRootViewTypeList(
     )
     => l_rootViewTypes / nil
```

Description

Returns a list of root view types, which have an associated tool as specified in the tool registry.

Arguments

None

Value Returned

l_rootViewTypes	List of root view types, which have an associated tool as specified in the tool registry.
nil	No such root view type is found.

Example

```
deGetRootViewTypeList()
=>
("adexl" "Cdl" "schematic" "config" "Hspice" "modelwriter" "Spectre" "Spice" "text"
"Vhdl" "VHDLAMSText" "physConfig" "text.v" "text.veriloga" "VerilogAMSText"
"maskLayout" "schematic" "schematicSymbol")
```

To generate a list of view types that you can choose from the New File dialog box, use:

```
(mapcar 'deMapViewTypeToUserType (deGetRootViewTypeList))
```


© 2020 All Rights Reserved.

deMapUserTypeToViewType

```
deMapUserTypeToViewType(
     t_userType
     )
     => t_viewType
```

Description

Maps a user type to the *dfII_viewType*.

If the given user type is not found in the mapping, the function returns the user type itself.

Arguments

t_userType	User type.

Value Returned

t_viewType	DFII view type.

Example

```
deMapUserTypeToViewType("layout")
=>
("masklayout")
```


© 2020 All Rights Reserved.

deMapViewTypeToUserType

```
deMapViewTypeToUserType(
     g_viewType
     )
     => t_userType / nil
```

Description

Maps the *dfII_viewType* to the user type.

If the given view type is not found in the mapping, the function returns the view type itself. For example, the function will return `nil` if the argument given is `nil`, because `nil` is not in the mapping.

Arguments

g_viewType	DFII view type.

Value Returned

`t_userType`	User type.
`nil`	Returns `nil` when the argument given is `nil`.

Example

```
deMapViewTypeToUserType("maskLayout")
=>
("layout")
deMapViewTypeToUserType ("unknownViewType")
=> "unknownViewType"
deMapViewTypeToUserType (nil)
=> nil
```


© 2020 All Rights Reserved.

deViewTypeIsRegistered

```
deViewTypeIsRegistered(
     t_viewType
     )
     => t / nil
```

Description

Returns a non-nil value if *t_viewType* is fully registered; that is, all ancestors up to and including the root type are currently registered.

Arguments

t_viewType	View type which might or might not be registered.

Value Returned

t	Returns a non-nil value only when the *t_viewType* is fully registered.
nil	The view type is not registered

Examples

The following example returns a list of registered views because view type `maskLayoutXL` is fully registered.

```
deViewTypeIsRegistered( "maskLayoutXL")
=> "maskLayoutXL" "maskLayoutEXL" "analogArtist-layout"
```

The following example returns `nil` because `unknown_view_type` is not registered.

```
deViewTypeIsRegistered( "unknown_view_type")
=> nil
```


© 2020 All Rights Reserved.

deGetLowerTierApp

```
deGetLowerTierApp(
     t_appName
     [ x_lowerBy ]
     )
     => l_lowertierAppNames
```

Description

Returns a list of application names, which are *x_lowerBy* levels lower in tier in the same family of given *t_appName*. The default value for *x_lowerBy* is one.

Arguments

t_appName	Specifies the name of the application name for which you want to find out the lower tier applications
x_lowerBy	Specifies the number of lower levels that you want to find out. The default is one.

Value Returned

l_lowertierAppNames	List of application names, which are *x_lowerBy* levels lower in tier in the same family of given *t_appName*

Example

The following examples are IC6.1.8 Only:

```
(deGetLowerTierApp "Symbol XL") => ("Schematics" "Symbol")
(deGetLowerTierApp "Schematics XL") =>  ("Schematics" "Symbol")
(deGetLowerTierApp "Schematics" -1) => ("Schematics XL" "Symbol XL")

(deGetLowerTierApp "VLS-GXL") => ("Virtuoso XL")
(deGetLowerTierApp "Virtuoso XL") => ("Layout")
(deGetLowerTierApp "VLS-GXL" 2) => ("Layout")
(deGetLowerTierApp "Layout" -2) => ("VLS-GXL")
```

The following examples are ICADVM20.1 Only:

```
(deGetLowerTierApp "VLS-EXL") => ("Virtuoso XL")
(deGetLowerTierApp "Virtuoso XL") => ("Layout")
```


© 2020 All Rights Reserved.

```
(deGetLowerTierApp "VLS-EXL" 2) => ("Layout")
```


© 2020 All Rights Reserved.

Extended Design Editor Example

This section contains an extended example of how the various Design Editor routines are used. Included is an example implementation of a typical data trigger, `exDataTrigger`, and application trigger, `exAppTrigger`. These procedures are included as a guide for developing your own data and application triggers. However, the installation of an application into a window requires C-code access to provide the implementation of the window interface. The standard triggers for applications, `geDataTrigger` and `geAppTrigger`, have been withdrawn from public use. See your Cadence representative if this is a problem.

The first example routine is a data trigger. This routine uses the information in the argument list to open up a data file, and should return the database ID of the result. It is also used to close or change the data file associated with a window.

The second example routine is an application trigger. This routine expects the data trigger to have returned successfully and has the job of managing the data while it is in the window. It must be able to install the data for the first time, save it while another file is stacked on top of it, restore it when that file is removed, remove itself, and copy itself. Sample menu and enable triggers are given showing you how to create menus for your application and to control its availability.

Normally you will not implement these routines yourself. They will be defined as part of a Cadence application, and you will use the user triggers to do your customization. The final example shows how you may customize an existing application with some sample user triggers.

If your application integration needs require special data and/or application triggers, see your Cadence representative for help in developing these.

```
;*****************************************************************
; exDataTrigger(args)
;
; FUNCTION
;   This is an example data trigger. It opens the appropriate
;   cellView as specified by the input arguments.
;*****************************************************************

procedure(exDataTrigger(args)
    prog((s cellViewId mode viewType)
            caseq(args->action
                 (open ;open the data file
                     s = args->viewNameList || args->viewName

                    if(args->accessMode != "w" then
                         if(args->accessMode != "r"  then
                             mode = strcat(args->accessMode "n")
                             viewType = args->rootType
                         else
                             mode = args->accessMode
                             viewType = ""
```


© 2020 All Rights Reserved.

```
                )
                cellViewId = dbOpenCellViewByType(args->
                        libName
                    args->cellName s viewType mode)
                unless(cellViewId
                    when(args->accessMode == "a"
                        ; the cellview could not be opened
                                for edit.
                        ; Open it for read.
                         cellViewId = dbOpenCellViewByType(args->libName
                            args->cellName s viewType "r")
                    )
                )
            )
        else
             cellViewId = dbOpenCellViewByType(args->libName
             args->cellName s args->rootType "r")
             if(cellViewId then
                   dbReopen(cellViewId "w")
            )
             else
                getWarn()
                cellViewId = dbOpenCellViewByType(args->libName
                args->cellName s args->rootType "wn")
                /* if it's a new cellview, do a save to force it
                to create an empty file, for discard edits */
                when(cellViewId
                    dbSave(cellViewId)
                )
             )
         )
         return(cellViewId)
         )
         (close
             cellViewId = deGetCellView(args->window)
             when(cellViewId~>superMaster
                 return(t)
             )
             return(dbClose(cellViewId))
         )
         (swap
             s = args->viewNameList || args->viewName

             if(args->accessMode != "w" then
                 if(args->accessMode == "r"  then
                     mode = args->accessMode
                 else
                     mode = strcat(args->accessMode "n")
                 )

                 cellViewId = dbOpenCellViewByType(args->libName
                 args->cellName s args->rootType mode)
        else
             cellViewId = dbOpenCellViewByType(args->libName
             args->cellName s args->rootType "r")
             if(cellViewId then
                 dbReopen(cellViewId "w")
             else
                 getWarn()
                 cellViewId = dbOpenCellViewByType(args->libName
```


© 2020 All Rights Reserved.

```
                        args->cellName s args->rootType "wn")
                )
            )
            return(cellViewId)
            )
        )
        return()
    )
)

; Here we remove the first
; standard menu and add a window property
procedure( exCustomPostInstallTrigger( args )
    ; remove one of the default menus
    hiDeleteBannerMenu(args->window 0)
    ; set a special window property
    window->pathWidth = 10.0
    )


; Now, register the example user triggers

deRegUserTriggers( "example" 'exCustomAppTrigger 'exCustomMenuTrigger
'exCustomPostInstallTrigger )

; finally, we list the app info for "example" and see that the
; custom triggers have been added to the list.
deGetAppInfo( "example" )
( nil viewType "example" parentType "" appName "Example Application" help "Example
Application Help" widgetType "graphics" dataTrigger exDataTrigger appTrigger
exAppTrigger enableTrigger exEnableTrigger menuTrigger exMenuTrigger
userAppTrigger exCustomAppTrigger userMenuTrigger exCustomMenuTrigger
postInstallTrigger exCustomPostInstallTrigger inheritMenus nil inheritEnabling nil
subtypes nil )


; This is a basic application trigger that can serve many
; applications

procedure( exAppTrigger(args)
    prog( (w ctx cv)
    w = args->window
    cv = deGetCellView(w)
    caseq( args->action
        (install

            ctx = <create your application context data structure
                 install your application context data structure in
                 the window, and initialize the window interface to
                 contain drawing and other procedures>

            return( ctx )
            )
        (push
            ctx = deGetContext( w )

        < update ctx to reflect the fact that this stack element is
             no longer current.>

             return( ctx )
             )
        (pop
             ctx = deGetContext( w )

             <update ctx to reflect the fact that this stack element
will now be current again>
```


© 2020 All Rights Reserved.

```
            < install your application context data structure in the
                window, and initialize the window interface to
                contain the proper drawing and other procedures.>
            return( ctx )
            )
        (copy
            ctx = <make a copy of your application data structure>
            return( ctx )
            )
        (close
            ctx = deGetContext( w )
            ctx = <result of attempting to destroy the current
                application data context >
            return( ctx )
            )
        (swap
            ctx = deGetContext( w )
            ctx = <re-initialize application data structure with a
                new data file)
            return( ctx )
            )
       (copyInstall
            ctx = deGetContext( w )
            <install your application context data structure in the
            window, and initialize the window interface to contain
            the proper drawing and other procedures>
            return( ctx )
            )
       (t return(t))
       )
    return(t)
    ))

; define the default menus
exMenu1 = hiCreateMenu(...)
exMenu2 = hiCreateMenu(...)
procedure( exMenuTrigger(args)
    ; return a list of menus
    list( exMenu1 exMenu2 )
    )
; define the conditions for allowing the user to use your application
procedure( exEnableTrigger(@optional args)
    if( !args then
        <check root data security>
    else
        <check example application security> )
    )
; Now, register the example application
deRegApp( ?viewType "example" ?appName "Example Application" ?help "Example
Application Help" ?dataTrigger 'exDataTrigger ?appTrigger 'exAppTrigger
?menuTrigger 'exMenuTrigger ?enableTrigger 'exEnableTrigger )
deGetAppInfo( "example" )
```


© 2020 All Rights Reserved.

```
( nil viewType "example" parentType "" appName "Example Application" help "Example
Application Help" widgetType "graphics" dataTrigger exDataTrigger appTrigger
exAppTrigger enableTrigger exEnableTrigger menuTrigger exMenuTrigger
userAppTrigger nil userMenuTrigger nil postInstallTrigger nil inheritMenus nil
inheritEnabling nil subtypes nil )
;*************************************************************************
; Here we show how to customize an existing application using user triggers.
;*************************************************************************
; Custom app trigger.
; This is called immediately after (or before, as determined by the ; situation)the
standard app trigger. It installs a custom popup menu ; and selection form
procedure( exCustomAppTrigger( args )
    caseq( args->action
        ((install pop copyInstall)
             < create your own selection form >
             < install your own popup menu >
             )
        ((push close)
             < remove your own selection form in case the
                 new application doesn't have one of its own >
             < remove your own popup menu - same reason >
             )
         ; in all other cases, we don't do anything special
        )
    t
    )
; the user menu trigger will add a menu to the banner
exCustomMenu = hiCreateMenu(...)
procedure( exCustomMenuTrigger( args )
    list( exCustomMenu )
    )
; the user post install trigger can customize the application
; in any way desired. Here we remove the first
; standard menu and add a window property
procedure( exCustomPostInstallTrigger( args )
    ; remove one of the default menus
    hiDeleteBannerMenu(args->window 0)
    ; set a special window property
    window->pathWidth = 10.0
    )
; Now, register the example user triggers
deRegUserTriggers( "example" 'exCustomAppTrigger 'exCustomMenuTrigger
'exCustomPostInstallTrigger )
; finally, we list the app info for "example" and see that the
; custom triggers have been added to the list.
deGetAppInfo( "example" )
( nil viewType "example" parentType "" appName "Example Application" help "Example
Application Help" widgetType "graphics" dataTrigger exDataTrigger appTrigger
exAppTrigger enableTrigger exEnableTrigger menuTrigger exMenuTrigger
userAppTrigger exCustomAppTrigger userMenuTrigger exCustomMenuTrigger
postInstallTrigger exCustomPostInstallTrigger inheritMenus nil inheritEnabling nil
subtypes nil )
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

A

Unsupported Functions

The functions listed below are unsupported and are retained only to maintain backward compatibility with previous releases:

- `dbConnectObject`
- `dbDisconnectObject`
- `dbIsObjectConnected`
- `dbObjectHasConnection`
- `dbCompactCellView`

These functions should not be used in any new scripts or code and should be removed from existing scripts and code where possible.

© 2020 All Rights Reserved.