

Virtuoso Studio Design Environment SKILL Reference

**Product Version IC23.1
November 2023**

© 2023 Cadence Design Systems, Inc.
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.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

1

Graphics Editor Functions	61
Design Functions	63
geChangeCellView	64
geChangeEditMode	66
geDiscardEdits	67
geNewWindow	68
geReturn	69
geReturnToLevel	70
geSave	71
geSaveAs	72
geSaveHier	74
Automatic Menu Enabling Functions	75
Defining a Menu Update Policy	76
geGetDisabledMenus	77
geGetEnablePredicate	79
geGetWindowStatus	80
geSetEnablePredicate	81
geSetWindowStatus	83
geUpdateWindowMenus	85
Cross Section Viewer Functions	87
geCrossSection	88
geCrossSectionAddLayerThickness	90
geCrossSectionAddLayerToSkip	92
geCrossSectionAddLayersToMerge	94
geCrossSectionAddMaterialToSkip	96
geCrossSectionClearConfig	98
geCrossSectionLoadConfig	99
Edit and Display Functions	101
geClearIgnoreProp	102
gec	103
geClearNetNameDisplayFilter	104

Virtuoso Studio Design Environment SKILL Reference

<u>geComputeQueryBbox</u>	105
<u>geCopySelSet</u>	106
<u>geDeleteSelSet</u>	108
<u>geDispWinEnvProps</u>	109
<u>geEditSelSet</u>	111
<u>geEditSingleObject</u>	112
<u>geEditToWindowPoint</u>	113
<u>geEnableNetNameDisplay</u>	115
<u>geEquivWindow</u>	117
<u>geEquivWindowSet</u>	118
<u>geFindPathToCellViewInst</u>	119
<u>geGetCellViewWindow</u>	120
<u>geGetEditCellView</u>	121
<u>geGetEditCellViewWindow</u>	122
<u>geGetEditRep</u>	123
<u>geGetHierMemInst</u>	124
<u>geGetInstHier</u>	126
<u>geGetInstHierPath</u>	128
<u>geGetNetNameDisplayFilter</u>	130
<u>geGetObjectSelectedSet</u>	131
<u>geGetSortedSelectByLine</u>	133
<u>geGetTopCellViewWindow</u>	134
<u>geGetTopLevelCellView</u>	135
<u>geGetWindowBox</u>	136
<u>geGetWindowCellView</u>	137
<u>geGetWindowRep</u>	139
<u>gelsGraphicalWindow</u>	140
<u>gelsNetNameDisplayActiveOnWindow</u>	141
<u>gelsNetNameDisplayEnabled</u>	142
<u>geMenuForObjectsUnderPoint</u>	143
<u>geNetNameDisplayOptionForm</u>	144
<u>geSaveNetNameDisplayFilter</u>	145
<u>geSelectNextInSelSet</u>	146
<u>geSelectPreviousInSelSet</u>	147
<u>geSelectToggleObject</u>	148
<u>geSetNetNameDisplayFilter</u>	149

Virtuoso Studio Design Environment SKILL Reference

<u>geSetTCProbeDisplayStyle</u>	150
<u>geTransparencyOptions</u>	151
<u>geWindowToEditPoint</u>	152
<u>gtc</u>	154
<u>gwc</u>	155
<u>Refresh Functions</u>	156
<u>geRefresh</u>	157
<u>geRefreshCellView</u>	158
<u>geRefreshCellViews</u>	159
<u>geRefreshWindow</u>	160
<u>geRefreshWorldView</u>	161
<u>View-Level Functions</u>	162
<u>geDeleteAllAreaViewLevel</u>	163
<u>geDeleteAllInstViewLevel</u>	164
<u>geDeleteAreaViewLevel</u>	165
<u>geDeleteInstViewLevel</u>	166
<u>geGetAllAreaViewLevel</u>	167
<u>geGetAllInstViewLevel</u>	168
<u>geGetAreaViewLevel</u>	169
<u>geGetInstViewLevel</u>	170
<u>geSetAreaViewLevel</u>	171
<u>geSetInstViewLevel</u>	173
<u>geRestoreWinView</u>	175
<u>geSaveWinView</u>	176
<u>Procedural-Level Functions</u>	177
<u>geEditInPlace</u>	178
<u>gePush</u>	180
<u>geSwitch</u>	182
<u>geSwitchInContext</u>	184
<u>Window Scroll Functions</u>	186
<u>geCenter</u>	187
<u>geGetScrollPercent</u>	188
<u>geScroll</u>	189

Virtuoso Studio Design Environment SKILL Reference

<u>geSetScrollPercent</u>	191
<u>geShift</u>	192
<u>geZoomToGrid</u>	193
<u>Selection Functions</u>	194
<u>Point Selection</u>	196
<u>Area Selection</u>	197
<u>Selection User Interface</u>	199
<u>geAdvWireSelection</u>	200
<u>geAddSelectBox</u>	201
<u>geAddSelectPoint</u>	203
<u>geAdjustPoint</u>	205
<u>geCyclePotentialSelectNext</u>	206
<u>geCycleSelectNext</u>	207
<u>geDeselectAll</u>	208
<u>geDeselectAllFig</u>	209
<u>geDeselectAllObject</u>	210
<u>geDeselectArea</u>	211
<u>geDeselectFig</u>	213
<u>geDeselectFigs</u>	214
<u>geDeselectFigNoFilter</u>	215
<u>geDeselectFigPoint</u>	216
<u>geDeselectObject</u>	218
<u>geDeselectObjectNoFilter</u>	219
<u>geDeselectObjectPoint</u>	220
<u>geDeselectPoint</u>	222
<u>geDeselectProtected</u>	224
<u>geDSAGetSelectedSet</u>	225
<u>geDSAFreeze</u>	227
<u>geGetPotentialSelection</u>	228
<u>geGetSelSet</u>	230
<u>geGetSelectedSet</u>	231
<u>geGetSelSetCount</u>	232
<u>geGetSelectedInstancesCount</u>	233
<u>geGetSelProtectedObjects</u>	234
<u>geGetSelSetFigPoint</u>	235
<u>geGetWindowSelProtectionHighlight</u>	236

Virtuoso Studio Design Environment SKILL Reference

<u>geGetWindowSelProtectionSelectability</u>	237
<u>geHiDragFig</u>	238
<u>geHiDisplaySelectionForm</u>	242
<u>geHighlightProtected</u>	243
<u>geHiSetSelProtectionEF</u>	244
<u>geHiUnsetSelProtectionEF</u>	245
<u>gelsFigAllSelected</u>	246
<u>gelsFigPartiallySelected</u>	247
<u>gelsFigSelected</u>	248
<u>gelsObjectAllSelected</u>	249
<u>gelsObjectPartiallySelected</u>	250
<u>gelsObjectSelected</u>	251
<u>gelsPointInsidePolygon</u>	252
<u>gelsSelProtection</u>	254
<u>gePointQuery</u>	255
<u>gePointToFig</u>	257
<u>geSelectAll</u>	259
<u>geSelectAllFig</u>	260
<u>geSelectArea</u>	261
<u>geSelectByLine</u>	263
<u>geSelectBy2PointsLine</u>	264
<u>geSelectFig</u>	265
<u>geSelectFigs</u>	266
<u>geSelectFigNoFilter</u>	267
<u>geSelectFigPoint</u>	268
<u>geSelectObject</u>	270
<u>geSelectObjectNoFilter</u>	271
<u>geSelectPoint</u>	272
<u>geSelectProtected</u>	274
<u>geSelectViaPile</u>	275
<u>geSetEnterFunctionSelectionMode</u>	276
<u>geSelObjectsPartiallySelected</u>	278
<u>geSetSelProtection</u>	279

Virtuoso Studio Design Environment SKILL Reference

<u>geSetSelProtectionAllObjects</u>	281
<u>geSetSelProtectionHighlightOptionForm</u>	282
<u>geSetSelProtectionHighlightOptions</u>	283
<u>geSingleSelectBox</u>	284
<u>geSingleSelectPoint</u>	286
<u>geSubSelectBox</u>	288
<u>geSubSelectPoint</u>	290
<u>geToggleAreaSelectOption</u>	292
<u>geToggleDisplayResolution</u>	293
<u>geTogglePartialSelect</u>	294
<u>geToggleWindowSelProtectionHighlight</u>	295
<u>geToggleWindowSelProtectionSelectability</u>	296
<u>geUnhighlightProtected</u>	297
<u>geUnsetAllSelProtection</u>	298
<u>geUnsetSelProtection</u>	299
<u>Stored Selection Functions</u>	301
<u>geDeleteStoredSelectionSet</u>	302
<u>geDeselectStoredSelectionSet</u>	303
<u>geGetStoredSelectionSets</u>	304
<u>geHiStoreSelectionForm</u>	305
<u>gelsAutoStoreSelectionOn</u>	306
<u>geSelectNextAutoStoreSet</u>	307
<u>geSelectPreviousAutoStoreSet</u>	308
<u>geSelectStoredSelectionSet</u>	309
<u>geSetAutoStoreSelection</u>	310
<u>geStoreSelectionSet</u>	311
<u>Highlight Functions</u>	312
<u>Highlighting Shapes in a Cellview</u>	312
<u>geAddHilightArc</u>	314
<u>geAddHilightBlockage</u>	315
<u>geAddHilightCircle</u>	316
<u>geAddHilightCurvedPath</u>	317
<u>geAddHilightCurvedPolygon</u>	318
<u>geAddHilightDistance</u>	319
<u>geAddHilightDonut</u>	320
<u>geAddHilightDot</u>	321

Virtuoso Studio Design Environment SKILL Reference

<u>geAddHilighEllipse</u>	322
<u>geAddHilighFig</u>	323
<u>geAddHilighLabel</u>	326
<u>geAddHilighLine</u>	328
<u>geAddHilighPath</u>	329
<u>geAddHilighPathSeg</u>	330
<u>geAddHilighPolygon</u>	332
<u>geAddHilighRow</u>	333
<u>geAddHilighRuler</u>	334
<u>geAddHilighSlicedCircle</u>	335
<u>geAddHilighSlicedDonut</u>	336
<u>geAddHilighTrackPattern</u>	337
<u>geAddHilighRectangle</u>	338
<u>geChangeHilighPacketName</u>	339
<u>geCreateHilighGroup</u>	340
<u>geCreateHilighSet</u>	341
<u>geCreateWindowHilighSet</u>	343
<u>geDeleteAllHilighSet</u>	344
<u>geDeleteAllWindowHilighSet</u>	345
<u>geDeleteHilighGroup</u>	346
<u>geDeleteHilighObject</u>	347
<u>geDeleteHilighSet</u>	348
<u>geDeleteHilighSetInWindows</u>	349
<u>geDrawHilighSet</u>	350
<u>geGetCurrentHilighSet</u>	351
<u>geGetCurrentWindowHilighSet</u>	352
<u>geGetFigHilighObjectId</u>	353
<u>geGetHilighObjectFigId</u>	354
<u>geInsertHilighObjectsInGroup</u>	355
<u>gelsValidHilighGroup</u>	356
<u>gelsValidHilighId</u>	357
<u>gelsValidHilighObject</u>	358
<u>gelsValidHilighSet</u>	360

Virtuoso Studio Design Environment SKILL Reference

<u>gePopHighlightStack</u>	362
<u>gePopWindowHighlightStack</u>	363
<u>gePushHighlightStack</u>	364
<u>geResetHighlightSetAlphaBlendingParameters</u>	365
<u>geResetHighlightSetHaloParameters</u>	366
<u>geSetHighlightSetAlphaBlendingParameters</u>	367
<u>geSetHighlightSetHaloParameters</u>	369
<u>geShareWindowHighlightSet</u>	371
<u>Background Cellview Functions</u>	372
<u>geBackgroundCellView</u>	373
<u>geDisableBackgroundCellView</u>	374
<u>geEnableBackgroundCellView</u>	375
<u>geGetBackgroundCellView</u>	376
<u>geResetBackgroundCellView</u>	377
<u>geSetBackgroundCellView</u>	378
<u>geZoomToBackgroundAndEditCV</u>	380
<u>geZoomToBackgroundCV</u>	381
<u>Marker Functions</u>	382
<u>geCreateMarker</u>	383
<u>geCreateMarkerByBBox</u>	385
<u>geCreateMarkerByPoints</u>	387
<u>geDeleteAllMarker</u>	389
<u>geGetMarkerDisplayText</u>	391
<u>geGetMarkerOwner</u>	392
<u>geGetMarkerReason</u>	393
<u>geGetMarkerType</u>	394
<u>geHasMarker</u>	395
<u>geHiCommonFindMarker</u>	396
<u>geHiDeleteAllMarker</u>	397
<u>geHiDeleteMarker</u>	398
<u>geHiExplainMarker</u>	399
<u>geHiFindMarker</u>	400
<u>gelsMarkerShape</u>	401
<u>Probe Functions</u>	402
<u>Probe Types</u>	404
<u>Probe Object</u>	406

Virtuoso Studio Design Environment SKILL Reference

<u>Probe Function SKILL Interface</u>	409
<u>Probe File Format</u>	409
<u>geAddInstProbe</u>	412
<u>geAddNetProbe</u>	414
<u>geAddNetProbeFromTo</u>	416
<u>geAddPathProbe</u>	418
<u>geAddTermProbe</u>	420
<u>geClearProbeNetFilter</u>	422
<u>geDeleteAllProbe</u>	423
<u>geDeleteInstProbe</u>	424
<u>geDeleteNetProbe</u>	425
<u>geDeletePathProbe</u>	426
<u>geDeleteProbe</u>	427
<u>geDeleteTermProbe</u>	428
<u>geEnterAddInstProbe</u>	429
<u>geEnterAddNetProbe</u>	431
<u>geEnterAddTermProbe</u>	433
<u>geEnterDeleteInstProbe</u>	435
<u>geEnterDeleteNetProbe</u>	437
<u>geEnterDeleteTermProbe</u>	438
<u>geExplainProbe</u>	440
<u>geGetAdjustedPath</u>	441
<u>geGetAllProbe</u>	443
<u>geGetCrossProbeValue</u>	444
<u>geGetProbeNetFilter</u>	445
<u>geGetProbeNetStopLevel</u>	447
<u>geGetProbeUserLPP</u>	448
<u>geiLoadProbe</u>	449
<u>geiSaveProbe</u>	450
<u>geLoadNextProbe</u>	451
<u>geLoadProbe</u>	453
<u>geMakeProbeWithColor</u>	455
<u>geMakeProbeWithLabel</u>	458

Virtuoso Studio Design Environment SKILL Reference

<u>geMakeProbeWithOnColor</u>	462
<u>geMakeProbeWithoutColor</u>	465
<u>gePrintAllProbe</u>	467
<u>geProbeP</u>	468
<u>geRegCrossProbeTrigger</u>	469
<u>geSaveProbe</u>	470
<u>geSaveProbeNetFilter</u>	471
<u>geSetCrossProbeValue</u>	472
<u>geSetLimitedProbe</u>	473
<u>geSetMaxCreatedProbe</u>	475
<u>geSetProbeDirection</u>	476
<u>geSetProbeNetFilter</u>	478
<u>geSetProbeNetStopLevel</u>	479
<u>geSetProbeUserLPP</u>	480
<u>geUnRegCrossProbeTrigger</u>	481
<u>Icon Creation and Loading Functions</u>	482
<u>geCellViewToDlist</u>	483
<u>geLoadIcon</u>	484
<u>geSaveIcon</u>	486
<u>Utility Functions</u>	488
<u>geAskOverwriteMode</u>	490
<u>geClosePolygon</u>	491
<u>geCompressPointArray</u>	492
<u>geCoordp</u>	493
<u>geCreatePacket</u>	494
<u>geGetInstTransform</u>	496
<u>geGetLockInfo</u>	498
<u>geInstIdToFileSpec</u>	499
<u>gelsPathReentrant</u>	501
<u>gelsPolygonReentrant</u>	503
<u>geOpen</u>	504
<u>geQuerySelSet</u>	506
<u>geRegSaveAllTrigger</u>	507
<u>geResetCellView</u>	508
<u>geTransformUserBBox</u>	509
<u>geTransformUserPoint</u>	510

Virtuoso Studio Design Environment SKILL Reference

<u>geValidateWindow</u>	512
<u>geWriteCellView</u>	513
<u>Microwave Object Functions</u>	515
<u>geCreateBend</u>	516
<u>geCreateTaper</u>	518
<u>geCreateTrl</u>	520
<u>Mouse Functions</u>	522
<u>mouseAddPt</u>	523
<u>mouseAddSelectPt</u>	524
<u>mouseFinishPt</u>	525
<u>mousePopUp</u>	526
<u>mouseSingleSelectPt</u>	527
<u>mouseSubSelectPt</u>	528
<u>mouseUndoPt</u>	529

2

<u>Database Access Functions</u>	533
<u>Data Stored as Objects</u>	535
<u>dbGetDatabaseType</u>	536
<u>Object Identifier (dbObject)</u>	537
<u>Object Types and Classes</u>	537
<u>Relationships Among Objects</u>	539
<u>Database Access Function Argument Types</u>	540
<u>dbIsId</u>	544
<u>dbValidP</u>	545
<u>dbobjectp</u>	546
<u>dbDeleteObject</u>	547
<u>Inherited Connections Functions</u>	549
<u>dbCreateSigNetExpr</u>	551
<u>dbCreateTermNetExpr</u>	552
<u>dbCVHasInheritedSig</u>	553
<u>dbCVHasInheritedTerm</u>	554

Virtuoso Studio Design Environment SKILL Reference

<u>dbCVHasNetSetProp</u>	555
<u>dbDeleteSigNetExpr</u>	556
<u>dbDeleteTermNetExpr</u>	557
<u>dbFix</u>	558
<u>dbGetCellViewInheritedSig</u>	559
<u>dbGetCellViewInheritedTerm</u>	560
<u>dbGetSigNetExpr</u>	561
<u>dbGetTermNetExpr</u>	562
<u>dbIsValidSigNetExpr</u>	563
<u>dbIsValidTermNetExpr</u>	564
<u>dbIsSigInherited</u>	565
<u>dbIsTermInherited</u>	566
<u>dbParseSigNetExpr</u>	567
<u>dbParseTermNetExpr</u>	569
<u>dbReplaceSigNetExpr</u>	571
<u>dbReplaceTermNetExpr</u>	572
<u>Net and Terminal Name Syntax</u>	573
<u>Instance Name Syntax</u>	576
<u>Undo and Redo Functions</u>	577
<u>dbSetUndoHandle</u>	578
<u>dbUndoToHandle</u>	580
<u>dbGetUndoHandles</u>	581
<u>dbGetRedoHandles</u>	582
<u>dbRedoToHandle</u>	583
<u>dbEnableUndo</u>	584
<u>Fluid Shape Functions</u>	585
<u>dbGetFluidShapeByName</u>	586
<u>dbGetFluidShapeName</u>	587
<u>dbGetFluidShapes</u>	588
<u>dbIsFluidPcell</u>	589
<u>dbIsFluidShape</u>	590
<u>dbSetFluidShape</u>	591
<u>Pcell Abutment Support Functions</u>	592
<u>dbAbutIsInteractive</u>	593
<u>dbAbutIsReAbutGroup</u>	594
<u>dbSetFigAbutName</u>	595

Virtuoso Studio Design Environment SKILL Reference

<u>dbSetFigAbutRefName</u>	597
<u>dbGetFigAbutName</u>	599
<u>dbGetFigAbutRefName</u>	600
<u>dbHasFigAbutName</u>	601
<u>dbHasFigAbutRefName</u>	602
<u>dbFindAbutFig</u>	603
<u>dbUnabutGroup</u>	604
<u>Object Creation and Retrieval Functions</u>	605
<u>dbCopyShape</u>	606
<u>dbCreateLine</u>	607
<u>dbCreatePath</u>	608
<u>dbCreatePolygon</u>	610
<u>dbCreateRect</u>	611
<u>dbCreateRuler</u>	612
<u>dbCreateEllipse</u>	613
<u>dbCreateArc</u>	614
<u>dbCreateDonut</u>	616
<u>dbCreateDot</u>	617
<u>dbCreateLabel</u>	618
<u>dbCreateTextDisplay</u>	620
<u>dbHasShape</u>	624
<u>dbMoveShape</u>	625
<u>dbSetTextDisplayNameValueVisible</u>	626
<u>Shape Conversion Functions</u>	627
<u>dbConvertArcToLine</u>	628
<u>dbConvertDonutToPolygon</u>	629
<u>dbConvertEllipseToPolygon</u>	630
<u>dbConvertPathToPolygon</u>	631
<u>dbConvertPathSegToPolygon</u>	632
<u>dbConvertLineToPath</u>	633
<u>dbConvertRectToPolygon</u>	634
<u>dbCompressPointArray</u>	635
<u>Mosaic Creation and Retrieval Functions</u>	636

Virtuoso Studio Design Environment SKILL Reference

<u>dbCreateParamSimpleMosaic</u>	637
<u>dbCreateParamSimpleMosaicByMasterName</u>	639
<u>dbCreateSimpleMosaic</u>	641
<u>dbFindMosaicByName</u>	643
<u>Property Creation and Modification Functions</u>	644
<u>dbCreateProp</u>	646
<u>dbCreateRangeProp</u>	647
<u>dbCreateEnumProp</u>	649
<u>dbCreateHierProp</u>	650
<u>dbFindProp</u>	651
<u>dbGetPropByName</u>	652
<u>dbReplaceProp</u>	653
<u>dbReplaceRangeProp</u>	655
<u>dbReplaceEnumProp</u>	657
<u>dbReplaceHierProp</u>	658
<u>dbCopyProp</u>	659
<u>dbCopyPropList</u>	660
<u>dbResetPropList</u>	661
<u>dbDeletePropByName</u>	664
<u>dbCopySingleProp</u>	665
<u>dbMergeSingleProp</u>	666
<u>dbReplacePropList</u>	667
<u>dbSearchPropByName</u>	669
<u>Group Creation and Modification Functions</u>	671
<u>dbAddObjectToGroup</u>	672
<u>dbChangeGroupType</u>	673
<u>dbCreateGroup</u>	674
<u>dbDeleteGroupByName</u>	676
<u>dbDeleteObjectFromGroup</u>	677
<u>dbGetGroupByName</u>	678
<u>dbGetGroupsWithDef</u>	679
<u>dbMoveGroupMember</u>	680
<u>dbDeleteGroupMemberFromGroup</u>	682
<u>Shape Transformation Functions</u>	683
<u>dbTransformCellView</u>	684
<u>dbTransformPoint</u>	685

Virtuoso Studio Design Environment SKILL Reference

<u>dbTransformPointList</u>	686
<u>dbTransformBBox</u>	688
<u>dbConcatTransform</u>	689
<u>dbCanonicalizeAnyAngleTransform</u>	691
<u>dbConvertAnyAngleTransformFromDbToMY</u>	694
<u>Bounding Box Functions</u>	696
<u>lowerLeft</u>	698
<u>upperRight</u>	699
<u>leftEdge</u>	700
<u>rightEdge</u>	701
<u>bottomEdge</u>	702
<u>topEdge</u>	703
<u>centerBox</u>	704
<u>isLocation</u>	705
<u>isBBox</u>	706
<u>Property Bag Functions</u>	707
<u>dbOpenBag</u>	708
<u>dbReOpenBag</u>	710
<u>dbGetBag</u>	711
<u>dbGetBagOwner</u>	712
<u>dbCloseBag</u>	713
<u>dbPurgeBag</u>	714
<u>dbRefreshBag</u>	715
<u>dbMakeBagCurrent</u>	716
<u>dbSaveBag</u>	717
<u>dbIsBagReadable</u>	718
<u>dbIsBagWritable</u>	719
<u>dbGetBagTimeStamp</u>	721
<u>dbFindOpenBag</u>	722
<u>dbGetBagDdId</u>	723
<u>dbGetOpenBags</u>	724
<u>dbIsBagId</u>	725
<u>Layer Boolean Functions</u>	726

Virtuoso Studio Design Environment SKILL Reference

<u>dbLayerAnd</u>	727
<u>dbLayerAndNot</u>	729
<u>dbLayerOr</u>	731
<u>dbLayerXor</u>	733
<u>dbLayerSize</u>	735
<u>dbLayerTile</u>	737
<u>dbLayerStraddle</u>	738
<u>dbLayerNoHoles</u>	740
<u>dbLayerHoles</u>	741
<u>dbLayerInside</u>	742
<u>dbLayerOutside</u>	744
<u>dbLayerEnclose</u>	746
<u>Point Array Functions</u>	747
<u>dbPointArrayAnd</u>	748
<u>dbPointArrayAndNot</u>	749
<u>dbPointArrayOr</u>	750
<u>dbPointArraySize</u>	751
<u>dbPointArrayXor</u>	752
<u>Soft Connect Terminal Functions</u>	753
<u>dbSetTermSoftConnect</u>	755
<u>dbGetSoftConnectTermConnectToLayer</u>	756
<u>dbGetSoftConnectTermPinlessLayer</u>	757
<u>dbIsTermSoftConnect</u>	758
<u>dbSetSoftConnectTermConnectToLayer</u>	759
<u>dbSetSoftConnectTermPinlessLayer</u>	760
<u>dbSoftConnectTermHasConnectToLayer</u>	761
<u>dbSoftConnectTermHasPinlessLayer</u>	762
<u>Information Retrieval Functions</u>	763
<u>dbDebugStats2</u>	764
<u>dbGetBuildInfo</u>	765
<u>dbGetFileDatabaseType</u>	766
<u>dbGetVersion</u>	767
<u>dbSetNIMsgMode</u>	768
<u>Blockage Functions</u>	769
<u>dbCreateAreaBlockage</u>	771
<u>dbCreateAreaHalo</u>	772

Virtuoso Studio Design Environment SKILL Reference

<u>dbCreateLayerBlockage</u>	773
<u>dbCreateLayerHalo</u>	774
<u>Boundary Functions</u>	776
<u>dbCreateAreaBoundary</u>	777
<u>dbCreateClusterBoundary</u>	778
<u>dbCreateCoverObstruction</u>	779
<u>dbCreatePRBoundary</u>	780
<u>dbCreateSnapBoundary</u>	781
<u>dbFindAreaBoundaryByName</u>	782
<u>dbFindClusterBoundaryByName</u>	783
<u>dbSetBoundaryEdge</u>	784
<u>Bus Definition and Bit Order Functions</u>	785
<u>dbCreateBusNetDef</u>	786
<u>dbCreateBusTermDef</u>	787
<u>dbCreateVectorInstDef</u>	788
<u>dbFindBusNetDefByName</u>	789
<u>dbFindBusTermDefByName</u>	790
<u>dbFindVectorInstDefByName</u>	791
<u>Cluster Functions</u>	792
<u>dbCreateCluster</u>	793
<u>dbFindClusterByName</u>	794
<u>Counter Functions</u>	795
<u>dbDisableCellViewCounter</u>	796
<u>dbEnableCellViewCounter</u>	798
<u>dbGetCellViewCounter</u>	799
<u>dbGetFileTimeStamp</u>	800
<u>dbIsConnCurrent</u>	801
<u>dbSetConnCurrent</u>	802
<u>dbSetInstHeadersCounter</u>	803
<u>Interconnect Functions</u>	804
<u>dbCreateGuide</u>	805
<u>dbCreatePathSeg</u>	807
<u>dbCreateRoute</u>	816

Virtuoso Studio Design Environment SKILL Reference

<u>dbCreateVia</u>	818
<u>dbCreateCustomViaVariant</u>	822
<u>dbCreateStdViaVariant</u>	824
<u>dbFindViaVariantByName</u>	827
<u>dbGetViaLayer</u>	828
<u>dbIsCellViewStdViaMaster</u>	829
<u>dbSetPathSegPoints</u>	830
<u>dbSetViaParamDefault</u>	831
<u>dbSetViaShapePurpose</u>	832
<u>dbGetRouteAuthor</u>	833
<u>dbSetRouteAuthor</u>	835
<u>dbGetRouteSubAuthor</u>	837
<u>dbSetRouteSubAuthor</u>	838
<u>xdvReplaceInstWithVias</u>	839
<u>Trim Functions</u>	840
<u>dbGetAllTrimFillShapes</u>	841
<u>dbGetShapeTrimFillType</u>	842
<u>Patch Functions</u>	843
<u>dbGetPatchType</u>	844
<u>dbSetBeginPatch</u>	845
<u>dbSetEndPatch</u>	846
<u>dbSetFullPatch</u>	847
<u>dbSetPatchType</u>	849
<u>Marker Functions</u>	850
<u>dbCreateMarker</u>	851
<u>dbAddObjectsToMarker</u>	853
<u>dbRemoveObjectsFromMarker</u>	854
<u>dbGetMarkerName</u>	855
<u>dbSetMarkerName</u>	857
<u>Query Functions</u>	858
<u>dbBoundaryQuery</u>	859
<u>dbFigGroupQuery</u>	861
<u>dbGuideQuery</u>	863
<u>dbInstQuery</u>	865
<u>dbLayerBlockageQuery</u>	867
<u>dbMarkerQuery</u>	869

Virtuoso Studio Design Environment SKILL Reference

<u>dbPlacementBlockageQuery</u>	871
<u>dbRowQuery</u>	873
<u>dbShapeQuery</u>	875
<u>dbSteinerQuery</u>	878
<u>dbViaQuery</u>	880
<u>Row Functions</u>	882
<u>dbCreateRow</u>	883
<u>dbFindRowByName</u>	885
<u>dbFindRowHeader</u>	886
<u>Steiner Functions</u>	887
<u>dbCreateSteiner</u>	888
<u>dbCreateTapSteiner</u>	889
<u>dbIsTapSteiner</u>	890
<u>Supply and Ground Sensitivity Functions</u>	891
<u>dbGetTermGroundSensitivity</u>	892
<u>dbGetTermSupplySensitivity</u>	893
<u>dbSetTermGroundSensitivity</u>	894
<u>dbSetTermSupplySensitivity</u>	895
<u>Terminal Ordering Functions</u>	896
<u>dbCreateInstTermByPosition</u>	897
<u>dbFindTermByPosition</u>	898
<u>Terminal Antenna Functions</u>	899
<u>dbDeleteAllTerminalAntennaRules</u>	900
<u>dbDeleteTerminalAntennaRuleOnLayer</u>	901
<u>dbDeleteTerminalAntennaRules</u>	903
<u>dbGetTerminalAntennaRuleOnLayer</u>	905
<u>dbGetTerminalAntennaRules</u>	907
<u>dbSetTerminalAntennaRuleOnLayer</u>	909
<u>dbSetTerminalAntennaRules</u>	911
<u>Track Pattern Functions</u>	913
<u>dbCreateTrackPattern</u>	914
<u>GCell Pattern Function</u>	916
<u>dbCreateGCellPattern</u>	917

Virtuoso Studio Design Environment SKILL Reference

<u>Via Functions</u>	919
<u>Via GenVia Functions</u>	920
<u>dbCreateGenViaVariant</u>	921
<u>dbIsCellViewGenViaMaster</u>	923
<u>dbIsGenVia</u>	924
<u>dbIsGenViaHeader</u>	925
<u>dbIsGenViaVariant</u>	926
<u>dbSetGenViaParamDefault</u>	927
<u>Via Header Functions</u>	928
<u>dbFindViaHeaderByName</u>	929
<u>Fixed Via Function</u>	930
<u>dbIsFixedVia</u>	931
<u>Optical Data Functions</u>	932
<u>dbCellViewHasOpticalData</u>	933
<u>dbIsNetOptical</u>	934
<u>dbListTypes</u>	935
<u>Remaster SKILL Functions</u>	936
<u>dbRemasterAnyInst</u>	937
<u>dbRemasterInstHeader</u>	939
<u>Alternate Foundry Constraint Group SKILL Functions</u>	941
<u>dbGetPerDesignAFCGName</u>	942
<u>dbSetPerDesignAFCGName</u>	943

3

<u>Cellview Input/Output Database Access Functions</u>	945
<u>dbAllCellViews</u>	948
<u>dbCellViewHasVirtHier</u>	949
<u>dbCheckRecursion</u>	950
<u>dbClose</u>	951
<u>dbComputeBBox</u>	952
<u>dbComputeBBoxNoNLP</u>	953
<u>dbCopyCellView</u>	954
<u>dbCreateLib</u>	956
<u>dbDefineProc</u>	957
<u>dbFeaturePrintInfo</u>	958

Virtuoso Studio Design Environment SKILL Reference

<u>dbFindOpenCellView</u>	959
<u>dbFindOpenCellViewByName</u>	960
<u>dbFlattenInst</u>	961
<u>dbFlattenInst2</u>	964
<u>dbFullLibPath</u>	968
<u>dbFullPath</u>	969
<u>dbGetAnyInstSwitchMaster</u>	970
<u>dbGetCellViewDdId</u>	971
<u>dbGetLibDataCompressionLevel</u>	972
<u>dbGetOpenCellViews</u>	973
<u>dbHasAutoSavedFile</u>	974
<u>dbHasPanicFile</u>	975
<u>dbIsNonSKILLPcell</u>	976
<u>dbOpenAutoSavedCellView</u>	977
<u>dbOpenCellViewByType</u>	979
<u>dbOpenHier</u>	983
<u>dbOpenPanicCellView</u>	985
<u>dbOpenParamCellView</u>	987
<u>dbOpenScratchDesign</u>	988
<u>dbProduceMemName</u>	990
<u>dbPurge</u>	992
<u>dbRefreshCellView</u>	993
<u>dbRefreshLibDataCompressionLevel</u>	994
<u>dbRegPostSaveAsTrigger</u>	995
<u>dbRegPostSaveTrigger</u>	996
<u>dbRegPurgeTrigger</u>	997
<u>dbRegSaveTrigger</u>	998
<u>dbReopen</u>	999
<u>dbRestoreAndOpenAutoSavedFile</u>	1001
<u>dbRestoreAndOpenPanicFile</u>	1002
<u>dbSave</u>	1003
<u>dbSearchCDF</u>	1005
<u>dbSetAutoSave</u>	1006

Virtuoso Studio Design Environment SKILL Reference

<u>dbSetAutoSaveCallback</u>	1008
<u>dbSetLibDataCompressionLevel</u>	1010
<u>dbUndefineProc</u>	1012
<u>dbUnregPostSaveAsTrigger</u>	1013
<u>dbUnregPostSaveTrigger</u>	1014
<u>dbUnregPurgeTrigger</u>	1015
<u>dbUnregSaveTrigger</u>	1016
<u>dbUnsetLibDataCompressionLevel</u>	1017
<u>dbWriteSkill</u>	1018
<u>dbWriteSkillWithLib</u>	1021

4

Instance Creation and Retrieval Database Access Functions

1023

<u>dbCreateInst</u>	1025
<u>dbCreateParamInst</u>	1028
<u>dbCreateExtParamInst</u>	1031
<u>dbCreateExtParamInstByMasterName</u>	1034
<u>dbCreateXformPCell</u>	1036
<u>dbDumpPcDefinePcell</u>	1037
<u>dbCheckParamCell</u>	1038
<u>dbEvalParamCell</u>	1039
<u>dbFixParamCell</u>	1040
<u>dbFreeze</u>	1041
<u>dbHasInstance</u>	1043
<u>dbIsCellViewModified</u>	1044
<u>dbReplaceParam</u>	1045
<u>dbReplaceInstParamList</u>	1046
<u>dbFindAnyInstByName</u>	1047
<u>dbFindMemInstByName</u>	1048
<u>dbGetHierPathTransform</u>	1050
<u>dbGetInstByName</u>	1051
<u>dbGetInstanceByName</u>	1052
<u>dbGetInstNameNumInst</u>	1053
<u>dbGetInstTransform</u>	1054

Virtuoso Studio Design Environment SKILL Reference

<u>dbIsAnyInstBaseNameUsed</u>	1055
<u>dbCreateInstByMasterName</u>	1056
<u>dbCreateParamInstByMasterName</u>	1058
<u>dbDumpPcell</u>	1060
<u>dbDumpPcellWithLib</u>	1062
<u>dbCreateViaShapeArray</u>	1063
<u>dbSetInstHeaderMasterName</u>	1065
<u>dbIsSameMaster</u>	1066
<u>dbIsObjectBound</u>	1067
<u>dbGetCellViewSymmetry</u>	1068
<u>dbSetCellViewSymmetry</u>	1069
<u>dbGetCellViewEEQMaster</u>	1070
<u>dbSetCellViewEEQMaster</u>	1071
<u>dbIsInstTransparent</u>	1072
<u>dbSetInstTransparent</u>	1073

5

Connectivity Database Access Functions 1075

<u>Connectivity Creation and Modification Functions</u>	1076
<u>dbCreateNet</u>	1078
<u>dbCreateNamedSubNet</u>	1080
<u>dbMakeNet</u>	1082
<u>dbMergeNet</u>	1084
<u>dbCreateUniqueNamedNet</u>	1085
<u>dbSetNetNameDescend</u>	1087
<u>dbIsNetNameDescend</u>	1088
<u>dbNetIsSig</u>	1089
<u>dbDeleteAllNet</u>	1090
<u>dbCreateTerm</u>	1091
<u>dbCreateInstTerm</u>	1093
<u>dbHasShortedTerminals</u>	1095
<u>dbCreatePin</u>	1096

Virtuoso Studio Design Environment SKILL Reference

<u>dbCreateConn</u>	1098
<u>dbCreateConnByName</u>	1099
<u>dbCreateConnByNewName</u>	1101
<u>dbSubFigFromNet</u>	1103
<u>dbAddFigToNet</u>	1104
<u>dbGetNetVoltageRange</u>	1105
<u>dbSetNetVoltageRange</u>	1107
<u>dbUnsetNetVoltageRange</u>	1108
<u>dbGetNetVoltageRangeSource</u>	1109
<u>dbSetNetVoltageRangeSource</u>	1110
<u>dbUnsetNetVoltageRangeSource</u>	1111
<u>dbGetCellViewNetVoltageRange</u>	1112
<u>dbSetCellViewNetVoltageRange</u>	1113
<u>dbIsCellViewNetVoltageRangeSet</u>	1114
<u>dbIsNetVoltageRangeSet</u>	1115
<u>dbUnsetCellViewNetVoltageRange</u>	1116
<u>dbGetNetPowerDomain</u>	1117
<u>dbSetNetPowerDomain</u>	1118
<u>dbGetPowerDomainClusters</u>	1119
<u>dbRenameNet</u>	1121
<u>dbIsCellViewPhysicalOnly</u>	1122
<u>dbUnsetCellViewPhysicalOnly</u>	1123
<u>dbGetAnyTermGroundSensitivity</u>	1124
<u>dbGetAnyTermSupplySensitivity</u>	1126
<u>dbIsAnyTermGroundSensitivityConsistent</u>	1128
<u>dbIsAnyTermSupplySensitivityConsistent</u>	1129
<u>dbSetAnyTermGroundSensitivity</u>	1130
<u>dbSetAnyTermSupplySensitivity</u>	1131
<u>Equivalent Connectivity Time Functions</u>	1132
<u>dbCellViewHasEquivalentConnectivityTime</u>	1133
<u>dbGetCellViewEquivalentConnectivityTime</u>	1134
<u>dbSetCellViewEquivalentConnectivityTime</u>	1135
<u>dbUnsetCellViewEquivalentConnectivityTime</u>	1136
<u>Connectivity Access Functions</u>	1137
<u>dbDeleteEmptyNet</u>	1138
<u>dbFindNetByName</u>	1139

<u>dbFindTermByName</u>	1140
<u>dbFindSigByName</u>	1141
<u>dbFindSigByAlias</u>	1142
<u>dbGetMemNetSig</u>	1143
<u>dbGetMemNetSigName</u>	1144
<u>dbGetNetTerms</u>	1145
<u>dbGetSigNameMemNets</u>	1146
<u>dbIsSigNameGlobal</u>	1147
<u>dbGetMemName</u>	1148
<u>dbExpandToMemNameExpr</u>	1149
<u>dbGetMaxNumBit</u>	1150
<u>dbGetNameNumBit</u>	1151
<u>dbMergeSignal</u>	1152
<u>dbMoveChildNet</u>	1153
<u>dbMovePinToNet</u>	1154

6

Pin Database Access Functions 1155

<u>Pin Connection Functions</u>	1159
<u>External, Strong, and Weak Pin Connections</u>	1159
<u>dbCreateStrongGroup</u>	1162
<u>dbExternallyConnectPins</u>	1163
<u>dbGetPinConnection</u>	1164
<u>dbGetStrongGroups</u>	1166
<u>dbHasExternalConnections</u>	1167
<u>dbHasMustConnections</u>	1168
<u>dbMergeStrongGroups</u>	1169
<u>dbMustConnectPins</u>	1170
<u>dbStronglyConnectPins</u>	1171
<u>dbTransformAccessDirection</u>	1172
<u>dbWeaklyConnectPins</u>	1173
<u>dbAddFigsToPin</u>	1174

Virtuoso Studio Design Environment SKILL Reference

<u>dbAddFigToPin</u>	1175
<u>dbGetPinFigAccessDirection</u>	1176
<u>dbGetPinFigName</u>	1177
<u>dbGetPinFigPlacementStatus</u>	1178
<u>dbSetPinFigAccessDirection</u>	1179
<u>dbSetPinFigName</u>	1180
<u>dbSetPinFigPlacementStatus</u>	1181
<u>dbSetTermMustJoin</u>	1182
<u>dbSubFigFromPin</u>	1183
<u>dbSubFigsFromPin</u>	1184
<u>dbUnsetTermMustJoin</u>	1185
<u>dbSetTermMustConnectAllPins</u>	1186
<u>dbTermHasMustConnectAllPins</u>	1187
<u>Pin Group Guide Functions</u>	1188
<u>dbCreateAreaPinGroupGuide</u>	1191
<u>dbCreateEdgePinGroupGuide</u>	1192
<u>dbFindPinGroupGuideByName</u>	1193
<u>dbGetAreaBoundaryPinGroupGuide</u>	1194
<u>dbGetAreaPinGroupGuideAreas</u>	1195
<u>dbGetFigPinGroupGuide</u>	1196
<u>dbGetPRBoundaryPinGroupGuides</u>	1197
<u>dbGetPinGroupGuideEdgeBoundary</u>	1198
<u>dbGetPinGroupGuideEdgeName</u>	1199
<u>dbGetPinGroupGuideEdgeStartOffset</u>	1200
<u>dbGetPinGroupGuideEdgeStopOffset</u>	1201
<u>dbGetPinGroupGuides</u>	1202
<u>dbGetPinPinGroupGuides</u>	1203
<u>dbAddAreaToPinGroupGuide</u>	1204
<u>dbAddFigToPinGroupGuide</u>	1205
<u>dbAddPinToPinGroupGuide</u>	1206
<u>dbSubAreaFromPinGroupGuide</u>	1207
<u>dbSubFigFromPinGroupGuide</u>	1208
<u>dbSubPinFromPinGroupGuide</u>	1209
<u>dbRemovePinGroupGuideEdge</u>	1210
<u>dbSetPinGroupGuideEdge</u>	1211
<u>dbSetPinGroupGuideEdgeStartOffset</u>	1212

Virtuoso Studio Design Environment SKILL Reference

<u>dbSetPinGroupGuideEdgeStopOffset</u>	1213
<u>dbGetAreaBoundaryValidRoutingLayers</u>	1214
<u>dbAreaBoundaryHasLayerPriorityOrder</u>	1215
<u>dbAreaBoundaryHasValidRoutingLayers</u>	1216
<u>dbGetAreaBoundaryLayerPriorityOrder</u>	1217
<u>dbGetPinGroupGuideKeepoutSpacing</u>	1218
<u>dbGetPinGroupGuideLayerPriorityOrder</u>	1219
<u>dbGetPinGroupGuideMinPinSpacing</u>	1220
<u>dbGetPinGroupGuideMinPinWidth</u>	1221
<u>dbGetPinGroupGuideValidRoutingLayers</u>	1222
<u>dbGetPinMinPinSpacing</u>	1223
<u>dbGetPinMinPinWidth</u>	1224
<u>dbPinGroupGuideHasKeepoutSpacing</u>	1225
<u>dbPinGroupGuideHasLayerPriorityOrder</u>	1226
<u>dbPinGroupGuideHasMinPinSpacing</u>	1227
<u>dbPinGroupGuideHasMinPinWidth</u>	1228
<u>dbPinGroupGuideHasValidRoutingLayers</u>	1229
<u>dbPinHasMinPinSpacing</u>	1230
<u>dbPinHasMinPinWidth</u>	1231
<u>dbSetAreaBoundaryLayerPriorityOrder</u>	1232
<u>dbSetAreaBoundaryValidRoutingLayers</u>	1233
<u>dbSetPinGroupGuideKeepoutSpacing</u>	1234
<u>dbSetPinGroupGuideLayerPriorityOrder</u>	1235
<u>dbSetPinGroupGuideMinPinSpacing</u>	1236
<u>dbSetPinGroupGuideMinPinWidth</u>	1237
<u>dbSetPinGroupGuideValidRoutingLayers</u>	1238
<u>dbSetPinMinPinSpacing</u>	1239
<u>dbSetPinMinPinWidth</u>	1240
<u>dbUnsetAreaBoundaryLayerPriorityOrder</u>	1241
<u>dbUnsetAreaBoundaryValidRoutingLayers</u>	1242
<u>dbUnsetPinGroupGuideKeepoutSpacing</u>	1243
<u>dbUnsetPinGroupGuideLayerPriorityOrder</u>	1244
<u>dbUnsetPinGroupGuideMinPinSpacing</u>	1245

Virtuoso Studio Design Environment SKILL Reference

<u>dbUnsetPinGroupGuideMinPinWidth</u>	1246
<u>dbUnsetPinGroupGuideValidRoutingLayers</u>	1247
<u>dbUnsetPinMinPinSpacing</u>	1248
<u>dbUnsetPinMinPinWidth</u>	1249

7

<u>Placement Database Access Functions</u>	1251
<u>dbCreateBackgroundDef</u>	1255
<u>dbCreateBackgroundDefByAttr</u>	1257
<u>dbGetBackgroundDefAttr</u>	1259
<u>dbGetCellViewBackgroundDef</u>	1260
<u>dbCreateCompTypeSetDef</u>	1261
<u>dbCreateCompTypeSetDefByAttr</u>	1263
<u>dbGetCellViewCompTypeSetDef</u>	1265
<u>dbGetCompTypeSetDefAttr</u>	1266
<u>dbGetCompTypeSetDefCompFilters</u>	1267
<u>dbSetCompTypeSetDefCompFilters</u>	1269
<u>dbUnsetCompTypeSetDefCompFilters</u>	1270
<u>dbCreateRailDef</u>	1272
<u>dbCreateRailDefByAttr</u>	1276
<u>dbGetCellViewRailDef</u>	1278
<u>dbGetRailDefAttr</u>	1279
<u>dbGetRailDefFigType</u>	1280
<u>dbSetRailDefFigType</u>	1282
<u>dbAddPlaceAreaBackgroundDef</u>	1283
<u>dbAddPlaceAreaCompTypeSetDef</u>	1284
<u>dbAddPlaceAreaRailDef</u>	1286
<u>dbCreatePlaceArea</u>	1288
<u>dbGetCellViewPlaceArea</u>	1290
<u>dbGetPlaceAreaBackgroundDef</u>	1291
<u>dbGetPlaceAreaCompTypeSetDef</u>	1292
<u>dbGetPlaceAreaOrient</u>	1293
<u>dbGetPlaceAreaRailDef</u>	1294
<u>dbGetPlaceAreaUtilization</u>	1295
<u>dbGetPlacementGrid</u>	1296

Virtuoso Studio Design Environment SKILL Reference

<u>dbRemovePlaceAreaBackgroundDef</u>	1297
<u>dbRemovePlaceAreaCompTypeSetDef</u>	1299
<u>dbRemovePlaceAreaRailDef</u>	1301
<u>dbSetPlaceAreaOrient</u>	1303
<u>dbSetPlaceAreaUtilization</u>	1304
<u>dbSetPlacementGrid</u>	1305
<u>dbAddRowBackgroundDef</u>	1306
<u>dbAddRowCompTypeSetDef</u>	1307
<u>dbGetRowBackgroundDef</u>	1309
<u>dbGetRowCompTypeSetDef</u>	1310
<u>dbAddRowRailDef</u>	1311
<u>dbGetRowRailDef</u>	1313
<u>dbRemoveRowCompTypeSetDef</u>	1314
<u>dbRemoveRowBackgroundDef</u>	1316
<u>dbRemoveRowRailDef</u>	1317
<u>dbCreateRowRegionSpec</u>	1319
<u>dbFindRowRegionSpec</u>	1324
<u>dbGetCellViewRowRegionSpecs</u>	1325
<u>dbAttachRowRegionToPRBoundary</u>	1326
<u>dbCreateRowRegion</u>	1327
<u>dbDetachPRBoundaryFromRowRegion</u>	1329
<u>dbFindRowRegion</u>	1330
<u>dbFlattenRowRegion</u>	1331
<u>dbGetCellViewRowRegions</u>	1332
<u>dbGetRowRegionAttachedToPRBoundary</u>	1333
<u>dbGetRowRegionPlaceRows</u>	1334
<u>dbGetRowRegionPoints</u>	1335
<u>dbGetRowRegionRowRegionSpec</u>	1336
<u>dbGetRowRegionSpecRowRegions</u>	1337
<u>dbGetRowRegionUsesPartialRowSpec</u>	1338
<u>dbIsRowRegion</u>	1339
<u>dbIsRowRegionAttachedToPRBoundary</u>	1340
<u>dbRowRegionPointsCutOut</u>	1341

Virtuoso Studio Design Environment SKILL Reference

<u>dbRebuildRowRegion</u>	1342
<u>dbRowRegionMatchesSpec</u>	1343
<u>dbSetRowRegionPoints</u>	1345
<u>dbSetRowRegionRowRegionSpec</u>	1346
<u>dbSetRowRegionUsesPartialRowSpec</u>	1348
<u>dbGetPlaceRowVersion</u>	1349
<u>dbGetRowRegionVersion</u>	1350
<u>dbIsImplicit</u>	1351
<u>dbIsInRowRegion</u>	1352

8

Topology Pattern Database Access Functions 1353

<u>dbCreateTopology</u>	1358
<u>dbDeleteTopology</u>	1359
<u>dbGetCellViewTopologies</u>	1360
<u>dbGetNetTopologies</u>	1361
<u>dbGetTopologyName</u>	1362
<u>dbGetTopologyNet</u>	1363
<u>dbGetTopologyObjects</u>	1364
<u>dbGetTopologyPatternType</u>	1365
<u>dbSetTopologyName</u>	1366
<u>dbSetTopologyNet</u>	1367
<u>dbCreateTrunk</u>	1368
<u>dbDeleteTrunk</u>	1369
<u>dbGetTrunkAnchor</u>	1370
<u>dbGetTrunkDirection</u>	1371
<u>dbGetTrunkLongOffset1</u>	1372
<u>dbGetTrunkLongOffset2</u>	1373
<u>dbGetTrunkLongOffsetSource</u>	1374
<u>dbGetTrunkLongOffsetSource1</u>	1375
<u>dbGetTrunkLongOffsetSource2</u>	1376
<u>dbGetTrunkName</u>	1377
<u>dbGetTrunkOrthoOffset</u>	1378
<u>dbGetTrunkSide</u>	1379
<u>dbGetTrunkTopology</u>	1380

Virtuoso Studio Design Environment SKILL Reference

<u>dbGetTrunkTwigs</u>	1381
<u>dbSetTrunkAnchor</u>	1382
<u>dbSetTrunkLongOffset1</u>	1383
<u>dbSetTrunkLongOffset2</u>	1384
<u>dbSetTrunkLongOffsetSource</u>	1385
<u>dbSetTrunkLongOffsetSource1</u>	1386
<u>dbSetTrunkLongOffsetSource2</u>	1387
<u>dbSetTrunkName</u>	1388
<u>dbSetTrunkDirection</u>	1389
<u>dbSetTrunkOrthoOffset</u>	1390
<u>dbSetTrunkSide</u>	1391
<u>dbCreateTwig</u>	1392
<u>dbDeleteTwig</u>	1393
<u>dbGetTwigName</u>	1394
<u>dbGetTwigObject</u>	1395
<u>dbGetTwigPinName</u>	1396
<u>dbGetShapeTwigs</u>	1397
<u>dbGetInstTermTwigs</u>	1398
<u>dbGetViaTwigs</u>	1399
<u>dbGetTwigTrunk</u>	1400
<u>dbSetTwigName</u>	1401
<u>dbSetTwigObject</u>	1402
<u>dbSetTwigPinName</u>	1403
<u>dbCreateTrunkConnect</u>	1404
<u>dbDeleteTrunkConnect</u>	1405
<u>dbGetTrunkConnectName</u>	1406
<u>dbGetTrunkConnectTopology</u>	1407
<u>dbGetTrunkConnectTrunks</u>	1408
<u>dbSetTrunkConnectName</u>	1409
<u>dbSetTrunkConnectTrunks</u>	1410
<u>dbCreateStrap</u>	1411
<u>dbDeleteStrap</u>	1412
<u>dbGetStrapName</u>	1413

Virtuoso Studio Design Environment SKILL Reference

<u>dbGetStrapObjects</u>	1414
<u>dbGetStrapTopology</u>	1415
<u>dbGetStrapTwig</u>	1416
<u>dbGetInstTermStraps</u>	1417
<u>dbGetViaStraps</u>	1418
<u>dbGetShapeStraps</u>	1419
<u>dbAddStrapObject</u>	1420
<u>dbRemoveStrapObject</u>	1421
<u>dbSetStrapName</u>	1422
<u>dbSetStrapTwig</u>	1423
<u>dbIsTopologyPatternId</u>	1424
<u>dbNetHasTopology</u>	1425

9

Generator Database Access Functions .. 1427

<u>dbStartGenAnyInstToInstTerm</u>	1428
<u>dbStartGenInstHeader</u>	1429
<u>dbStartGenInstHeaderToAnyInst</u>	1430
<u>dbStartGenLPPHeaderToShape</u>	1431
<u>dbStartGenLPToShape</u>	1432
<u>dbStartGenMarker</u>	1433
<u>dbStartGenNet</u>	1434
<u>dbStartGenNetToInstTerm</u>	1435
<u>dbStartGenNetToRoute</u>	1436
<u>dbStartGenRoute</u>	1437
<u>dbStartGenShape</u>	1438
<u>dbStartGenTerm</u>	1439
<u>dbStartGenViaHeader</u>	1440
<u>dbStartGenViaHeaderToVia</u>	1441
<u>dbStopGen</u>	1442

10

InPlace Cover Obstruction Database Access Functions .. 1443

<u>dbCreateInPlaceCoverObstruction</u>	1445
<u>dbDestroyInPlaceCoverObstruction</u>	1446

<u>dbGetInPlaceCoverObstructionAbstractFileName</u>	1447
<u>dbGetInPlaceCoverObstructionAbstractFilePath</u>	1448
<u>dbGetInPlaceCoverObstructionAbstractViewName</u>	1449
<u>dbGetInPlaceCoverObstructionBloat</u>	1450
<u>dbGetInPlaceCoverObstructionBlockageAttributeValue</u>	1451
<u>dbGetInPlaceCoverObstructionBlockageModel</u>	1452
<u>dbGetInPlaceCoverObstructionDoughnutHalo</u>	1453
<u>dbGetInPlaceCoverObstructionLayers</u>	1454
<u>dbGetInPlaceCoverObstructionMaxMask</u>	1455
<u>dbGetInPlaceCoverObstructionMergeDistance</u>	1456
<u>dbGetInPlaceCoverObstructionMinMask</u>	1457
<u>dbGetInPlaceCoverObstructionNeedsRemodeling</u>	1458
<u>dbGetInPlaceCoverObstructionPinCutOut</u>	1459
<u>dbGetInPlaceCoverObstructionPinRemodeling</u>	1460
<u>dbGetInPlaceCoverObstructionSpacingModel</u>	1461
<u>dbInPlaceCoverObstructionExists</u>	1462
<u>dbSetInPlaceCoverObstructionAbstractFileName</u>	1463
<u>dbSetInPlaceCoverObstructionAbstractFilePath</u>	1464
<u>dbSetInPlaceCoverObstructionAbstractViewName</u>	1465
<u>dbSetInPlaceCoverObstructionBloat</u>	1466
<u>dbSetInPlaceCoverObstructionBlockageAttributeValue</u>	1467
<u>dbSetInPlaceCoverObstructionBlockageModel</u>	1468
<u>dbSetInPlaceCoverObstructionDoughnutHalo</u>	1469
<u>dbSetInPlaceCoverObstructionMergeDistance</u>	1470
<u>dbSetInPlaceCoverObstructionNeedsRemodeling</u>	1471
<u>dbSetInPlaceCoverObstructionPinCutOut</u>	1472
<u>dbSetInPlaceCoverObstructionPinRemodeling</u>	1473
<u>dbSetInPlaceCoverObstructionSpacingModel</u>	1474

11

<u>Figure Database Access Functions</u>	1477
<u>FigGroup Creation and Modification Functions</u>	1478

Virtuoso Studio Design Environment SKILL Reference

<u>dbCreateFigGroup</u>	1479
<u>dbAddFigToFigGroup</u>	1480
<u>dbRemoveFigFromFigGroup</u>	1481
<u>dbGetFigGroupByName</u>	1482
<u>Figure Creation and Retrieval Functions</u>	1483
<u>dbCopyFig</u>	1484
<u>dbMoveFig</u>	1486
<u>dbGetOverlaps</u>	1487
<u>dbGetTrueOverlaps</u>	1491
<u>dbGetNeighbor</u>	1494
<u>dbGetNeighborList</u>	1497
<u>dbGetMaxHierDepth</u>	1499
<u>dbGetPathSegStyle</u>	1500
<u>dbSetPathSegStyle</u>	1502
<u>dbGetPathBoundary</u>	1503
<u>dbCheckParentChild</u>	1504
<u>dbProduceOverlap</u>	1506
<u>dbProduceOverlapInst</u>	1508

12

Figure Set Database Access Functions 1511

<u>FigSet Functions</u>	1512
<u>txCreateFigSet</u>	1514
<u>txGetFigSetData</u>	1515
<u>txAppendObjectToFigSet</u>	1516
<u>txClearFigSet</u>	1517
<u>txCloneFigSet</u>	1518
<u>txConcatFigSet</u>	1520
<u>txCopyFigSet</u>	1522
<u>txFindFigSet</u>	1524
<u>txDeleteFigSet</u>	1525
<u>txGetCellViewFigSets</u>	1527
<u>txGetFigSetBBox</u>	1528
<u>txIsFigSet</u>	1531
<u>txMakeFigSet</u>	1532

<u>txMoveFigSet</u>	1534
<u>txPrependObjectToFigSet</u>	1536
<u>txRemoveFigSetData</u>	1537
<u>txRemoveObjectFromFigSet</u>	1538
<u>FigSetGroup Functions</u>	1539
<u>Use of the SKILL Operator ~> with FigGroups</u>	1539
<u>dbMakeFigSetGroup</u>	1541
<u>dbFindFigSetGroup</u>	1543
<u>dbFindFigSetGroupByName</u>	1544
<u>dbGetFigSetGroupName</u>	1545
<u>dbGetCellViewFigSetGroups</u>	1546
<u>dbIsFigSetGroup</u>	1547
<u>dbDeleteFigSetGroup</u>	1548

13

Multi-Patterning Technology Database Access Functions 1551

<u>dbCellViewAreLayerShiftsValid</u>	1554
<u>dbCellViewClearLayerShifts</u>	1555
<u>dbCellViewHasLayerShifts</u>	1556
<u>dbCellViewUpdateLayerShifts</u>	1557
<u>dbCellViewInitForLayerShifting</u>	1558
<u>dbClearUnusedOccShapeColorData</u>	1559
<u>dbInstGetLayerShifts</u>	1561
<u>dbInstGetLayerShifts2</u>	1562
<u>dbInstHasLayerShifts</u>	1564
<u>dbInstSetLayerShifts</u>	1565
<u>dbInstSetLayerShifts2</u>	1566
<u>dbInstClearLayerShifts</u>	1568
<u>dbIsShapeColoringAllowed</u>	1569
<u>dbIsShapeColored</u>	1570
<u>dbGetShapeColor</u>	1571
<u>dbSetShapeColor</u>	1572

Virtuoso Studio Design Environment SKILL Reference

<u>dbIsShapeColorLocked</u>	1573
<u>dbSetShapeColorLocked</u>	1574
<u>dbIsBlockageColored</u>	1575
<u>dbIsBlockageColoringAllowed</u>	1576
<u>dbGetBlockageColor</u>	1577
<u>dbSetBlockageColor</u>	1578
<u>dbIsViaColorStateLocked</u>	1579
<u>dbSetViaColorStateLocked</u>	1580
<u>dbIsViaColorStateLayerLocked</u>	1581
<u>dbSetViaColorStateLayerLocked</u>	1582
<u>dbGetViaLayerNumColorMasks</u>	1583
<u>dbGetViaLayer1Control</u>	1584
<u>dbGetViaLayer2Control</u>	1585
<u>dbGetViaCutLayerControl</u>	1586
<u>dbGetViaLayerControl</u>	1587
<u>dbSetViaLayer1Control</u>	1589
<u>dbSetViaLayer2Control</u>	1591
<u>dbSetViaCutLayerControl</u>	1593
<u>dbSetViaLayerControl</u>	1594
<u>dbSetViaColorInfo</u>	1596
<u>dbIsTrackPatternColored</u>	1599
<u>dbIsTrackPatternColorAlternating</u>	1600
<u>dbIsTrackPatternColoringAllowed</u>	1601
<u>dbGetTrackPatternFirstTrackColor</u>	1602
<u>dbSetTrackPatternFirstTrackColor</u>	1603
<u>dbSetTrackPatternColorAlternating</u>	1604
<u>dbCreateSameMaskGroup</u>	1605
<u>dbCreateDiffMaskGroup</u>	1606
<u>dbGetSameMaskDiffMaskGroups</u>	1608
<u>dbGetShapeSameMaskGroups</u>	1609
<u>dbColorShapeQuery</u>	1610
<u>dbGetIntegrationColorModel</u>	1612
<u>dbGetColorModel</u>	1613
<u>dbSetIntegrationColorModel</u>	1614
<u>dbSetColorModel</u>	1615
<u>dbRemoveLayerColorData</u>	1616

<u>dbColorShapeQuery2</u>	1618
<u>dbGetColoredOccShapes</u>	1623
<u>dbGetShapeEffectiveColor</u>	1627
<u>dbSetOccShapeColor</u>	1631
<u>dbSetOccShapeColorLocked</u>	1634
<u>dbGetHierColorOverride</u>	1637
<u>dbGetShapeColorLockType</u>	1640
<u>dbSetShapeColorLockType</u>	1643
<u>dbGetViaLayerColorLockType</u>	1646
<u>dbSetViaLayerColorLockType</u>	1649
<u>dbCellViewHasUnknownColorLockChanges</u>	1652
<u>dbGetShapeEffectiveColorLockType</u>	1654
<u>dbCellViewResetAllColorLockTypes</u>	1658
<u>dbSetOccShapeColorLockType</u>	1660

14

Width Spacing Pattern Database Access Functions 1665

<u>Width Spacing Pattern Functions</u>	1666
<u>dbCreateWidthSpacingPattern</u>	1667
<u>dbCreateWidthSpacingPatternWithColor</u>	1675
<u>dbFindWidthSpacingPattern</u>	1680
<u>dbGetWidthSpacingPatternAllowedRepeatMode</u>	1681
<u>dbGetWidthSpacingPatternDefaultRepeatMode</u>	1682
<u>dbGetWidthSpacingPatterns</u>	1683
<u>dbSetWidthSpacingPatternRepeatMode</u>	1684
<u>Width Spacing Pattern Group Functions</u>	1685
<u>dbCreateWidthSpacingPatternGroup</u>	1686
<u>dbGetWidthSpacingPatternGroups</u>	1687
<u>dbFindWidthSpacingPatternGroup</u>	1688
<u>Width Spacing Snap Pattern Def Functions</u>	1689
<u>dbCreateWidthSpacingSnapPatternDef</u>	1690
<u>dbFindWidthSpacingSnapPatternDefByName</u>	1693

Virtuoso Studio Design Environment SKILL Reference

<u>dbDeleteWidthSpacingSnapPatternDef</u>	1694
<u>dbGetWidthSpacingSnapPatternDefsByLP</u>	1695
<u>Related Snap Pattern Functions</u>	1696
<u>dbCreateRelatedSnapPatterns</u>	1697
<u>dbGetRelatedSnapPatterns</u>	1699
<u>dbFindRelatedSnapPatterns</u>	1700
<u>Pattern Region Functions</u>	1701
<u>dbGetPatternRegionActivePattern</u>	1702
<u>dbSetPatternRegionActivePattern</u>	1703
<u>dbHasPatternRegionActivePattern</u>	1704
<u>dbGetPatternRegionAllowedPatterns</u>	1705
<u>dbSetPatternRegionAllowedPatterns</u>	1706
<u>dbGetPatternRegionAllowedPatternGroups</u>	1707
<u>dbSetPatternRegionAllowedPatternGroups</u>	1708
<u>dbGetPatternRegionCommittedPatterns</u>	1709
<u>dbGetPatternRegionCommittedPatternShiftColor</u>	1710
<u>dbGetPatternRegionCreateType</u>	1711
<u>dbGetPatternRegionShiftColor</u>	1712
<u>dbSetPatternRegionShiftColor</u>	1713
<u>dbHasPatternRegionInfo</u>	1714
<u>dbGetPatternRegionName</u>	1715
<u>dbSetPatternRegionName</u>	1717
<u>dbGetPatternRegionWidthSpacingSnapPatternDefName</u>	1719
<u>dbSetPatternRegionWidthSpacingSnapPatternDefName</u>	1720
<u>dbGetPatternRegionDefaultRepeatMode</u>	1721
<u>dbSetPatternRegionDefaultRepeatMode</u>	1722
<u>dbGetPatternRegionRepeatMode</u>	1723
<u>dbSetPatternRegionRepeatMode</u>	1724
<u>Active Wire Types Functions</u>	1725
<u>Per-Design Database Basis Functions</u>	1726
<u>dbGetCellViewActiveWireTypes</u>	1726
<u>dbSetCellViewActiveWireTypes</u>	1727
<u>dbClearAllActiveWireTypes</u>	1728
<u>Per-widthSpacingSnapPatternDef Basis Functions</u>	1729
<u>dbGetCellViewActiveWireTypesForWidthSpacingSnapPatternDef</u>	1729
<u>dbSetCellViewActiveWireTypesForWidthSpacingSnapPatternDef</u>	1730

<u>Per-Pattern Region Basis Functions</u>	1731
<u>dbGetPatternRegionActiveWireTypes</u>	1731
<u>dbSetPatternRegionActiveWireTypes</u>	1732
<u>dbSetPatternRegionNoActiveWireTypes</u>	1733
<u>dbHasPatternRegionNoActiveWireTypes</u>	1734
<u>Global Grid Offset Functions</u>	1735
<u>dbSetGlobalGridOffsetReferenceType</u>	1736
<u>dbGetGlobalGridOffsetReferenceType</u>	1738
<u>dbGetGlobalGridDefaultRepeatMode</u>	1739
<u>dbSetGlobalGridDefaultRepeatMode</u>	1740
<u>dbGetGlobalGridRepeatMode</u>	1741
<u>dbSetGlobalGridRepeatMode</u>	1742

15

Hierarchical Design Functions 1743

<u>Hierarchy Database Functions</u>	1744
<u>dbDeleteHierCache</u>	1745
<u>dbGetHierCacheUsage</u>	1747
<u>dbHasValidHierCache</u>	1749
<u>dbSetHierCacheUsage</u>	1751
<u>hdbAddConfigToBag</u>	1753
<u>hdbBind</u>	1754
<u>hdbCloseConfigsInBag</u>	1756
<u>hdbCreateConfigBag</u>	1758
<u>hdbCreatePathVectorStack</u>	1759
<u>hdbConfigHasOccRules</u>	1760
<u>hdbExplainBinding</u>	1761
<u>hdbExplainSignature</u>	1763
<u>hdbExplainStop</u>	1764
<u>hdbGetPathStackStr</u>	1765
<u>hdbGetPathStr</u>	1767
<u>hdbGetPathVectorStackTop</u>	1768

Virtuoso Studio Design Environment SKILL Reference

<u>hdbIsAtStopPoint</u>	1770
<u>hdbIsConfig</u>	1771
<u>hdbIsOpenConfig</u>	1772
<u>hdbReOpenNewConfig</u>	1773
<u>hdbPopPathVect</u>	1775
<u>hdbPushPathVect</u>	1777
<u>hdbSaveACopy</u>	1779
<u>hdbSaveAs</u>	1781
<u>hdbSave</u>	1783
<u>hdbClose</u>	1784
<u>hdbOpen</u>	1785
<u>hdbCreatePathVector</u>	1787
<u>hdbDestroyPathVector</u>	1788
<u>hdbGetCellName</u>	1789
<u>hdbGetLibName</u>	1790
<u>hdbGetTopCellName</u>	1791
<u>hdbGetTopLibName</u>	1792
<u>hdbGetTopViewName</u>	1793
<u>hdbGetViewName</u>	1794
<u>hdbPopCell</u>	1795
<u>hdbPushCell</u>	1796
<u>hdbSetObjBindRule</u>	1798
<u>hdbSetDefaultLibListString</u>	1802
<u>hdbSetDefaultViewListString</u>	1803
<u>hdbSetDefaultStopListString</u>	1804
<u>hdbSetTopCellViewName</u>	1805
<u>hdbBindBit</u>	1806
<u>hdbBitHasRules</u>	1808
<u>hdbHasIterInstBitRules</u>	1810
<u>hdbIsIterInst</u>	1811
<u>hdbIterInstHasBitRules</u>	1813
<u>hdbPushBitCell</u>	1815
<u>Occurrence Property Functions</u>	1817
<u>occpClose</u>	1818
<u>occpCreateProp</u>	1819
<u>occpGetBestProp</u>	1821

Virtuoso Studio Design Environment SKILL Reference

<u>occpGetBitInstProp</u>	1823
<u>occpGetCellProp</u>	1825
<u>occpGetDefaultProp</u>	1827
<u>occpGetInstProp</u>	1828
<u>occpGetPathProp</u>	1830
<u>occpOpen</u>	1832
<u>occpRmBitInstProp</u>	1834
<u>occpRmCellProp</u>	1836
<u>occpRmDefaultProp</u>	1837
<u>occpRmPathProp</u>	1838
<u>occpRmInstProp</u>	1839
<u>occpSave</u>	1841
<u>occpSetBitInstProp</u>	1842
<u>occpSetCellProp</u>	1844
<u>occpSetDefaultProp</u>	1846
<u>occpSetInstProp</u>	1847
<u>occpSetPathProp</u>	1849
<u>Parent/Child Database Functions</u>	1851
<u>pcdbClose</u>	1852
<u>pcdbGetInstGen</u>	1853
<u>pcdbGetInstMasterGen</u>	1854
<u>pcdbGetIgnoredInstMasterGen</u>	1855
<u>pcdbInstMasterCell</u>	1856
<u>pcdbInstMasterCellFixed</u>	1857
<u>pcdbInstMasterLib</u>	1858
<u>pcdbInstMasterLibFixed</u>	1859
<u>pcdbInstMasterView</u>	1860
<u>pcdbInstMasterViewFixed</u>	1861
<u>pcdbInstName</u>	1863
<u>pcdbInstWidth</u>	1864
<u>pcdbNextInst</u>	1865
<u>pcdbNextInstMaster</u>	1866
<u>pcdbOpen</u>	1868

Virtuoso Studio Design Environment SKILL Reference

<u>pcdbViewHasPcdbInfo</u>	1871
<u>pcdbSave</u>	1873
<u>Example for all pcdb Functions</u>	1874

16

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)..... 1877

<u>Enable Virtuoso RF Solution and MultiTech Framework Functions</u>	1880
<u>dbIsMultiTechEnabled</u>	1880
<u>dbIsVRFEnabled</u>	1882
<u>dbIsVRFInfraEnabled</u>	1883
<u>Curved Shape Functions</u>	1884
<u>dbAddPinToPinGroup</u>	1885
<u>dbCreateCurvedAreaBoundary</u>	1887
<u>dbCreateCurvedPath</u>	1891
<u>dbGetCurvedPathRealWidth</u>	1894
<u>dbSetCurvedPathRealWidth</u>	1896
<u>dbCreateCurvedPolygon</u>	1898
<u>dbCreateMultipleCurvedPolygons</u>	1907
<u>dbCreatePlaceBound</u>	1910
<u>dbGetPlaceBounds</u>	1912
<u>dbIsCurved</u>	1914
<u>dbGetCurvedAreaBoundaryPoints</u>	1916
<u>dbGetCurvedPolygonPoints</u>	1918
<u>dbSetCurvedAreaBoundaryPoints</u>	1920
<u>dbCreatePinGroup</u>	1922
<u>dbGetPinGroupPlaceBoundNameAndFace</u>	1924
<u>dbGetPinGroups</u>	1926
<u>dbSetCurvedPolygonPoints</u>	1928
<u>dbCreateRegionBoundary</u>	1930
<u>dbGetPlaceBoundMaxHeight</u>	1933
<u>dbSetPlaceBoundMaxHeight</u>	1935
<u>dbGetPlaceBoundMinHeight</u>	1937
<u>dbSetPlaceBoundMinHeight</u>	1939
<u>dbGetDieStack</u>	1941

Virtuoso Studio Design Environment SKILL Reference

<u>dbGetDieStacks</u>	1943
<u>dbGetRegionBoundaryLayer</u>	1945
<u>dbGetRegionBoundaryLayerType</u>	1946
<u>dbGetRegionBoundaryName</u>	1947
<u>dbIsRegionBoundary</u>	1948
<u>dbMapPinGroupPins</u>	1949
<u>Virtuoso RF Solution Package Infrastructure Functions</u>	1951
<u>pkxAddFuncDefPin</u>	1952
<u>pkxCreateCompDef</u>	1953
<u>pkxCreateCompDefPin</u>	1954
<u>pkxCreateFuncDef</u>	1955
<u>pkxCreateFuncDefPin</u>	1956
<u>pkxDeleteCompDef</u>	1958
<u>pkxFindCompDef</u>	1959
<u>pkxFindCompDefPin</u>	1960
<u>pkxFindFuncDef</u>	1961
<u>pkxFindFuncDefPin</u>	1962
<u>pkxGetCompDefPins</u>	1964
<u>pkxGetCompDefs</u>	1965
<u>pkxGetFuncDefPins</u>	1966
<u>pkxGetFuncDefs</u>	1967
<u>pkxGetProp</u>	1968
<u>pkxGetProps</u>	1969
<u>pkxSetProp</u>	1970
<u>pkxPadFile</u>	1972
<u>pkxUnPadFile</u>	1973

17

<u>Photonics Functions (Virtuoso Photonics Option)</u>	1975
<u>cclsEnabled</u>	1977
<u>Curvy Core Mathematical Object Creation Functions</u>	1978
<u>ccCreateCurve</u>	1979

Virtuoso Studio Design Environment SKILL Reference

<u>ccCreateFacet</u>	1981
<u>ccCreateLineSegment</u>	1983
<u>ccCreatePolyCurve</u>	1984
<u>ccGetFacetCorners</u>	1986
<u>ccGetFacetOrigin</u>	1987
<u>Curvy Core Shape Creation Functions</u>	1989
<u>ccCreateEllipse</u>	1990
<u>ccCreateLine</u>	1993
<u>ccCreatePath</u>	1995
<u>ccCreatePie</u>	1999
<u>ccCreatePolygon</u>	2002
<u>ccCreateRect</u>	2005
<u>ccCreateSurface</u>	2007
<u>ccCreateSurfaceFromPath</u>	2009
<u>Curvy Core Shape Geometry Functions</u>	2011
<u>ccCopyFig</u>	2012
<u>ccConvertIntoCurvy</u>	2014
<u>ccGenFigs</u>	2015
<u>ccGenOffsetFig</u>	2017
<u>ccGetFromFig</u>	2020
<u>ccMoveFig</u>	2021
<u>ccRemoveFigs</u>	2023
<u>Curvy Core Mathematical Object Functions</u>	2024
<u>ccLayerOffset</u>	2025
<u>ccOffsetPolyCurve</u>	2027
<u>ccOrientCurve</u>	2029
<u>ccRenameFacet</u>	2031
<u>Curvy Core Mathematical Object Attribute Query Functions</u>	2032
<u>ccCubicSplineConnector</u>	2033
<u>ccGetCurveLength</u>	2035
<u>ccGetCurvePoint</u>	2036
<u>ccGetCurveDerivative</u>	2038
<u>ccGetPolyCurveMinRadius</u>	2040
<u>ccIsLinearCurve</u>	2041
<u>Curvy Core Shape Object Attribute Query Functions</u>	2042
<u>ccGetAll</u>	2043

Virtuoso Studio Design Environment SKILL Reference

<u>ccGetPathBoundary</u>	2044
<u>ccGetPathCenterLine</u>	2045
<u>ccGetPathEdge</u>	2046
<u>ccFindSurfaceFacet</u>	2048
<u>Curvy Core Transformations Functions</u>	2049
<u>ccLayerAnd</u>	2050
<u>ccLayerAndNot</u>	2052
<u>ccLayerOr</u>	2054
<u>ccMirrorFacet</u>	2056
<u>ccReversePolyCurve</u>	2057
<u>ccSmoothenFig</u>	2058
<u>ccSmoothenPolyCurve</u>	2059
<u>ccTransformFacet</u>	2060
<u>ccTransformPolyCurve</u>	2061
<u>Curvy Core Connector Functions</u>	2062
<u>ccBendConnector</u>	2063
<u>ccCreateConnectorWidthExpressionList</u>	2066
<u>ccCurveConnector</u>	2070
<u>ccSpiralConnector</u>	2074
<u>ccTromboneConnector</u>	2077
<u>ccWaypointConnector</u>	2079
<u>ccWideConnector</u>	2083
<u>Curvy Core Optical Rule Engine Functions</u>	2088
<u>ccORE</u>	2089
<u>ccORECompile</u>	2093
<u>ccOREGetContext</u>	2095
<u>ccOREGetContextObject</u>	2097
<u>ccOREGetContexts</u>	2099
<u>ccOREPurgeContext</u>	2100
<u>ccOREPurgeContexts</u>	2102
<u>Photonics Database Functions</u>	2104
<u>dbCellViewHasPhotonicPinFig</u>	2105
<u>dbGetPhotonicPinFigAngle</u>	2106

Virtuoso Studio Design Environment SKILL Reference

<u>dbGetPhotonicPinFigRadius</u>	2107
<u>dbGetPhotonicPinFigWidth</u>	2108
<u>dbIsPhotonicsEnabled</u>	2109
<u>dbSetPhotonicPinFigAngle</u>	2110
<u>dbSetPhotonicPinFigRadius</u>	2111
<u>dbSetPhotonicPinFigWidth</u>	2112
<u>dbUnsetPhotonicPinFigAngle</u>	2113
<u>dbUnsetPhotonicPinFigRadius</u>	2114
<u>dbUnsetPhotonicPinFigWidth</u>	2115
<u>Photonics Flow Functions</u>	2116
<u>phoAddWaveguidePorts</u>	2117
<u>phoCompose</u>	2121
<u>phoComputeModeProps</u>	2123
<u>phoHiCompose</u>	2125
<u>phoCreatePort</u>	2127
<u>phoGetConnectorAnchors</u>	2130
<u>phoGenWaveguide</u>	2132
<u>phoHiDecompose</u>	2135
<u>phoHiEditCompositeWG</u>	2137
<u>phoIsWaveguide</u>	2138
<u>phoTechDeleteModePropDataTable</u>	2140
<u>phoTechGetModePropDataTableParamValues</u>	2141
<u>phoTechHasModeProp</u>	2143
<u>phoTechHasModePropDataTable</u>	2145
<u>phoTechIsModePropLocalOnly</u>	2147
<u>phoTechLoadModePropDataTable</u>	2149
<u>phoRegisterInstTermOpticalMonitorMenuItem</u>	2152
<u>phoOMGetObject</u>	2153
<u>Photonics Symbol Generator Functions</u>	2155
<u>Symbol Pcell Example</u>	2156
<u>phoDrawBendConnectorSymbol</u>	2157
<u>phoDrawCWLaserSymbol</u>	2159
<u>phoDrawCompWaveguideSymbol</u>	2161
<u>phoDrawCrossBBSymbol</u>	2163
<u>phoDrawCurveDeltaConnectorSymbol</u>	2166
<u>phoDrawDCSymbol</u>	2168

<u>phoDrawEdgeCouplerBBSymbol</u>	2171
<u>phoDrawMMISymbol</u>	2173
<u>phoDrawPhotoDiodeBBSymbol</u>	2176
<u>phoDrawSBendWaveguideSymbol</u>	2178
<u>phoDrawStraightModulatorSymbol</u>	2180
<u>phoDrawStraightWaveguideSymbol</u>	2183
<u>phoDrawWayPointConnectorSymbol</u>	2186
<u>phoDrawYSplitterSymbol</u>	2188
<u>Photonics Connector Functions</u>	2190
<u>phoBendConnector</u>	2191
<u>phoCurveDeltaConnector</u>	2198
<u>phoDrawCompWaveguideLayout</u>	2208
<u>phoSineDeltaConnector</u>	2214
<u>phoStraightConnector</u>	2221
<u>phoWayPointConnector</u>	2227

18

<u>Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions</u>	2237
<u>pgssDRC</u>	2238
<u>pvsApplyDRC</u>	2241

19

<u>Attribute Retrieval and Modification Functions</u>	2245
<u>dbGet Functions</u>	2248
<u>dbGetq, dbGet</u>	2249
<u>dbSet Functions</u>	2253
<u>dbSetq, dbSet</u>	2254
<u>Generic Attributes</u>	2256
<u>Generic Object Attributes</u>	2257
<u>Generic Figure Attributes</u>	2258

Virtuoso Studio Design Environment SKILL Reference

<u>Generic Shape Attributes</u>	2259
<u>Generic anyInst Attributes</u>	2260
<u>Generic Boundary Attributes</u>	2261
<u>Generic Blockage Attributes</u>	2262
<u>Generic Via Attributes</u>	2264
<u>Generic Via Header Attributes</u>	2267
<u>Generic widthSpacingPattern Attributes</u>	2269
<u>Generic widthSpacingPatternGroup Attributes</u>	2270
<u>Generic relatedSnapPattern Attributes</u>	2271
<u>Generic widthSpacingSnapPatternDef Attributes</u>	2272
<u>Generic rowRegion Attributes</u>	2273
<u>Cellview Attributes</u>	2275
<u>Instance Attributes</u>	2280
<u>Instance Header Attributes</u>	2283
<u>Mosaic Attributes</u>	2284
<u>Instance Terminal Attributes</u>	2286
<u>Pin Attributes</u>	2288
<u>Terminal Attributes</u>	2290
<u>Bus Net Definition Attributes</u>	2293
<u>Bus Terminal Definition Attributes</u>	2294
<u>Vector Instance Definition Attributes</u>	2295
<u>Net Attributes</u>	2296
<u>Signal Attributes</u>	2300
<u>Layer-Purpose Pair Attributes</u>	2302
<u>Layer Header Attributes</u>	2303
<u>Group Attributes</u>	2304
<u>Group Member Attributes</u>	2305
<u>Property Attributes</u>	2306
<u>Attributes of Objects and Figures</u>	2308
<u>Area Blockage Attributes</u>	2309
<u>Area Halo Attributes</u>	2309
<u>Layer Blockage Attributes</u>	2310
<u>Layer Halo Attributes</u>	2312
<u>Area Boundary Attributes</u>	2313
<u>Cluster Boundary Attributes</u>	2313
<u>prBoundary Attributes</u>	2314

Virtuoso Studio Design Environment SKILL Reference

<u>Snap Boundary Attributes</u>	2315
<u>Cluster Attributes</u>	2315
<u>figGroup Attributes</u>	2317
<u>GCellPatterns Attributes</u>	2319
<u>Guide Attributes</u>	2320
<u>Marker Attributes</u>	2321
<u>Route Attributes</u>	2323
<u>Row Attributes</u>	2326
<u>Row Header Attributes</u>	2327
<u>Steiner Attributes</u>	2327
<u>Track Pattern Attributes</u>	2329
<u>Custom Via Attributes</u>	2330
<u>Standard Via Attributes</u>	2331
<u>Custom Via Header Attributes</u>	2336
<u>Standard Via Header Attributes</u>	2336
<u>Custom viaVariant Attributes</u>	2336
<u>Attributes</u>	2338
<u>Arc Attributes</u>	2339
<u>Donut Attributes</u>	2340
<u>Dot Attributes</u>	2341
<u>Ellipse Attributes</u>	2341
<u>Label Attributes</u>	2342
<u>Line Attributes</u>	2345
<u>Path Attributes</u>	2346
<u>pathSeg Attributes</u>	2347
<u>Polygon Attributes</u>	2349
<u>Rectangle Attributes</u>	2350
<u>Text Display Attributes</u>	2350
<u>Technology File Attributes</u>	2353
<u>Generic Technology File Object Attributes</u>	2353
<u>Technology File Attributes</u>	2353
<u>Layer Attributes</u>	2354
<u>Display Attributes</u>	2355

Virtuoso Studio Design Environment SKILL Reference

<u>Color Attributes</u>	2356
<u>Stipple Attributes</u>	2357
<u>Line Style Attributes</u>	2358
<u>Two-Layer Attributes</u>	2359
<u>Via Spec Attributes</u>	2363
<u>SiteDef Attributes</u>	2364

20

Design Management Functions..... 2367

<u>ddAmUsingDM</u>	2369
<u>ddAutoCheckin</u>	2370
<u>ddAutoCheckout</u>	2372
<u>ddAutoCtlGetVars</u>	2375
<u>ddAutoCtlSetVars</u>	2376
<u>ddCatRefresh</u>	2378
<u>ddCheckin</u>	2379
<u>ddCheckout</u>	2380
<u>ddClearLibUnmanaged</u>	2381
<u>ddCreateLib</u>	2382
<u>ddDeleteObj</u>	2384
<u>ddDeleteLocal</u>	2386
<u>ddDumpLibList</u>	2387
<u>ddGetCoManagedSet</u>	2388
<u>ddGetCombineValue</u>	2389
<u>ddGetDisplayValue</u>	2390
<u>ddGetForcedLib</u>	2391
<u>ddGetForcedLibEx</u>	2392
<u>ddGetHiddenCellsFuncList</u>	2394
<u>ddGetLibList</u>	2395
<u>ddsGetLibManLogFileName</u>	2396
<u>ddGetObj</u>	2398
<u>ddGetObjAccess</u>	2405
<u>ddGetObjAccessEx</u>	2406
<u>ddGetObjChildren</u>	2408
<u>ddGetObjChildrenEx</u>	2409

Virtuoso Studio Design Environment SKILL Reference

<u>ddGetObjDMSys</u>	2410
<u>ddGetObjFiles</u>	2411
<u>ddGetObjFilesEx</u>	2412
<u>ddGetObjGroup</u>	2413
<u>ddGetObjGroupEx</u>	2414
<u>ddGetObjLastModify</u>	2415
<u>ddGetObjLastModifyEx</u>	2416
<u>ddGetObjLib</u>	2417
<u>ddGetObjName</u>	2418
<u>ddGetObjOwner</u>	2419
<u>ddGetObjOwnerEx</u>	2420
<u>ddGetObjParent</u>	2421
<u>ddGetObjReadPath</u>	2422
<u>ddGetObjReadPathEx</u>	2423
<u>ddGetObjSize</u>	2424
<u>ddGetObjSizeEx</u>	2425
<u>ddGetObjType</u>	2426
<u>ddGetObjWritePath</u>	2427
<u>ddGetObjWritePathEx</u>	2428
<u>ddGetPathDMSys</u>	2429
<u>ddGetStartup</u>	2430
<u>ddGetTmpdirWarn</u>	2432
<u>ddGetTriggerList</u>	2434
<u>ddGetUpdatedLib</u>	2435
<u>ddlslid</u>	2436
<u>ddlshiddenCell</u>	2437
<u>ddlsobjLibMarked</u>	2438
<u>ddlsobjReadable</u>	2439
<u>ddlsobjReadableEx</u>	2440
<u>ddlsobjWritable</u>	2441
<u>ddlsobjWritableEx</u>	2443
<u>ddlspathManageable</u>	2444
<u>ddlspathManaged</u>	2445

Virtuoso Studio Design Environment SKILL Reference

<u>ddLibIs5X</u>	2446
<u>ddLibHasTmpDir</u>	2447
<u>ddLockDescribe</u>	2448
<u>ddLockFree</u>	2449
<u>ddLockGetStatus</u>	2450
<u>ddLockPath</u>	2452
<u>ddLockSet</u>	2453
<u>ddLockSetEx</u>	2455
<u>ddMapGetDataTypeName</u>	2457
<u>ddMapGetDataList</u>	2458
<u>ddMapGetFileData</u>	2459
<u>ddMapGetFileTool</u>	2460
<u>ddMapGetFileViewType</u>	2461
<u>ddMapGetToolNameData</u>	2462
<u>ddMapGetToolNameList</u>	2463
<u>ddMapGetToolNameViewType</u>	2464
<u>ddMapGetViewTypeFileName</u>	2465
<u>ddMapGetViewTypeList</u>	2466
<u>ddMapIsFileNameKnown</u>	2467
<u>ddMapIsToolLegal</u>	2468
<u>ddMapIsToolNameKnown</u>	2469
<u>ddMarkObjMaster</u>	2470
<u>ddNeedCheckout</u>	2471
<u>ddObjHasNamedObj</u>	2472
<u>ddObjHasNamedObjEx</u>	2473
<u>ddObjIsInMaster</u>	2475
<u>ddObjIsInTmp</u>	2476
<u>ddProduceObjRelNames</u>	2477
<u>ddProduceObjRelNamesEx</u>	2479
<u>ddRegHiddenCellsFunc</u>	2481
<u>ddRegTrigger</u>	2483
<u>ddReleaseObj</u>	2486
<u>ddSetForcedLib</u>	2487
<u>ddSetForcedLibEx</u>	2488
<u>ddSetLibUnmanaged</u>	2490
<u>ddSetLibDmType</u>	2491

Virtuoso Studio Design Environment SKILL Reference

<u>ddSetTmpdirWarn</u>	2492
<u>ddUnregHiddenCellsFunc</u>	2493
<u>ddUnregTrigger</u>	2494
<u>ddUpdateLibList</u>	2495
<u>Design Data Services</u>	2497
<u>Design Data Services Environment Variables</u>	2497
<u>Library Functions</u>	2499
<u>Library Synchronization Functions</u>	2499
<u>Customer Library Browser Integration</u>	2499
<u>CIW Menu Customization</u>	2501
<u>ciwAddTranslatorItem</u>	2502
<u>ciwCreateMenu</u>	2503
<u>ciwCreatePulldown</u>	2504
<u>ciwHiExit</u>	2505
<u>ciwMenuInit</u>	2506
<u>ciwMenuInstall</u>	2507
<u>ciwMenuLoadFile</u>	2508
<u>ciwRemoveTranslatorItem</u>	2509
<u>ddsAboutDFII</u>	2510
<u>ddsBrowserAction</u>	2511
<u>ddsCheckinPreferences</u>	2513
<u>ddsCheckoutPreferences</u>	2514
<u>ddsCheckSessionName</u>	2515
<u>ddsCloseLibManager</u>	2516
<u>ddsEditProp</u>	2517
<u>ddsEndSyncWithForm</u>	2518
<u>ddsFileBrowseCB</u>	2519
<u>ddsGetCurrentLib</u>	2522
<u>ddsGetCustomLibSelect</u>	2523
<u>ddsGetServiceName</u>	2524
<u>ddsHiCloseData</u>	2525
<u>ddsHiCreateLibrary</u>	2526
<u>ddsHiMakeReadOnly</u>	2527

Virtuoso Studio Design Environment SKILL Reference

<u>ddsHiOpenCdsLibEditor</u>	2528
<u>ddsHiPreferences</u>	2529
<u>ddsHiRefresh</u>	2530
<u>ddsLEWhatsNewAtStartup</u>	2531
<u>ddsOnLineHelp</u>	2532
<u>ddsOpenConvertToolBox</u>	2533
<u>ddsOpenLibManager</u>	2534
<u>ddsProgramName</u>	2535
<u>ddsRefresh</u>	2536
<u>ddsRegPreRefreshTrigger</u>	2538
<u>ddsRegPostRefreshTrigger</u>	2540
<u>ddsRegCustomLibSelect</u>	2542
<u>ddsHiSaveData</u>	2544
<u>ddsServSelect</u>	2545
<u>ddsSetLibManLCV</u>	2546
<u>ddsGetLibManLCV</u>	2547
<u>ddsSyncWithForm</u>	2548
<u>ddsUnregCustomLibSelect</u>	2554
<u>ddsUpdateSyncWithForm</u>	2555
<u>ddsViewDDProp</u>	2556
<u>ddsWhatsNew</u>	2557
<u>Custom Library Browser Template</u>	2558
<u>dmb Compatibility Functions</u>	2566
<u>dmbGetCurrentLib</u>	2567
<u>dmbLibBrowserFormDone</u>	2568
<u>dmbOpenLibBrowser</u>	2570
<u>dmbSetBrowserInteractions</u>	2571
<u>dmbSyncWithForm</u>	2573
<u>dmbSyncWithCellForm</u>	2576
<u>dmbUpdateBrowser</u>	2579
<u>Cell Category Functions (ddCat Interface)</u>	2581
<u>ddCatOpen</u>	2583
<u>ddCatOpenEx</u>	2585
<u>ddCatAddItem</u>	2587
<u>ddCatSubItem</u>	2588
<u>ddCatSave</u>	2589

Virtuoso Studio Design Environment SKILL Reference

<u>ddCatRemove</u>	2590
<u>ddCatClose</u>	2591
<u>ddCatGetCatName</u>	2592
<u>ddCatGetCatMembers</u>	2593
<u>ddCatGetLibCats</u>	2594
<u>ddCatFindCat</u>	2595
<u>ddCatGetCellsNotInCat</u>	2596
<u>ddCatsObjInCat</u>	2597
<u>Old Cell Category Functions</u>	2598
<u>dmAddCellToCategory</u>	2599
<u>dmCreateCellCategory</u>	2600
<u>dmRemoveCellCategory</u>	2601
<u>dmRemoveCellFromCategory</u>	2602
<u>Design Editor Functions</u>	2603
<u>Registering and Unregistering Applications</u>	2603
<u>deAddAppCapabilities</u>	2604
<u>deRegApp</u>	2605
<u>deRemoveAppCapabilities</u>	2610
<u>deUnRegApp</u>	2611
<u>deRegUserTriggers</u>	2612
<u>deUnRegUserTriggers</u>	2614
<u>deInheritEnabling</u>	2615
<u>deUnInheritEnabling</u>	2616
<u>deInheritMenus</u>	2617
<u>deUnInheritMenus</u>	2618
<u>deRegProbeTrigger</u>	2619
<u>deUnRegProbeTrigger</u>	2620
<u>deGetProbeTrigger</u>	2621
<u>deGetAppInfo</u>	2622
<u>deHiSetDefaultApplication</u>	2625
<u>deSetWorkspace</u>	2626
<u>deRegPlugin</u>	2627
<u>deInstallPlugin</u>	2630

Virtuoso Studio Design Environment SKILL Reference

<u>deRemovePlugin</u>	2631
<u>deFindPlugin</u>	2632
<u>deBeginConfigurePlugins</u>	2633
<u>deEndConfigurePlugins</u>	2635
<u>deFindAssistant</u>	2636
<u>deGetPlugins</u>	2638
<u>deGetAppClassInfo</u>	2639
<u>deoEnableTrigger</u>	2640
<u>deoExitTrigger</u>	2642
<u>deoGetAssistantList</u>	2643
<u>deoGetBindkeys</u>	2644
<u>deoMenuTrigger</u>	2645
<u>deoOnTabSwitch</u>	2647
<u>deoOnApplicationChange</u>	2648
<u>deoPostInstallTrigger</u>	2649
<u>deoRequireToolbars</u>	2651
<u>deoUpdateToolbar</u>	2652
<u>deoOnEvent</u>	2654
<u>Client Interface</u>	2655
<u>deFileNew</u>	2656
<u>deOpen</u>	2657
<u>deFileOpen</u>	2665
<u>deNew</u>	2666
<u>deNewCellView</u>	2669
<u>deOpenCellView</u>	2671
<u>deBack</u>	2673
<u>deForward</u>	2674
<u>dePush</u>	2675
<u>deSetHistoryOpenMode</u>	2677
<u>deSwap</u>	2678
<u>deToggleAssistants</u>	2680
<u>deToggleAssistantVisibility</u>	2681
<u>deToggleToolbars</u>	2682
<u>deFindToolbar</u>	2683
<u>deManageToolbar</u>	2685
<u>deCustomizeToolbars</u>	2686

Virtuoso Studio Design Environment SKILL Reference

<u>deUnmanageToolbar</u>	2687
<u>deEditInPlace</u>	2688
<u>deReturn</u>	2690
<u>deCopy</u>	2691
<u>Trigger Functions</u>	2693
<u>dataTrigger</u>	2696
<u>appTrigger</u>	2698
<u>userAppTrigger</u>	2702
<u>enableTrigger</u>	2703
<u>menuTrigger</u>	2704
<u>userMenuTrigger</u>	2705
<u>postInstallTrigger</u>	2706
<u>userPostInstallTrigger</u>	2707
<u>probeTrigger</u>	2708
<u>Licensing Functions</u>	2710
<u>lbuiGetCheckedOutLicenses</u>	2711
<u>lbuiPrintLicenseUseReport</u>	2712
<u>iclicCheckOutAndLock</u>	2714
<u>eliCheckOutAndLock</u>	2715
<u>Design Editor Support Routines</u>	2717
<u>deGetCellView</u>	2719
<u>deGetEditCellView</u>	2720
<u>deSetCellView</u>	2721
<u>deGetViewType</u>	2722
<u>deGetEditViewType</u>	2723
<u>deGetRootType</u>	2724
<u>deGetEditRootType</u>	2725
<u>deGetContext</u>	2726
<u>deGetEditContext</u>	2727
<u>delsEditInContext</u>	2728
<u>deSetContext</u>	2729
<u>deGetMaxInContextLevel</u>	2730
<u>deGetConfigId</u>	2731

Virtuoso Studio Design Environment SKILL Reference

<u>deSetConfigId</u>	2732
<u>deGetVector</u>	2733
<u>deGetEIP</u>	2734
<u>deGetPushed</u>	2735
<u>deGetProbeContext</u>	2736
<u>deSetProbeContext</u>	2737
<u>deGetEditLevel</u>	2738
<u>deGetMaxEditLevel</u>	2739
<u>deEnableApp</u>	2740
<u>deDisableApp</u>	2741
<u>deInstallApp</u>	2742
<u>deGetAllApplicationNames</u>	2743
<u>deGetAllRootTypes</u>	2744
<u>deGetAllViewTypes</u>	2745
<u>deGetRootViewTypeList</u>	2746
<u>deMapUserTypeToViewType</u>	2747
<u>deMapViewTypeToUserType</u>	2748
<u>deViewTypesRegistered</u>	2749
<u>deGetLowerTierApp</u>	2750
<u>Extended Design Editor Example</u>	2751

Graphics Editor Functions

This topic provides a list of Cadence® SKILL function categories associated with 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.

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

- Design Functions
- Automatic Menu Enabling Functions
- Cross Section Viewer Functions
- Edit and Display Functions
- Refresh Functions
- View-Level Functions
- Procedural-Level Functions
- Window Scroll Functions
- Selection Functions
- Stored Selection Functions
- Highlight Functions
- Background Cellview Functions
- Marker Functions
- Probe Functions
- Icon Creation and Loading Functions

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

- Utility Functions
- Microwave Object FunctionsMouse Functions

Design Functions

A design can be opened for reading or editing. 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* makes 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.

You can use the following SKILL functions to work with your designs:

geChangeCellView

geChangeEditMode

geDiscardEdits

geNewWindow

geReturn

geReturnToLevel

geSave

geSaveAs

geSaveHier

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 dialog box 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

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

Value Returned

<i>w_windowId</i>	Returns the database ID of the window passed in.

Examples

Returns `win2`.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

geOpen

Design Functions

geChangeEditMode

```
geChangeEditMode (
    t_newMode { r | a }
    [ w_windowId ]
)
=> t / nil
```

Description

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

Arguments

<i>t_newMode</i>	New access mode. The valid values are: <i>r</i> (read) and <i>a</i> (edit)
<i>w_windowId</i>	Database ID of the window containing the design you want to change. If not specified, current window is used.

Value Returned

<i>t</i>	The mode is changed.
<i>nil</i>	The mode is not changed.

Examples

Changes the mode in *win3* to read.

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

Related Topics

[Design Functions](#)

geDiscardEdits

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

Description

Discards all edits made to a design.

Arguments

<i>w_windowId</i>	Database ID of the window for which 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.

Examples

Discards the edits in win3.

```
geDiscardEdits( win3 )
```

Related Topics

[Design Functions](#)

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 independent of the old window.

Arguments

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

Value Returned

<i>w_windowId</i>	A copy of the specified window is created.
<i>nil</i>	The specified window is not copied.

Examples

Copies `win3`; returns `win4` if it is successful.

```
geNewWindow( win3 )
```

Related Topics

[Design Functions](#)

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. This function does not support figGroups.

Arguments

<i>w_windowId</i>	Database ID of the window containing the cellview.

Value Returned

<i>t</i>	The display returns to the previous level.
<i>nil</i>	The display does not return to the previous level.

Examples

Returns to the previous level in the current window.

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

Related Topics

[Design Functions](#)

geReturnToLevel

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

Description

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

Arguments

<code>w_windowId</code>	Database ID of the window containing the design.

Value Returned

<code>nil</code>	Always returns <code>nil</code> .

Examples

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

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

Related Topics

[Design Functions](#)

geSave

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

Description

Saves a design.

Arguments

<i>w_windowId</i>	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.

Examples

Saves the design in win2.

```
geSave( win2 )
```

Related Topics

[Design Functions](#)

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 an invalid 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

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

Value Returned

<i>t</i>	The cellview is saved.
<i>nil</i>	The cellview is not saved.

Examples

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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

Related Topics

Design Functions

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. The `geSaveHier` function works only on maskLayout views.

Arguments

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

Value Returned

<code>t</code>	The design is saved hierarchically.

Examples

Saves the design hierarchically up to depth 32 and prompts to specify the name of the cellview to be saved.

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

Related Topics

[Design Functions](#)

Automatic Menu Enabling 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.

Related Topics

[Defining a Menu Update Policy](#)

[geGetEnablePredicate](#)

[geGetWindowStatus](#)

[geSetEnablePredicate](#)

[geSetWindowStatus](#)

[geUpdateWindowMenus](#)

Defining a Menu Update Policy

To define a menu update policy in your application, you need to do the following:

1. Decide what status flags your application needs.

For example, the Layout editor defines the following flags:

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

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)
```

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

Related Topics

[geSetEnablePredicate](#)

[geSetWindowStatus](#)

[geGetWindowStatus](#)

[Automatic Menu Enabling Functions](#)

geGetDisabledMenus

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

Description

Given a *windowId*, returns all the inactive submenus for all banner menus on that window. If *menuList* is specified, only the inactive 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 inactive submenus of the banner menu, or *nil* if the corresponding banner menu does not contain any inactive submenus. If a slider-menu item is inactive, the slider-menu submenus are omitted from the sublist. Each sublist is sorted alphabetically by symbol name.

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

Arguments

<i>w_windowId</i>	The ID of the window containing the inactive submenus to be listed.
<i>l_menuList</i>	A list of the symbolic names of banner menus or the symbolic name of one banner menu. If specified, only the inactive submenus of the menus in <i>l_menuList</i> are returned.

Value Returned

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

Examples

Returns all the inactive submenus for all banner menus on the window.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
windowId = window(2)
=> window:2
deGetRootType(windowId)
=> "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 that 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.

Related Topics

[geUpdateWindowMenus](#)

[Automatic Menu Enabling Functions](#)

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

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

Value Returned

<code>n_enablePredicate</code>	The enable predicate of the <code>menu</code> for the application <code>appName</code> .
<code>nil</code>	Undefined

Examples

Returns the enable predicate of the menu for the application.

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

Related Topics

[Automatic Menu Enabling Functions](#)

geGetWindowStatus

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

Description

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

Arguments

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

Value Returned

<i>n_status</i>	The status for application named <i>appName</i> on window <i>windowId</i> .
<i>nil</i>	The status for <i>appName</i> is undefined.

Examples

Returns the status for the application.

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

Related Topics

[Automatic Menu Enabling Functions](#)

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

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

Value Returned

None

Examples

Sets the enable predicate of the *menu*.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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

Related Topics

[Automatic Menu Enabling Functions](#)

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

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

Value Returned

<i>n_previousStatus</i>	Previous status for <i>appName</i> on <i>w_windowId</i> or <code>~0</code> if the previous status was undefined.

Examples

Sets the status of the window and the application.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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

Related Topics

[Automatic Menu Enabling Functions](#)

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`.

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

Arguments

<i>w_windowId</i>	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`.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

[Automatic Menu Enabling Functions](#)

Cross Section Viewer Functions

(Layout MXL Only) You can view cross sections of physical layers in a layout by using the cross section viewer. The viewer displays the different layers, from top to bottom, present in a cross section, enabling you to view each layer clearly. This is helpful to identify any missing or wrongly placed layers and to debug connection issues across layers.

You can use the following SKILL functions to work with the cross section viewer. Using these functions you can invoke the cross section viewer, load or unload the configuration file used for customizing the viewer, and specify the settings used for displaying the cross section of a layout.

- geCrossSection
- geCrossSectionAddLayerThickness
- geCrossSectionAddLayerToSkip
- geCrossSectionAddLayersToMerge
- geCrossSectionAddMaterialToSkip
- geCrossSectionClearConfig
- geCrossSectionLoadConfig

geCrossSection

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

Description

(Layout MXL Only) Invokes the *Cross Section* command in the specified window. The cross section viewer is displayed and a cutline appears on the canvas. The cross section of the area under the cutline is displayed in the viewer.

Arguments

<i>w_windowId</i>	ID of the window in which you want to view the cross section of a layout. If you do not specify the window ID, the current window is used.
<i>l_point</i>	A point on the canvas where you want the cutline to appear. If you do not specify a point and your mouse pointer is not on the canvas, the cutline appears at the center of the canvas. If the pointer is on the canvas, the cutline appears where the pointer is. You can specify a point only if you have specified a window ID.

Value Returned

<i>t</i>	The cross section viewer is displayed and the cutline appears on the canvas.
<i>nil</i>	The command failed.

Example

Displays the cross section of the layout at point 0 : 0 in the current window.

```
geCrossSection(hiGetCurrentWindow() 0:0)
```

Related Topics

[Cross Section Viewer](#)

[Cross Section Viewer Configuration File](#)

[Viewing the Cross Section of a Layout](#)

[Cross Section Viewer Functions](#)

geCrossSectionAddLayerThickness

```
geCrossSectionAddLayerThickness(' (
    l_layerName
    l_thickness
))
=> t / nil
```

Description

(Layout MXL Only) Specifies the thickness to be used for displaying the given layers in the cross section viewer. By default, all layers are displayed using the same thickness. This SKILL function is used in a loading environment; you cannot run it from the CIW. You can define this function in the configuration file that you want to use for customizing the cross section viewer. The function is run when you load the configuration file by using [geCrossSectionLoadConfig](#).

Arguments

<i>l_layerName</i>	Name of the layer for which you want to define the thickness.
<i>l_thickness</i>	Thickness of the layer to be displayed in the cross section viewer.

Value Returned

<i>t</i>	The thickness to be used for displaying the specified layers in the cross section viewer is added to the configuration file.
<i>nil</i>	The command failed.

Example

Adds the specified thickness for displaying `Metal1`, `Metal2`, and `Metal3` in the cross section viewer.

```
geCrossSectionAddLayerThickness(' (
( "Metal1" "0.1" )
( "Metal2" "0.1" )
( "Metal3" "0.2" )
))
```

Related Topics

[Cross Section Viewer](#)

[Cross Section Viewer Configuration File](#)

[Specifying Cross Section Viewer Settings](#)

[Cross Section Viewer Functions](#)

geCrossSectionAddLayerToSkip

```
geCrossSectionAddLayerToSkip(' (
    l_layerName
))
=> t / nil
```

Description

(Layout MXL Only) Specifies the layers that should not be displayed in the cross section viewer.

This SKILL function is used in a loading environment; you cannot run it from the CIW. You can define this function in the configuration file that you want to use for customizing the cross section viewer. The function is run when you load the configuration file by using [geCrossSectionLoadConfig](#).

Arguments

<i>l_layerName</i>	Layer to be skipped from the cross section viewer.

Value Returned

<i>t</i>	The layers to be skipped from the cross section viewer are added to the configuration file.
<i>nil</i>	The command failed.

Example

Specifies that the `PWdummy` and `Metal5` layers should not be displayed in the cross section viewer.

```
geCrossSectionAddLayerToSkip(' (
    ("PWdummy")
    ("Metal5")
))
```

Related Topics

[Cross Section Viewer](#)

[Cross Section Viewer Configuration File](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Specifying Cross Section Viewer Settings

Cross Section Viewer Functions

geCrossSectionAddLayersToMerge

```
geCrossSectionAddLayersToMerge(' (
    l_layerName
))
=> t / nil
```

Description

(Layout MXL Only) Specifies the layers that should be displayed on the same line in the cross section viewer.

This SKILL function is used in a loading environment; you cannot run it from the CIW. You can define this function in the configuration file that you want to use for customizing the cross section viewer. The function is run when you load the configuration file by using [geCrossSectionLoadConfig](#).

Arguments

<i>l_layerName</i>	Layers to be displayed on the same line in the cross section viewer.

Value Returned

<i>t</i>	The layers to be displayed on the same line in the cross section viewer are added to the configuration file.
<i>nil</i>	The command failed.

Example

Specifies that the Nwell, Oxide, and "Nimp" layers should be displayed on the same line in the cross section viewer.

```
geCrossSectionAddLayersToMerge(' (
    ("Nwell" "Oxide" "Nimp")
))
```

Related Topics

[Cross Section Viewer](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

[Cross Section Viewer Configuration File](#)

[Specifying Cross Section Viewer Settings](#)

[Cross Section Viewer Functions](#)

geCrossSectionAddMaterialToSkip

```
geCrossSectionAddMaterialToSkip(' (
    l_materialName
))
=> t / nil
```

Description

(Layout MXL Only) Specifies the material to be skipped. Layers made of the specified material are not displayed in the cross section viewer.

This SKILL function is used in a loading environment; you cannot run it from the CIW. You can define this function in the configuration file that you want to use for customizing the cross section viewer. The function is run when you load the configuration file by using [geCrossSectionLoadConfig](#).

Arguments

<i>l_materialName</i>	Name of the material to be skipped from the cross section viewer.

Value Returned

<i>t</i>	The name of the material to be skipped is added to the configuration file.
<i>nil</i>	The command failed.

Example

Specifies that the layers made of the following material should not be displayed in the cross section viewer: *nplus*, *pplus*, *recognition*, and *passivationCut*.

```
geCrossSectionAddMaterialToSkip(' (
    ("nplus")
    ("pplus")
    ("recognition")
    ("passivationCut")
))
```

Related Topics

[Cross Section Viewer](#)

[Cross Section Viewer Configuration File](#)

[Specifying Cross Section Viewer Settings](#)

[Cross Section Viewer Functions](#)

geCrossSectionClearConfig

```
geCrossSectionClearConfig(  
    t_techLibName  
)  
=> t / nil
```

Description

(Layout MXL Only) Unloads the configuration file from the memory so that it is not used for displaying the cross section of a layout.

Arguments

<i>t_techLibName</i>	Technology library name associated with the design.

Value Returned

t	The configuration file is unloaded from the memory.
nil	The command failed.

Example

Clears the configuration file associated with the technology library `gpdk045`, from the memory.

```
geCrossSectionClearConfig("gpdk045")
```

Clears the configuration file associated with the current technology library from the memory.

```
geCrossSectionClearConfig(techGetTechLibName(techGetTechFile(gec())))
```

Related Topics

[Cross Section Viewer](#)

[Cross Section Viewer Configuration File](#)

[Specifying Cross Section Viewer Settings](#)

[Cross Section Viewer Functions](#)

geCrossSectionLoadConfig

```
geCrossSectionLoadConfig(  
    t_techLibName  
    t_configFileName  
)  
=> t / nil
```

Description

(Layout MXL Only) Loads the specified configuration file for displaying the cross section of a layout in the cross section viewer.

Arguments

<i>t_techLibName</i>	Name of the technology library associated with your design.
<i>t_configFileName</i>	Name of the configuration file used for customizing the cross section viewer.

Value Returned

<i>t</i>	The configuration file is loaded.
<i>nil</i>	The command failed.

Example

Loads the `crossConfig.il` configuration file associated with the technology library of the current design.

```
geCrossSectionLoadConfig(techGetTechLibName(techGetTechFile(gec())))  
".crossConfig.il")
```

Loads the `crossConfig.il` configuration file associated with the technology library `gpdk045`.

```
geCrossSectionLoadConfig("gpdk045" "./crossConfig.il")
```

Related Topics

[Cross Section Viewer](#)

[Cross Section Viewer Configuration File](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Specifying Cross Section Viewer Settings

Cross Section Viewer Functions

Edit and 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 edit and display functions help you perform tasks such as get a cellview, determine instance hierarchy, and assign a cellview to a particular window.

Related Topics

[geClearIgnoreProp](#)

[geComputeQueryBbox](#)

[geCopySelSet](#)

[geDeleteSelSet](#)

[geMenuForObjectsUnderPoint](#)

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

<code>l_inst l_net</code>	List of instances or list of nets

Value Returned

<code>l_obj</code>	Returns a list of objects passed to the function.

Examples

The following example clears the property on the selected instances.

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

The following example clear the property on all the nets in the current cellview.

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

Related Topics

[Edit and Display Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

gec

```
gec (  
    )  
=> d_cellview
```

Description

Returns the ID for the cellview being edited. This function is a shortcut for the function, `geGetEditCellView`.

Arguments

None

Value Returned

<i>d_cellview</i>	Database ID of the cellview being edited in the window.
<i>nil</i>	The cellview is not found or an error occurred.

Examples

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

```
gec ()
```

Related Topics

[Edit and Display Functions](#)

[geGetEditCellView](#)

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

<i>d_cellviewID</i>	ID of the cellview for which the net name filter settings are cleared.

Value Returned

<i>t</i>	The net name filter settings were cleared for the specified cellview ID.
<i>nil</i>	The command failed.

Examples

Clears the filter applied to hide nets VDD and VSS.

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

Related Topics

[Edit and Display Functions](#)

geComputeQueryBbox

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

Description

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 `selectionAperture`.

If the window is a layout window, $\text{delta} = \text{selectionAperture}/2$.

If the window is a schematic window, $\text{delta} = \text{selectionAperture}$.

If `w_windowID` is not specified, the current window ID is used.

Arguments

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

Value Returned

<code>t</code>	Returns bounding box coordinates.
<code>nil</code>	Unable to process bounding box.

Related Topics

[Edit and Display Functions](#)

geCopySelSet

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

Description

Procedural interface to copy the selected set.

Arguments

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

Value Returned

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

Examples

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

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

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Edit and Display Functions

geDeleteSelSet

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

Description

Deletes all selected objects in the specified window.

Arguments

<i>w_windowId</i>	Window ID of the window to use. If not specified, the current window is used.

Value Returned

<i>t</i>	The selected objects are deleted.
<i>nil</i>	The selected objects are not deleted.

Related Topics

[Edit and Display Functions](#)

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

<i>t_type</i>	Optional argument that describes the type of environment variables.
	<code>all</code>
	<code>boolean</code>
	<code>integer</code>
	<code>float</code>
	<code>string</code>
	<code>list</code>
	Default Value: <code>all</code>

Value Returned

<i>t</i>	Lists of available variables.

Examples

The following examples list the window environment variables that can be set.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Edit and Display Functions

geEditSelSet

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

Description

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

Arguments

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

Value Returned

<i>t</i>	The selected objects are found.
<i>nil</i>	The selected objects are not found.

Examples

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

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

Related Topics

[Edit and Display Functions](#)

geEditSingleObject

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

Description

Displays and edits the properties of an object.

Arguments

<i>d_objectId</i>	Database ID of the object to edit.

Value Returned

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

Examples

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

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

Related Topics

[Edit and Display Functions](#)

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*.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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`.

Examples

The following example specifies that 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

Related Topics

Edit and Display Functions

`geWindowToEditPoint`

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

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

Value Returned

<i>t</i>	Net name label display for shapes in the specified window is enabled.
<i>nil</i>	Net name label display is disabled.

Examples

The following example enables the net name label display for shapes in the specified window ID.

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

The following example disables the net name label display for shapes in the specified window ID.

```
geEnableNetNameDisplay(window nil)
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

[Net Name Display](#)

[Net Name Display Options Form](#)

[Edit and Display Functions](#)

geEquivWindow

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

Description

Returns the ID of the 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

<i>w_windowId</i>	Database ID of the window whose equivalent you want to find. Defaults to the current window.

Value Returned

<i>w_windowEquiv</i>	The window with the same edit cellview as <i>w_windowId</i> if it exists.
<i>nil</i>	The window does not exist.

Examples

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

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

Returns `window:3`.

Related Topics

[Edit and Display Functions](#)

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

<i>w_windowId</i>	Database ID of the window whose equivalent you want to find. Defaults to the current window.

Value Returned

<i>l_windowList</i>	List of windows with the same edit cellview as <i>w_windowId</i> .

Examples

Returns (window:2 window:3), assuming that window(2) and window(3) both are editing the same cellview.

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

Related Topics

[Edit and Display Functions](#)

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

<i>d_cellview</i>	Database ID of the cellview.
<i>d_masterId</i>	Database ID of the master.

Value Returned

<i>l_hierPath</i>	The path to the master cellview.
<i>nil</i>	The cellview is not found.

Related Topics

[Edit and Display Functions](#)

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

<i>d_cellview</i>	Database ID of the cellview.

Value Returned

<i>w_windowId</i>	The database ID of the window if the cellview is displayed.
<i>nil</i>	The cellview is not displayed.

Examples

Returns *win3*.

```
geGetCellViewWindow( cell3 )
```

Related Topics

Edit and Display Functions

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`.

You can also use the function `gec`, which is a shortcut for the function `geGetEditCellView`.

Arguments

<code>w_windowId</code>	Database ID of the window of the cellview being edited.

Value Returned

<code>d_cellview</code>	Database ID of the cellview being edited in the window.

Examples

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

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

Related Topics

[Edit and Display Functions](#)

[geGetWindowCellView](#)

geGetEditCellViewWindow

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

Description

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

Arguments

<i>d_cellview</i>	Database ID of the cellview.

Value Returned

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

Related Topics

Edit and Display Functions

geGetEditRep

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

Description

Returns the ID for the cellview being edited. It is an alias for `geGetEditCellView`.

Arguments

<i>w_windowId</i>	Database ID of the window of the cellview being edited.

Value Returned

<i>d_cellview</i>	Database ID of the cellview being edited in the window.

Related Topics

Edit and Display Functions

`geGetEditCellView`

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

<i>w_windowId</i>	Database ID of the window from which the hierarchy information is wanted.

Value Returned

<i>l_list</i>	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. For example, if you descend into iteration number 4, it always means I1<4> regardless of the order of the iterated instance.
<i>nil</i>	The current window is empty or if the current window is undefined.

Examples

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

The example returns the following:

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

db:01234567 0
db:45678900 1

Related Topics

Edit and Display Functions

geGetHierMemInst

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

<i>w_windowId</i>	Database ID of the window from which the hierarchy information is wanted.

Value Returned

<i>t_instance-hierarchy</i>	Instance hierarchy description for this window.
<i>nil</i>	The current window is empty or the current window is undefined

Examples

The following example returns `/db:13240512/db:13405278`.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Edit and Display Functions

geGetInstHierPath

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

<i>w_windowId</i>	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.

Examples

The following example returns `((db:13240512 0 0 0) (db:13402578 1 4 2))`.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

[Edit and Display Functions](#)

[geGetInstHier](#)

geGetNetNameDisplayFilter

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

Description

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

Arguments

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

Value Returned

<i>l_filterSettings</i>	The net name filtering settings.
<i>nil</i>	The net name filtering settings were not found.

Examples

Returns the net name filtering settings currently in use.

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

Related Topics

[Edit and Display Functions](#)

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

<i>d_cellview</i>	Database ID of the top-level cellview in the window.

Value Returned

<i>l_selectedSet</i>	A list of lists describing objects selected in the hierarchy of the given cellview.

Examples

Select a pin on an instance in a schematic.

```
geGetObjectSelectedSet ()
((db:3865864
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

((db:38370404 0 0 0))))

Related Topics

Edit and Display Functions

geGetSortedSelectByLine

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

Description

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

Arguments

<i>w_windowId</i>	Database ID of the cellview.
<i>l_linePoints</i>	Database ID of the master.

Value Returned

<i>l_sortedObjects</i>	List of sorted objects.

Examples

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

```
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) )
```

Related Topics

[Edit and Display Functions](#)

geGetTopCellViewWindow

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

Description

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

You also use the function `gtc`, which is a shortcut for the function, `geGetTopCellViewWindow`.

Arguments

<i>d_cellview</i>	Database ID of the cellview.

Value Returned

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

Related Topics

[Edit and Display Functions](#)

[geGetWindowBox](#)

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

<code>w_windowId</code>	Database ID of the window in which the cellview is at the top level.

Value Returned

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

Examples

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

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

Related Topics

[Edit and Display Functions](#)

geGetWindowBox

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

Description

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

Arguments

<i>w_windowId</i>	Database ID of the window.

Value Returned

<i>l_bBox</i>	The coordinates are found.
<i>nil</i>	The coordinates are not found.

Examples

Returns (0:0 1200:1000).
`geGetWindowBox( window(2) )`

Related Topics

[Edit and Display Functions](#)

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. If you know what window you are interested in, a shorthand equivalent to this function can be used:

`window(n)->displayCellView.`

You also use the function `gwc`, which is a shortcut for the function, `geGetWindowCellView`.

Arguments

<code>w_windowId</code>	Database ID of window displaying the cellview whose object identifier you want.

Value Returned

<code>d_cellview</code>	Database ID of the object displayed in the window.
<code>nil</code>	The current window is empty or no current window is defined.

Examples

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

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

Related Topics

[Edit and Display Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

geGetEditCellView

gwc

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. If you know what window you are interested in, a shorthand equivalent to this function can be used:

`window(n)->displayCellView.`

Arguments

<i>w_windowId</i>	Database ID of window displaying the cellview whose object identifier you want.

Value Returned

<i>d_cellview</i>	Database ID of the object displayed in the window.
<i>nil</i>	The current window is empty or if no current window is defined.

Related Topics

[Edit and Display Functions](#)

[geGetWindowCellView](#)

[geGetEditCellView](#)

geIsGraphicalWindow

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

Description

Tests if the specified window, or by default the current window, is a graphical window.

Arguments

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

Value Returned

<i>t</i>	Testing is set successfully.
<i>nil</i>	Returns <i>nil</i> when unable to set or determine window.

Examples

Tests if the window is a graphical window.

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

Related Topics

[Edit and Display Functions](#)

gelsNetNameDisplayActiveOnWindow

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

Description

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

Arguments

<i>w_windowID</i>	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. The Net Name Display functionality being available does not mean that the functionality is enabled. Use gelsNetNameDisplayEnabled to check if the functionality is enabled for the specified window.
nil	The Net Name Display functionality is not available for the specified window.

Examples

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

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

Related Topics

[Net Name Display](#)

[Edit and Display Functions](#)

geIsNetNameDisplayEnabled

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

Description

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

Arguments

<i>w_windowID</i>	Window ID for which the state of the Net Name Display functionality is evaluated.

Value Returned

<i>t</i>	The Net Name Display functionality is enabled for the given window ID.
<i>nil</i>	The Net Name Display functionality is not enabled for the given window ID.

Examples

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

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

Related Topics

[Net Name Display](#)

[Edit and Display Functions](#)

geMenuForObjectsUnderPoint

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

Description

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

Arguments

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

Value Returned

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

Related Topics

Edit and Display Functions

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

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

Value Returned

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

Related Topics

[Net Name Display](#)

[Net Name Display Options](#)

[Edit and Display Functions](#)

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 `geSaveNetNameDisplayFilter`, the settings will be reloaded automatically when you open the cellview in a new Virtuoso session.

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

Arguments

<code>d_cellview</code>	Database ID of the cellview.

Value Returned

<code>t</code>	The net name filtering is saved.
<code>nil</code>	The net name filtering is not saved.

Examples

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

Related Topics

[Edit and Display Functions](#)

[geSetNetNameDisplayFilter](#)

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, you select three objects, A, B, and C. Then, calling the `geSelectNextInSelSet()` function will select A. When you next call the function, object B is selected, and so on.

Arguments

<i>d_cellviewId</i>	(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

<i>t</i>	The object is selected.
<i>nil</i>	The object is not selected.

Related Topics

[Edit and Display Functions](#)

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 `geSelectPreviousInSelSet()` function will select A. Calling the function again will select B. If you now call the function, object A will be selected.

Arguments

<code>d_cellviewId</code>	(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

<code>t</code>	The object is selected.
<code>nil</code>	The object is not selected.

Related Topics

[Edit and Display Functions](#)

[geSelectPreviousInSelSet](#)

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

<i>w_windowID</i>	(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

<i>t</i>	The objects in the window are successfully toggled.
<i>nil</i>	The objects in the window could not be toggled.

Related Topics

[Edit and Display Functions](#)

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

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

<i>t</i>	The net name filtering is set.
<i>nil</i>	The net name filtering is not set.

Examples

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

Related Topics

Edit and Display Functions

geSetTCProbeDisplayStyle

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

Description

Sets the *trueColorProbeVal* and *probeHighlightVal* options of the specified *probeId*.

Arguments

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

Value Returned

t	The specified values have been successfully set to the specified <i>probeId</i> .
nil	The specified values have not been successfully set to the specified <i>probeId</i> .

Examples

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

```
"drawing7" ) ?probeType "net")
```

```
geSetTCProbeDisplayStyle(myProbe t t)
```

Related Topics

Edit and Display Functions

geTransparencyOptions

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

Description

Opens the Transparency Options form.

Arguments

None

Value Returned

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

Examples

Opens the Transparency Options form.

```
geTransparencyOptions()
```

Related Topics

[Edit and Display Functions](#)

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*.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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).

Examples

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.

Related Topics

[Edit and Display Functions](#)

[geEditToWindowPoint](#)

gtc

```
gtc(  
    )  
=> d_cellview / nil
```

Description

Returns the ID of the window in which the specified cellview is the top cellview. This function is a shortcut for the function `geGetTopCellViewWindow`.

Arguments

None

Value Returned

<code>d_cellview</code>	Database ID of the window in which the specified cellview is the top cellview.
<code>nil</code>	A top cellview window was not found or an error occurred.

Examples

Returns `db:123456`, in which the specified cellview is the top cellview.

```
gtc()
```

Related Topics

[Edit and Display Functions](#)

[geGetTopCellViewWindow](#)

gwc

```
gwc (  
    )  
    => d_cellview / nil
```

Description

Returns the ID of the cellview displayed in a window. This function is a shortcut of the function `geGetWindowCellView`.

Arguments

None

Value Returned

<i>d_cellview</i>	Database ID of the object displayed in the window.
<i>nil</i>	The current window is empty or no current window is defined.

Examples

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

```
gwc ()
```

Related Topics

[Edit and Display Functions](#)

[geGetWindowCellView](#)

Refresh Functions

You can use refresh functions to redraw a cellview, a window, or all areas of all windows. You can also refresh the worldView window.

Related Topics

[geRefresh](#)

[geRefreshCellView](#)

[geRefreshCellViews](#)

[geRefreshWindow](#)

[geRefreshWorldView](#)

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.

Examples

Redraws all areas of all windows that need updating.

```
geRefresh( )
```

Related Topics

[Refresh Functions](#)

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

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

Value Returned

<i>t</i>	Cellview or area is successfully refreshed.
<i>nil</i>	Cellview or area is not refreshed.

Related Topics

[Refresh Functions](#)

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

<i>d_cvId</i>	Cellview ID of the cellview to be redrawn.

Value Returned

<i>t</i>	Cellview is successfully refreshed.
<i>nil</i>	Cellview is not refreshed.

Examples

Redraws `cell1`.

```
geRefreshCellViews ( cell1 )
```

Related Topics

[Refresh Functions](#)

geRefreshWindow

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

Description

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

Arguments

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

Value Returned

<i>t</i>	The window is refreshed successfully.
<i>nil</i>	The window is not refreshed successfully.

Examples

Redraws win2.

```
geRefreshWindow( win2 )
```

Related Topics

[Refresh Functions](#)

geRefreshWorldView

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

Description

Refreshes the worldView window.

Arguments

<i>w_windowId</i>	The worldView window ID.

Value Returned

<i>t</i>	The window is refreshed.
<i>nil</i>	The window is not refreshed.

Related Topics

[Refresh Functions](#)

View-Level Functions

You can use view-level functions to remove area and instance-specific view levels from a window. You can retrieve area view levels and the start level and stop level of an instance or those assigned within a window. You can also perform tasks such as set the display start and stop threshold levels for an area or instance.

Related Topics

[geDeleteAllAreaViewLevel](#)

[geDeleteAllInstViewLevel](#)

[geDeleteAreaViewLevel](#)

[geDeleteInstViewLevel](#)

[geGetAllAreaViewLevel](#)

[geGetAllInstViewLevel](#)

[geGetAreaViewLevel](#)

[geGetInstViewLevel](#)

[geSetAreaViewLevel](#)

[geSetInstViewLevel](#)

geDeleteAllAreaViewLevel

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

Description

Removes all area view levels from a window.

Arguments

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

Value Returned

<i>t</i>	Area view levels were deleted.
<i>nil</i>	No area view levels were deleted.

Examples

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

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

Related Topics

[View-Level Functions](#)

geDeleteAllInstViewLevel

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

Description

Removes all instance-specific view levels from a window.

Arguments

<i>w_windowId</i>	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.

Examples

Deletes all the instance view levels in the current window.

```
geDeleteAllInstViewLevel( )
```

Related Topics

[View-Level Functions](#)

geDeleteAreaViewLevel

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

Description

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

Arguments

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

Value Returned

<i>t</i>	An area view level is deleted.
<i>nil</i>	No area view level is deleted.

Examples

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

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

Related Topics

[View-Level Functions](#)

geDeleteInstViewLevel

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

Description

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

Arguments

<i>w_windowId</i>	Database ID of the window from which you want the instance view levels removed. If <i>w_windowId</i> is not specified, the current window is used.
<i>d_instId</i>	Database ID of the instance that has its view levels removed.

Value Returned

<i>t</i>	A view level was found and deleted.
<i>nil</i>	A view level was not found and deleted.

Examples

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

```
geDeleteInstViewLevel( instance4 )
```

Related Topics

[View-Level Functions](#)

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

<i>w_windowId</i>	Database ID of the window for which you want the view levels.

Value Returned

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

Examples

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

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

Related Topics

[View-Level Functions](#)

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

<code>w_windowId</code>	Database ID of the window where you want the view levels.

Value Returned

<code>l_instStartStop</code>	If the instances in <code>w_windowId</code> 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.
<code>nil</code>	There are no view levels available.

Examples

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

```
geGetAllInstViewLevel( )
```

Related Topics

[View-Level Functions](#)

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

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

Value Returned

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

Examples

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

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

Related Topics

View-Level Functions

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

<i>w_windowId</i>	Database ID of the window where you want the view levels for <i>d_instId</i> .
<i>d_instId</i>	Database ID of the instance for which you want the view levels.

Value Returned

<i>l_startStop</i>	If <i>d_instId</i> has view levels associated with it in <i>w_windowId</i> , then a list with the start level and stop level is returned.
<code>nil</code>	<i>d_instId</i> has no view levels associated with it in <i>w_windowId</i> .

Examples

Returns (0 20) assuming that the instance has start level 0 and stop level 20.

```
geGetInstViewLevel( )
```

Related Topics

[View-Level Functions](#)

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

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

Examples

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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

Related Topics

[View-Level Functions](#)

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

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

Examples

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

```
geSetInstViewLevel(  
    hiGetCurrentWindow()
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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

Related Topics

[View-Level Functions](#)

geRestoreWinView

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

Description

Restores the view saved with `geSaveWinView`.

Arguments

<code>w_windowId</code>	Database ID of the window to restore.

Value Returned

<code>t</code>	The window is restored.
<code>nil</code>	The window is not restored.

Examples

Restores the window `newwin`.

```
geRestoreWinView( newwin )
```

Related Topics

[geSaveWinView](#)

[View-Level Functions](#)

geSaveWinView

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

Description

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

Arguments

<i>w_windowId</i>	Database ID of the window containing the view to save. If <i>w_windowId</i> is not specified, the current window is used.
<i>t_name</i>	Name of the saved window view. If <i>t_name</i> 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.

Examples

Saves win3 to newwin.

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

Related Topics

[geRestoreWinView](#)

[View-Level Functions](#)

Procedural-Level Functions

You can use procedure-level functions to edit an instance in place and to push or switch into an instance.

Related Topics

[geEditInPlace](#)

[gePush](#)

[geSwitch](#)

[geSwitchInContext](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

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

Value Returned

<i>t</i>	The instance is edited in place.
<i>nil</i>	The instance is not edited in place.

Examples

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

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

Related Topics

[Procedural-Level Functions](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

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

Value Returned

<i>t</i>	The instance is pushed into.
<i>nil</i>	The instance is not pushed into.

Examples

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.

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

Related Topics

[Procedural-Level Functions](#)

geSwitch

```
geSwitch(  
    w_windowId  
    t_accessMode  
    d_instId  
    x_iteration  
    x_row  
    x_col  
)  
=> 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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>w_windowId</i>	Database ID of the window in which to do the switch.
<i>t_accessMode</i>	Access mode used to open the instance master. Valid Values: <i>r</i> (read) or <i>a</i> (append)
<i>d_instId</i>	Database ID of the instance to switch into.
<i>x_iteration</i>	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
<i>x_row</i>	Row of the arrayed instance.
<i>x_col</i>	Column of the arrayed instance.

Value Returned

<i>w_windowId</i>	The database ID of the window that is switched into.
<i>nil</i>	The instance is not switched into.

Examples

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

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

Related Topics

[Procedural-Level Functions](#)

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

<i>w_windowId</i>	Database ID of the window in which to do the switch.
<i>t_accessMode</i>	Access mode used to open the instance master. Valid Values: <i>r</i> (read) and <i>a</i> (append)
<i>d_instId</i>	Database ID of the instance to switch into.
<i>x_iteration</i>	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>
<i>x_row</i>	Row of the arrayed instance.
<i>x_col</i>	Column of the arrayed instance.
<i>verbose</i>	This optional argument displays a warning message when something is incorrect.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Value Returned

<code>w_windowId</code>	The database ID of the window that is switched into.
<code>nil</code>	The instance is not switched into.

Examples

Switches into the instance, `instId`, by using the view name specified in the hierarchy manager configuration context.

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

Related Topics

[Procedural-Level Functions](#)

Window Scroll Functions

You can use the window scrolling functions to scroll a window view, set and obtain the scroll percentage, and shift a window in the specified direction. You can also zoom into a subject up to the last point at which the major grid is visible.

Related Topics

[geCenter](#)

[geGetScrollPercent](#)

[geScroll](#)

[geSetScrollPercent](#)

[geShift](#)

[geZoomToGrid](#)

geCenter

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

Description

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

Arguments

<i>w_windowId</i>	Database ID of the window in which to center the design. If <i>w_windowId</i> is not specified or <code>nil</code> , the current window is used.

Value Returned

<code>t</code>	The display is panned.
<code>nil</code>	The display is not panned.

Examples

Centers the design in `window(3)`.

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

Related Topics

[Window Scroll Functions](#)

geGetScrollPercent

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

Description

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

Arguments

None

Value Returned

<i>x_percent</i>	Percentage of the display to scroll.

Examples

Returns the scroll percentage.

```
geGetScrollPercent ( )
```

Related Topics

[geScroll](#)

[Window Scroll Functions](#)

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

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

Value Returned

<i>t</i>	The display is panned.
<i>nil</i>	The display is not panned.

Examples

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

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

Related Topics

[geSetScrollPercent](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Window Scroll Functions

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

<i>n_percent</i>	New percentage of the display to scroll.

Value Returned

<i>n_percent</i>	New percentage of the display to scroll.

Examples

Returns 25 and sets the new percentage to 25 percent.

```
geSetScrollPercent( 25 )
```

Related Topics

[geScroll](#)

[Window Scroll Functions](#)

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

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

Value Returned

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

Examples

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

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

Related Topics

[geScroll](#)

[Window Scroll Functions](#)

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

<i>w_windowId</i>	Database ID of the window that displays the subject.

Value Returned

<i>t</i>	The subject is displayed successfully.
<i>nil</i>	The subject is not displayed successfully.

Related Topics

[Window Scroll Functions](#)

Selection Functions

A group of objects selected together, either by point selection or area selection, is referred to as a selected set. By using selection functions, you can do the following:

- 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).

There are four categories of selection functions, as describes in the following table. The table lists the functions available at the SKILL level for creating, maintaining, and accessing selected sets.

Type of Selection Functions	Description
Point Selection Functions	Implement the interactive selection by point. These functions can be interactive if the point information is not passed as an argument. ■ <u>geAddSelectPoint</u> ■ <u>geSingleSelectPoint</u> ■ <u>geSubSelectPoint</u>
Area Selection Functions	Implement the functions for interactive selection by area. These functions can be interactive if the area bounding box information is not passed as an argument. ■ <u>geAddSelectBox</u> ■ <u>geSingleSelectBox</u> ■ <u>geSubSelectBox</u>

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Type of Selection Functions	Description
Figure Selection 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. <u>geSelectFig</u> <u>geSelectFigPoint</u> <u>geSelectAllFig</u> <u>geDeselectAll</u> <u>geDeselectAllFig</u> <u>geDeselectAllObject</u> <u>geDeselectArea</u> <u>geDeselectFig</u> <u>geDeselectFigNoFilter</u> <u>geDeselectFigPoint</u> <u>geDeselectObject</u> <u>geDeselectObjectNoFilter</u> <u>geDeselectObjectPoint</u> <u>geDeselectPoint</u> <u>gelsFigAllSelected</u> <u>gelsFigPartiallySelected</u> <u>gelsFigSelected</u> <u>geGetSelSet</u> <u>geGetSelectedSet</u> <u>geGetSelSetCount</u>

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Type of Selection Functions

Description

Selection Queries Functions

Allow you to query the contents of the selected set.

- [gelsFigAllSelected](#)
 - [gelsFigPartiallySelected](#)
 - [gelsFigSelected](#)
 - [gePointQuery](#)
-

Related Topics

[Point Selection](#)

[Area Selection](#)

[Selection User Interface](#)

Point Selection

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

- **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 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.

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.

Related Topics

[Area Selection](#)

[Selection User Interface](#)

[Selection Functions](#)

Area 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.

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.

Related Topics

[Selection User Interface](#)

[Point Selection](#)

[Selection Functions](#)

Selection 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 modes of selection are supported using modifier keys. The following table describes the standard selection mouse bindings.

Modifier	Left Mouse Click	Left Mouse Drag
None	Single-Select Point	Single-Select Box
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`.

Related Topics

[Selection Functions](#)

[Point Selection](#)

[Area Selection](#)

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

<code>w_windowId</code>	Database ID of the window.

Value Returned

<code>t</code>	The selection is performed.
<code>nil</code>	The initial conditions are not correct.

Related Topics

[Selection Functions](#)

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

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

Value Returned

<code>t</code>	The selection is performed.
<code>nil</code>	The selection is not performed.

Examples

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

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

[partialSelect](#)

[Object Stretching](#)

[Selection Functions](#)

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

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

Value Returned

<code>t</code>	The selection is performed.
<code>nil</code>	The selection is not performed.

Examples

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

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

Related Topics

[partialSelect](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Selection Functions

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

<i>l_point</i>	The point to be adjusted.
<i>w_windowId</i>	Database ID of the window.

Value Returned

<i>l_point</i>	The point that has been adjusted.
<i>nil</i>	The point was not adjusted.

Related Topics

[Selection Functions](#)

geCyclePotentialSelectNext

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

Description

Cycles the potential selection to the next object.

Arguments

<i>w_windowID</i>	Optional argument that specifies the window to be used. The current window is used if <i>w_windowID</i> is not specified.
<i>l_point</i>	Optional argument that specifies the point to be potentially selected. The current point is used if <i>l_point</i> is not specified.

Value Returned

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

Examples

```
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)
```

Related Topics

Selection Functions

geCycleSelectNext

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

Description

Cycles to the next object at the current selection point.

Arguments

<i>w_windowID</i>	Optional argument that specifies the window to be used.

Value Returned

<i>t</i>	Next object is selected.
<i>nil</i>	No other objects are selected.

Related Topics

[Selection Functions](#)

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

<i>w_windowId</i>	Database ID of the window containing the object.

Value Returned

t	Everything is deselected.
nil	Everything is not deselected.

Related Topics

[Selection Functions](#)

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

<i>d_cellview</i>	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.

Related Topics

[Selection Functions](#)

geDeselectAllObject

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

Description

Deselects everything in the specified cellview.

Arguments

<i>d_cellview</i>	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.

Related Topics

[Selection Functions](#)

geDeselectArea

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

Description

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

Arguments

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

Value Returned

<i>t</i>	The objects within the bounding box were deselected.
<i>nil</i>	The objects were not deselected.

Examples

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

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

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

geSubSelectBox

geDeselectFig

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

Description

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

Arguments

<i>d_figId</i>	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 deselected.

Examples

Deselects fig3.

```
geDeselectFig( fig3 )
```

Related Topics

[Selection Functions](#)

geDeselectFigs

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

Description

Deselects figures from a specified list of figures.

Arguments

<i>l_figList</i>	List of figure IDs.

Value Returned

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

Examples

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

Related Topics

[Selection Functions](#)

geDeselectFigNoFilter

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

Description

Deselects the specified figure.

Arguments

<i>d_figId</i>	The figure to be deselected.

Value Returned

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

Examples

Deselects figId.

```
geDeselectFigNoFilter( figId )
```

Related Topics

[Selection Functions](#)

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

<i>d_figId</i>	Database ID of the object you want to select.
<i>l_vertex</i>	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 <i>l_vertex</i> than in the object, they are applied to the first vertices in the object and the remaining vertices are unaffected. Valid Values: <i>t</i> , <i>nil</i>

Value Returned

<i>t</i>	The object passes the selection filter and the vertices are deselected.
<i>nil</i>	The function failed.

Examples

Deselects every other vertex of *fig3*.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

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

<i>d_figId</i>	The figure to be deselected.
<i>l_path</i>	Hierarchical path to <i>d_figId</i>, in the following format: <pre>list(list(instId memInst x y) list(instId memInst x y) . . .)</pre> <i>x</i> and <i>y</i> are integers defining the row and column in case <i>instId</i> is a mosaic.

Value Returned

<i>t</i>	The specified figures are deselected.
<i>nil</i>	The specified figures are not deselected.

Examples

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

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

Related Topics

[Selection Functions](#)

geDeselectObjectNoFilter

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

Description

Deselects the specified object at the specified hierarchical path.

Arguments

<i>d_figId</i>	The figure to be deselected.
<i>l_path</i>	Hierarchical path to <i>d_figId</i> , in the following format: <pre>list(list(instId memInst x y) list(instId memInst x y) . . .)</pre> <i>x</i> and <i>y</i> are integers defining the row and column in case <i>instId</i> is a mosaic.

Value Returned

<i>t</i>	The specified figure is deselected.
<i>nil</i>	The specified figure is not deselected.

Examples

Deselects the seventh member of the specified *instId*.

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

Related Topics

Selection Functions

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

<i>d_figId</i>	The figure to be deselected.
<i>l_path</i>	Hierarchical path to <i>d_figId</i> , in the following format: <pre>list(list(<i>instId</i> <i>memInst</i> <i>x</i> <i>y</i>) list(<i>instId</i> <i>memInst</i> <i>x</i> <i>y</i>) . . .)</pre> <i>x</i> and <i>y</i> are integers defining the row and column in case <i>instId</i> is a mosaic.
<i>l_points</i>	The list of the points to be deselected.

Value Returned

<i>t</i>	The specified vertices of the specified figure are deselected.
<i>nil</i>	The specified vertices of the specified figure are not deselected.

Examples

Deselects the object at x 1 y 2.

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


Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

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

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

Value Returned

<i>t</i>	The figure is deselected.
<i>nil</i>	The figure is not deselected.

Examples

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

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

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

partialSelect

geSubSelectPoint

geDeselectProtected

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

Description

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

Arguments

<i>d_cellview</i>	The cellview in which the protected objects are to be deselected.

Value Returned

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

Examples

Deselects the protected objects in the edit cellview.

```
geDeselectProtected()
```

Related Topics

[Selection Functions](#)

geDSAGetSelectedSet

```
geDSAGetSelectedSet(  
    [ ?window w_windowId ]  
    [ ?objTypesFilter l_objTypesFilter ]  
)  
=> list / nil
```

Description

Returns the hierarchical selected set from the Dynamic Selection assistant.

Arguments

?window w_windowId The ID of the window from which the objects are selected.

?objTypesFilter l_objTypesFilter

A list of selected object types.

Value Returned

list Hierarchical selected set from the Dynamic Selection assistant.

nil The command as unsuccessful or the command failed.

Examples

The following example with no parameters, returns all selected items from the Dynamic Selection assistant opened in the current window.

```
geDSAGetSelectedSet()  
=> (((("SMALL" "dsaGetItems" "layout")  
      ("M1(1,0)" "SMALL" "low" "layout")  
      ("ILow" "SMALL" "low2" "layout")  
      ("ILow2" "SMALL" "low3" "layout")  
      )  
    (rect  
      (db:0x2c8ddb1a)  
      )  
    )  
  (((("SMALL" "dsaGetItems" "layout")  
      ("M1(1,0)" "SMALL" "low" "layout")  
      ("ILow" "SMALL" "low2" "layout")  
      )  
    (inst  
      ("ILow2")  
      )  
  )
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
)  
((( "SMALL" "dsaGetItems" "layout" ))  
(mosaic  
  ("M1")  
)  
)  
)
```

The following example with parameters `window(2)` and `list('inst)`, returns the selected instance items from the Dynamic Selection assistant opened in `window(2)`.

```
geDSAGetSelectedSet(window(2) list('inst))  
=> (((("SMALL" "dsaGetItems" "layout")  
      ("M1(1,0)" "SMALL" "low" "layout")  
      ("ILow" "SMALL" "low2" "layout")  
      )  
     (inst  
      ("ILow2")  
      )  
     )  
     )  
     )
```

Related Topics

[Dynamic Selection Assistant](#)

geDSAFreeze

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

Description

Toggles between the freeze and unfreeze states of the Dynamic Selection Assistant (DSA).

Arguments

<i>w_windowId</i>	Database ID of the window for which the DSA is required to be frozen or unfrozen.
<i>l_point</i>	Position of the cursor (x and y coordinates) at which the SKILL function is called.

Value Returned

<i>t</i>	The DSA toggled between freeze and unfreeze state successfully.
<i>nil</i>	The DSA did not toggle between freeze and unfreeze state successfully.

Examples

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.

```
geDSAFreeze(window(2) 8.0:17.0)
```

Related Topics

[Selection Functions](#)

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

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

Value Returned

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

Examples

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


Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

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

<i>w_windowId</i>	Window ID of the window to use.
<i>d_cellview</i>	The cellview ID.

Value Returned

<i>l_selectedSet</i>	A list of the selected objects.
<i>nil</i>	The function failed.

Examples

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

```
geGetSelSet()
```

Related Topics

[Selection Functions](#)

geGetSelectedSet

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

Description

Returns a list of selected objects in a cellview.

Arguments

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

Value Returned

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

Related Topics

[Selection Functions](#)

geGetSelSetCount

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

Description

Counts selected objects in a cellview.

Arguments

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

Value Returned

<i>n_count</i>	The number of objects in the selected set.
0	No objects are in the selected set.

Related Topics

[Selection Functions](#)

geGetSelectedInstancesCount

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

Description

Counts selected instances in a cellview.

Arguments

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

Value Returned

<i>n_count</i>	The number of objects in the selected set.

Examples

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

Related Topics

[Selection Functions](#)

geGetSelProtectedObjects

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

Description

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

Arguments

<i>d_cellview</i>	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

<i>l_results</i>	Returns a list of protected objects.
<i>nil</i>	No list is returned.

Examples

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

```
geGetSelProtectedObjects()
```

Related Topics

[Selection Functions](#)

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.

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

Arguments

<i>d_figId</i>	Database ID of the object you want to select.

Value Returned

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

Related Topics

[Selection Functions](#)

geGetWindowSelProtectionHighlight

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

Description

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

Arguments

<i>w_window</i>	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

<i>t</i>	Returned if the protected objects in the specified window are highlighted.
<i>nil</i>	Returned if the protected objects in the specified window are not highlighted.

Examples

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

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

Related Topics

[Selection Functions](#)

geGetWindowSelProtectionSelectability

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

Description

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

Arguments

<i>w_window</i>	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

<i>t</i>	Returned if the protected objects in the specified window can be selected.
<i>nil</i>	Returned if the protected objects in the specified window cannot be selected.

Examples

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

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

Related Topics

[Selection Functions](#)

geHiDragFig

```
geHiDragFig(  
    w_windowID  
    r_form  
    t_srcPrompt  
    t_dstPrompt  
    t_cmdName  
    l_selSet  
    g_useMultiWindow  
    [ l_firstPoint ]  
)  
=> l_results
```

Description

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

Arguments

<i>w_windowID</i>	Window ID.
<i>r_form</i>	Form data structure. This works best with an options form created using <code>hiCreateAppForm</code> or <code>hiCreateLayoutForm</code> , with <code>?formType</code> as 'options'. You can toggle the appearance of the options form using the F3 key. The form is closed when the <code>geHiDragFig</code> function completes. Valid Values: Any valid form of data structure or 0 if there is no form.
<i>t_srcPrompt</i>	Prompt string for the origin or source point.
<i>t_dstPrompt</i>	Prompt string for the destination point.
<i>t_cmdName</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

<i>l_selSet</i>	Specifies which selected set or object's image is to be dragged. Its value could be a selected set, a selected object, or <code>nil</code>. 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). Valid Values: <code>geGetSelSet</code> or any valid selected set or <code>nil</code>. If the argument is <code>nil</code>, a dragging image does not appear on the screen; otherwise, a dragging image appears and can be moved with the mouse. You can use <code>geGetSelSet</code> to get the value of the selected set or object. It returns <code>nil</code> if nothing is selected.
<i>g_useMultiWindow</i>	Supports multiple windows. If you set this argument to <code>nil</code>, you cannot use snap mode. If you set this argument to <code>t</code>, the snap mode is only respected in the source window. Valid Values: <code>t</code> (supports multiple windows), <code>nil</code> (does not support multiple windows).
<i>l_firstPoint</i>	First source point specified as a list.

Value Returned

<i>l_results</i>	Returns the value in the following format: <pre>((source_point_CV source_point_coordination) (destination_point_C V destination_point_coordination))</pre>

Examples

The following example shows `geHiDragFig` used along with `geGetSelSet` to implement the dragging of database figures:

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

The example returns the following value:

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

The following example shows the use of an options form:

```
let((selSet result)
  hiCreateAppForm(
    ?name 'ExDragForm
    ?formType 'options
    ?buttonLayout 'HideCancel
    ?formTitle "Drag Options"
    ?fields list(
      list(
        hiCreateBooleanButton(?name 'deleteOrig ?buttonText "Delete Original")
        0:0 400:30 100
      )
    )
  )
  selSet = geGetSelSet()
  result = geHiDragFig(
    hiGetCurrentWindow()
    ExDragForm
    "Enter Origin Point"
    "Enter Destination Point"
    "Copy"
    selSet
    t
  )
  printf("Delete: %L\nResult: %L\n"
    ExDragForm->deleteOrig->value
    result
  )
)
```

The example returns the following result:

```
Delete: nil
Result: ((db:0x2af5f19a (34.44 55.61)) (db:0x2af5f19a (44.745 -13.895)))
```

The following example shows a multi-window scenario. When you use the same code as in the previous example to drag an image to another window, the example returns the following result:

```
Delete: t
Result: ((db:0x2af5f19a (29.575 55.285)) (db:0x2c659a9a (0.3 0.795)))
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

The image that is dragged into another window is just an image, not a real object. You cannot select or save it. The window is not edited.

Related Topics

[Selection Functions](#)

[hiCreateAppForm](#)

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.

Related Topics

[Selection Functions](#)

geHighlightProtected

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

Description

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

Arguments

<i>w_window</i>	The window in which the protected objects are to be highlighted. If no window Id is specified, the protected objects in the current window are highlighted.

Value Returned

<i>t</i>	Protected objects are highlighted.
<i>nil</i>	Protected objects are not highlighted.

Examples

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

```
geHighlightProtected()
```

Related Topics

[Selection Functions](#)

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

Starts the enter function to set selection protection.

```
geHiSetSelProtectionEF()
```

Related Topics

[Selection Functions](#)

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

Starts the enter function to remove selection protection.

```
geHiUnsetSelProtectionEF()
```

Related Topics

[Selection Functions](#)

gelsFigAllSelected

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

Description

Checks whether an object is completely selected.

Arguments

<i>d_figId</i>	Database ID of the object you want to select.

Value Returned

t	<i>d_figId</i> is completely selected.
nil	<i>d_figId</i> is not completely selected.

Related Topics

[Selection Functions](#)

gelsFigPartiallySelected

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

Description

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

Arguments

<i>d_figId</i>	Database ID of the object you want to select.

Value Returned

t	<i>d_figId</i> is partially selected.
nil	<i>d_figId</i> is not partially selected.

Related Topics

[Selection Functions](#)

gelsFigSelected

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

Description

Checks whether an object is at least partially selected.

Arguments

<i>d_figId</i>	Database ID of the object you want to select.

Value Returned

t	Object <i>d_figId</i> is at least partially selected.
nil	Object <i>d_figId</i> is not at least partially selected.

Related Topics

gelsObjectAllSelected

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

Description

Tests if the specified figure is fully selected.

Arguments

<i>d_figId</i>	The figure to be tested.
<i>l_path</i>	Hierarchical path to the <i>d_figId</i> , 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.

Examples

Checks to see if *figId* is selected.

```
geIsObjectAllSelected(figId)
```

Related Topics

Selection Functions

geIsObjectPartiallySelected

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

Description

Tests if the specified figure is partially selected.

Arguments

<i>d_figId</i>	The figure to be tested.
<i>l_path</i>	Hierarchical path to the <i>d_figId</i> , in the following format: <pre>list(list(<i>instId memInst x y</i>) list(<i>instId memInst x y</i>) . . .)</pre> <i>x</i> and <i>y</i> are integers defining the row and column in case <i>instId</i> is a mosaic.

Value Returned

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

Examples

Checks to see if *figId* is partially selected.

```
geIsObjectPartiallySelected(figId)
```

Related Topics

Selection Functions

gelsObjectSelected

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

Description

Tests if the specified figure is partially or fully selected.

Arguments

<i>d_figId</i>	The figure to be tested.
<i>l_path</i>	Hierarchical path to the <i>d_figId</i> , 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.

Examples

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

```
geIsObjectSelected(figId)
```

Related Topics

Selection Functions

geIsPointInsidePolygon

```
geIsPointInsidePolygon(  
    d_ cellViewId  
    l_point  
    l_pointsArray  
    g_edges  
)  
=> t / nil
```

Description

Checks if the specified point is inside a polygon.

Arguments

<i>d_cellViewId</i>	Database ID of the cellview containing the polygon.
<i>l_point</i>	Coordinates of the input point to be checked.
<i>l_pointsArray</i>	Coordinates of the polygon as an array of points.
<i>g_edges</i>	A Boolean value specifying whether the points on the edges of the polygon should be considered. The default value is <i>t</i> .
Valid Values: <i>t</i> or <i>nil</i>	

Value Returned

<i>t</i>	Returned if the polygon array of points <i>l_pointsArray</i> contains the given point <i>l_point</i> .
<i>nil</i>	Returned if the polygon array of points does not contain the given point.

Example

Checks if the polygon described by the coordinates 1:1 1:4 3:6 5:4 5:1 in the current cellview contains the point 3:6. Because the points on the edges of the polygon are not considered, the function returns *nil*.

```
cv=geGetEditCellview()  
geIsPointInsidePolygon(cv 3:6 list(1:1 1:4 3:6 5:4 5:1) nil)  
-> nil
```


Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

[IsPointInsideFig](#)

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

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

Value Returned

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

Examples

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

```
geIsSelProtection(winId figId)
```

Related Topics

Selection Functions

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

<i>w_windowId</i>	Database ID of the window containing the object.
<i>l_point</i>	Location of the point used for the search.
<i>t_filterProc</i>	The name of a user-defined procedure which accepts <i>d_figId</i> , and returns <i>nil</i> if <i>figID</i> is filtered.

Value Returned

<i>d_figId</i>	The ID of the object found in the query.
<i>nil</i>	No object is found.

Examples

Returns the ID of the database object found at the point 1:1, only if the object is a `rect`.

```
procedure(myFilterProc(objId)  
if(objId~>objType=="rect" t nil)  
)  
gePointQuery( win3 1:1 "myFilterProc")
```

Related Topics

Selection Functions

geSingleSelectPoint

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

<i>w_windowId</i>	Database ID of the window containing the objects.
<i>g_ignoreSelect</i>	Specifies whether to ignore objects currently selected when searching for the object. Valid Values: <code>t</code> , <code>nil</code>
<i>l_list</i>	List of X and Y coordinates in user units to specify locations to look for objects.

Value Returned

<i>d_figure</i>	Database ID of the first object satisfying the selection criteria.
<code>nil</code>	No objects satisfy the selection criteria.

Examples

Returns the object IDs of the objects located at the given coordinates.

```
gePointToFig( win2 list(0:0 10:10) )
```

Related Topics

Selection Functions

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

geSingleSelectPoint

geSelectAll

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

Description

Selects all objects in the specified window.

Arguments

<i>w_windowId</i>	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.

Related Topics

[Selection Functions](#)

geSelectAllFig

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

Description

Selects all objects in a cellview that pass the selection filter.

Arguments

<i>d_cellview</i>	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.

Examples

Selects all objects in `cell3`.

```
geSelectAllFig( cell3 )
```

Related Topics

[Selection Functions](#)

geSelectArea

```
geSelectArea(  
    [ w_windowId ]  
    [ l_points ]  
    [ t_areaShapeType ]  
)  
=> t / nil
```

Description

Adds the objects within the area defined by *t_areaShapeType* to the selection.

You can use the left mouse button to select the area in enclosed mode and the middle mouse button to select the area in enclosed and crossed mode.

Arguments

<i>w_windowId</i>	Window ID of the window containing the objects. If not specified, the current window is used.
<i>l_points</i>	List of points that define the area for selection.
<i>t_areaShapeType</i>	The shape of the area to use for selection. Valid Values: <code>rectangle</code> , <code>polygon</code> . The default shape is <code>rectangle</code> .

Value Returned

<i>t</i>	The objects within the specified area are added to the selection.
<code>nil</code>	The objects within the specified area are not added to the selection.

Examples

Adds the objects within the specified polygon to the selection.

```
geSelectArea(hiGetCurrentWindow() list(list(0 0) list(100 100) list(0 100))  
"polygon")
```

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.

Related Topics

[Select Area Shape Type Form](#)

[Selection Functions](#)

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

<code>t</code>	Objects overlapping the line are selected.
<code>nil</code>	Objects overlapping the line are not selected.

Examples

Enables you to draw a line to overlap all objects you want to select.

```
geSelectByLine()
```

Related Topics

[Selection Functions](#)

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

<i>w_windowId</i>	Database ID of the window containing the objects to be selected. If not specified, the current window is used.

Value Returned

<i>t</i>	All objects are selected.
<i>nil</i>	No object is selected.

Examples

```
geSelectBy2PointsLine(window(2))
```

Related Topics

[Selection Functions](#)

geSelectFig

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

Description

Selects an object in the layout.

Arguments

<i>d_figId</i>	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.

Examples

Selects fig35.

```
geSelectFig( fig35 )
```

Related Topics

[Selection Functions](#)

geSelectFigs

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

Description

Selects figures from a specified list of figures.

Arguments

<i>l_figList</i>	List of figure IDs.

Value Returned

<i>l_list</i>	List of selected figures.
<i>nil</i>	An error occurred, the figures are not selected and the list of figures is not returned.

Examples

```
geSelectAllFig()  
sel = geGetSelSet()  
geSelectFigs(sel)
```

Related Topics

[Selection Functions](#)

geSelectFigNoFilter

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

Description

Selects the specified figure regardless of the selection filter.

Arguments

<i>d_figId</i>	The figure to be selected.

Value Returned

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

Examples

Ignores all selection filters and selects `figId`.

```
geSelectFigNoFilter( figId )
```

Related Topics

[Selection Functions](#)

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

<i>d_figId</i>	Database ID of the object you want to select.
<i>l_vertex</i>	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 <i>l_vertex</i> 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.

Examples

Selects every other vertex on fig35.

```
geSelectFigPoint( fig35 list(t nil t nil t nil t nil) )
```


Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

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

<i>d_figId</i>	The database ID of the figure to be selected.
<i>l_path</i>	A path in the format ((<i>d_instId</i> <i>x_memInst</i> <i>x_row</i> <i>x_column</i>) (<i>d_instId</i> <i>x_memInst</i> <i>x</i> <i>y</i>) ...) <i>x_memInst</i> is used for an iterated instance. <i>x</i> and <i>y</i> are used for mosaic instances.

Value Returned

<i>t</i>	Selects the specified figure at the specified hierarchical path, if it passes the selection criteria.
<i>nil</i>	Does not select the specified figure.

Related Topics

[Selection Functions](#)

geSelectObjectNoFilter

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

Description

Selects the specified figure at the specified hierarchical path.

Arguments

<i>d_figId</i>	Database ID of the figure to be selected.
<i>l_path</i>	The optional path in the format ((<i>d_instId</i> <i>x_memInst</i> <i>x_row</i> <i>x_column</i>) (<i>d_instId</i> <i>x_memInst</i> <i>x</i> <i>y</i>) ...) <i>x_memInst</i> is used for an iterated instance. <i>x</i> and <i>y</i> are used for mosaic instances.

Value Returned

<i>t</i>	Selects the specified figure at the specified hierarchical path.
<i>nil</i>	Does not select the specified figure.

Related Topics

[Selection Functions](#)

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

<i>w_windowId</i>	Window ID of the window to use. If not specified, the current window is used.
<i>l_point</i>	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

<i>t</i>	The point is used to select the object.
<i>nil</i>	The point is not used to select the object.

Examples

Selects the object at 70x 70y in the current window.

```
geSelectPoint( hiGetCurrentWindow() list(70 70) )
```

Interactive Function

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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.

Related Topics

Selection Functions

geAddSelectPoint

geSelectProtected

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

Description

Selects the objects that are protected from being selected in the specified cellview.

Arguments

<i>d_cellview</i>	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

Selects all the protected objects in the edit cellview.

```
geSelectProtected()
```

Related Topics

[Selection Functions](#)

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

<i>d_cellview</i>	Optional argument specifies the cellview to process. If not specified, the current cellview is used.

Value Returned

<code>t</code>	An additional via has been added to the selection set.
<code>nil</code>	The selection set does not contain vias.

Examples

Selects vias associated with the via pile of the via selected in the current cellview.

```
geSelectViaPile( )
```

Related Topics

[Selection Functions](#)

geSetEnterFunctionSelectionMode

```
geSetEnterFunctionSelectionMode(  
    w_windowId  
    x_mode  
)  
=> t / nil
```

Description

Sets the selection mode for the enter function associated with the specified window.

Arguments

<i>w_windowId</i>	Window ID.
<i>x_mode</i>	Sets the selection mode as follows: ■ 0 = no select option ■ 1 = select override/full select override ■ 2 = partial select ■ 3 = partial select override ■ 4 = selection highlight

Value Returned

<i>t</i>	Default or if the window ID has an enter function and the mode was set.
<i>nil</i>	The selection mode is not set.

Examples

The following examples set the selection mode to 2 (partial select) for the enter function associated with the current window, (window 6).

```
geSetEnterFunctionSelectionMode(hiGetCurrentWindow() 2)  
geSetEnterFunctionSelectionMode(window(6) 2)
```


Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

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

<code>w_windowId</code>	Database ID of the window containing the objects.

Value Returned

<code>t</code>	The objects in the window are fully selected.
<code>nil</code>	The objects in the window are not fully selected.

Examples

```
geSelObjectsPartiallySelected()
```

Related Topics

[Selection Functions](#)

[geSelObjectsPartiallySelected](#)

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

<i>l_results</i>	List of objects that are set as protected.
<i>nil</i>	The specified object is not set as protected.

Examples

The following example sets selection protection on the *figId1* and *figId2* objects:

```
geSetSelProtection(list(figId1 figId2))
```

The following example sets selection protection on all the selected objects in the *cellViewA* cellview:

```
geSetSelProtection(cellViewA)
```

The following example sets selection protection on all the selected objects in the edit cellview:

```
geSetSelProtection()
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

geSetSelProtectionAllObjects

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

Description

Sets all the objects in the specified cellview as protected, which prevents them from being selected.

Arguments

<i>d_cellview</i>	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

Sets all the objects in the edit cellview as protected.

```
geSetSelProtectionAllObjects()
```

Related Topics

[Selection Functions](#)

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

Opens the Selection Protection Highlight Options form.

```
geSetSelProtectionHighlightOptionForm()
```

Related Topics

[Selection Functions](#)

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

<i>w_window</i>	The window in which the specified highlight options are set on the protected objects.
<i>d_cellview</i>	The cellview in which the protected objects are to be selected.
<i>l_options</i>	List of highlight settings to be used on the protected objects. If no options are specified, the default settings will be applied.

Examples

Sets the following characteristics of the halo on the selection protected objects: *Halo Mode*: fadeout, *Level*: under, *Width*: thin, and *Transparency*: 40.

```
geSetSelProtectionHighlightOptions(windowId cellviewId list("fadeout" "under"  
"thin" 40))
```

Related Topics

[Selection Functions](#)

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

<i>w_windowId</i>	Database ID of the window containing the objects.
<i>g_partial</i>	Whether partial selection should be performed. Valid Values: <code>t</code> , <code>nil</code>
<i>l_point</i>	The bounding box to be used for selection. When no bounding box is specified, this function operates interactively.

Value Returned

<code>t</code>	The objects are selected.
<code>nil</code>	The objects are not selected.

Examples

Selects the objects in `win1` enclosed in the bounding box described by the coordinates 0:0, 25:25. Partial selection is not performed.

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

Related Topics

[partialSelect](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Selection Functions

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

<i>w_windowId</i>	Database ID of the window containing the objects.
<i>g_partial</i>	Whether partial selection should be performed. Valid Values: <code>t</code> , <code>nil</code>
<i>l_point</i>	Point to be used for selection. When no point is specified, this function operates interactively.

Value Returned

<code>t</code>	The objects are selected.
<code>nil</code>	The objects are not selected.

Examples

Selects the object in `win3` located at the coordinate 0:0. Partial selection is not performed.

```
geSingleSelectPoint ( win3 nil (0:0) )
```

Related Topics

[partialSelect](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Selection Functions

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

<i>w_windowId</i>	Window ID of the window containing the objects.
<i>g_partial</i>	Whether partial selection should be performed. Valid Values: <code>t</code> , <code>nil</code>
<i>l_bBox</i>	The bounding box to be used for de-selection. When no bounding box is specified, this function operates interactively.

Value Returned

<code>t</code>	The de-selection is performed.
<code>nil</code>	The de-selection is not performed.

Examples

Deselects the objects in `win1` enclosed in the bounding box described by the coordinates 0:0, 25:25. Partial selection is not performed.

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

Related Topics

[partialSelect](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Selection Functions

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

<i>w_windowId</i>	Database ID of the window containing the objects.
<i>g_partial</i>	Whether partial selection should be performed. Valid Values: <code>t</code> , <code>nil</code>
<i>l_point</i>	The point to be used for de-selection. When no point is specified, this function operates interactively.

Value Returned

<code>t</code>	The de-selection is performed.
<code>nil</code>	The de-selection is not performed.

Examples

Deselects the object in `win3` located at the coordinate 0:0. Partial selection is not performed.

```
geSubSelectPoint( win3 nil list(0:0) )
```

Related Topics

[partialSelect](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Selection Functions

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

<i>enclosed_crossed</i>	<code>enclosed_crossed</code> area selection mode is active.
<i>enclosed</i>	<code>enclosed</code> area selection mode is active.
<i>edgeOnly</i>	<code>edgeOnly</code> area selection mode is active.
<i>vertexAndEdge</i>	<code>vertexAndEdge</code> area selection mode is active.

Related Topics

[Selection Functions](#)

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

<i>t_message</i>	Returns an information message indicating which resolution is applied to the current window.

Examples

Returns the information message, `Display resolution set to 'Very High'`.

```
geToggleDisplayResolution()
```

Related Topics

[displayResolution](#)

[Selection Functions](#)

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.

Related Topics

[Selection Functions](#)

geToggleWindowSelProtectionHighlight

```
geToggleWindowSelProtectionHighlight (
    [ w_window ]
)
=> t
```

Description

Toggles the highlight state of protected objects in the specified window.

Arguments

<i>w_window</i>	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

<i>t</i>	Highlight state of the protected objects is reversed.

Examples

Unhighlights the protected objects in the edit cellview if they are highlighted, or highlights the protected objects if they are not highlighted.

```
geToggleWindowSelProtectionHighlight ()
```

Related Topics

[Selection Functions](#)

geToggleWindowSelProtectionSelectability

```
geToggleWindowSelProtectionSelectability(  
    [ w_window ]  
)  
=> t
```

Description

Toggles the selectability state of protected objects in the specified window.

Arguments

<i>w_window</i>	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

<i>t</i>	Selectability of the protected objects is reversed.

Examples

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.

```
geToggleWindowSelProtectionSelectability()
```

Related Topics

[Selection Functions](#)

geUnhighlightProtected

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

Description

Removes the halo from the objects marked as protected in the specified cellview.

Arguments

<i>w_window</i>	The window in which protected objects are to be unhighlighted.

Value Returned

<i>t</i>	Protected objects are unhighlighted.
<i>nil</i>	Protected objects remain highlighted.

Examples

Removes the halo from all the protected objects in the edit cellview.

```
geUnhighlightProtected()
```

Related Topics

[Selection Functions](#)

geUnsetAllSelProtection

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

Description

Removes the selection protection from all the protected objects in the specified cellview.

Arguments

<i>d_cellview</i>	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.

Examples

Removes the selection protection from all the objects in the edit cellview.

```
geUnsetAllSelProtection()
```

Related Topics

[Selection Functions](#)

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

<i>l_results</i>	List of figures made unprotected, and therefore, selectable.
<i>nil</i>	The protection of the specified object is not removed.

Examples

The following example removes the selection protection from the *figId1* and *figId2* objects:

```
geUnsetSelProtection(list(figId1 figId2))
```

The following example removes the selection protection from all the selected objects in the *cellViewA* cellview:

```
geUnsetSelProtection(cellViewA)
```

The following example removes the selection protection from all the selected objects in the edit cellview:

```
geUnsetSelProtection()
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Selection Functions

Stored Selection Functions

A selection set consists of full or partial selection of figures. A selection set can be stored and restored using the stored selection functions.

Related Topics

[geDeleteStoredSelectionSet](#)

[geDeselectStoredSelectionSet](#)

[geGetStoredSelectionSets](#)

[geHiStoreSelectionForm](#)

[gelsAutoStoreSelectionOn](#)

[geSelectNextAutoStoreSet](#)

[geSelectPreviousAutoStoreSet](#)

[geSelectStoredSelectionSet](#)

[geSetAutoStoreSelection](#)

[geStoreSelectionSet](#)

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

<i>t_setName</i>	The name of the stored selection set.
<i>d_cellviewId</i>	Optional argument. When not specified, the current edit cellview is used. There is one current selection set by cellview.

Value Returned

<i>t</i>	The removal of the set is successful.
<i>nil</i>	The stored selection set does not exist.

Examples

Removes the specified stored selection set.

```
geDeleteStoredSelectionSet("selset1")
```

Related Topics

[Stored Selection Functions](#)

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

<i>t_setName</i>	The name of the stored selection set.
<i>d_cellviewId</i>	Optional argument. When not specified, the current edit cellview is used. There is one current selection set by cellview.

Value Returned

<i>t</i>	De-selection of elements is successful.
<i>nil</i>	The stored selection set does not exist.

Examples

Removes elements of the specified selection set.

```
geDeselectStoredSelectionSet("selset1")
```

Related Topics

[Stored Selection Functions](#)

geGetStoredSelectionSets

```
geGetStoredSelectionSets(  
    [ d_cellviewId ]  
)  
=> t_nameList / nil
```

Description

Returns the list of names for the stored selection sets in the given cellview.

Arguments

<i>d_cellviewId</i>	Optional argument. Database ID of the cellview that is queried for the stored selection sets. If not specified, the cellview of the edit cellview of the current window is used.

Value Returned

<i>t_nameList</i>	Returns the list of names.
<i>nil</i>	The cellview ID is invalid.

Examples

Returns the list of names for the stored selection sets in the *cv* cellview.

```
geGetStoredSelectionSets(cv)
```

Related Topics

[Stored Selection Functions](#)

geHiStoreSelectionForm

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

Description

Opens the Save/Restore Selection Set form.

Arguments

None.

Value Returned

t	The <i>Save/Restore Selection Set</i> form opens.
nil	An error occurred.

Examples

Opens the Save/Restore Selection Set form.

```
geHiStoreSelectionForm()
```

Related Topics

[Stored Selection Functions](#)

gelsAutoStoreSelectionOn

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

Description

Returns true if the auto store selection set feature is enabled.

Arguments

<i>d_cellviewId</i>	Optional argument of the database ID of the cellview.

Value Returned

t	Auto store selection set is on.
nil	The cellview ID is invalid.

Examples

```
geIsAutoStoreSelectionOn(cv)
```

Related Topics

[Stored Selection Functions](#)

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

<i>d_cellviewId</i>	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.

Examples

```
geSelectNextAutoStoreSet()
```

Related Topics

[Stored Selection Functions](#)

geSelectPreviousAutoStoreSet

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

Description

Restores the previous stored auto selection set. This function ends on the first stored selection set.

Arguments

<i>d_cellviewId</i>	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.

Examples

```
geSelectPreviousAutoStoreSet()
```

Related Topics

[Stored Selection Functions](#)

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

<i>t_setName</i>	The name of the stored selection set.
<i>d_cellviewId</i>	Optional argument of a cellview ID.

Value Returned

<i>t</i>	Elements of the specified selection are added to the current selection set.
<i>nil</i>	The cellview ID is invalid or the stored selection set does not exist.

Examples

```
geSelectStoredSelectionSet("selset1")
```

Related Topics

[Stored Selection Functions](#)

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

<i>g_autoStore</i>	Optional argument that specifies whether the auto stored set is enabled. Valid Values: t, nil
<i>d_cellviewId</i>	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.

Examples

```
geSetAutoStoreSelection(t)
```

Related Topics

[Stored Selection Functions](#)

geStoreSelectionSet

```
geStoreSelectionSet(  
    t_setName  
    [ d_cellviewId ]  
)  
=> t / nil
```

Description

Records the current selection set into the cellview manual store selection set list.

Arguments

<i>t_setName</i>	The name of the stored selection set.
<i>d_cellviewId</i>	Optional argument of a cellview ID.

Value Returned

<i>t</i>	The current selection set is stored.
<i>nil</i>	The selection set is not stored.

Examples

```
geStoreSelectionSet(setName)
```

Related Topics

Stored Selection Functions

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 `gePushHighlightStack`. You can only draw one non-global highlight set at a time.

Related Topics

Highlighting Shapes in a Cellview

`gePushHighlightStack`

`geAddHighlightArc`

`geAddHighlightBlockage`

`geAddHighlightCircle`

`geAddHighlightCurvedPath`

`geAddHighlightCurvedPolygon`

Highlighting Shapes in a Cellview

To highlight shapes in a cellview:

1. Create a highlight set using `geCreateHighlightSet`. 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 `geAddHighlight` functions. You can also remove shapes from a highlight set with the `geDeleteHighlight` functions.
3. Enable the highlight set by setting the enable attribute with `hlSetId->enable = t`.
4. If the highlight set is non-global, use `gePushHighlightStack` to draw the highlight set and `gePopHighlightStack` to remove the highlight set from the cellview.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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
```

Related Topics

[gePushHighlightStack](#)

[geCreateHighlightSet](#)

[gePopHighlightStack](#)

[Highlight Functions](#)

geAddHighlightArc

```
geAddHighlightArc (
    g_hlSetId
    l_ellipseBox
    l_arcBox
)
=> g_hlObjectId / nil
```

Description

Adds an arc to a highlight set.

l_ellipseBox and *l_arcBox* specify the location of the arc.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_ellipseBox</i>	List of points used to describe the ellipse.
<i>l_arcBox</i>	List of points used to describe the arc.

Value Returned

<i>g_hlObjectId</i>	The arc is added to the set.
<i>nil</i>	The arc is not added to the set.

Examples

Adds an arc to the highlight set *set1* and returns the object ID of the arc.

```
geAddHighlightArc ( set1 list(0:0 10:10) list(0:8 10:8) )
```

Related Topics

[Highlight Functions](#)

geAddHighlightBlockage

```
geAddHighlightBlockage(  
    g_hlSetId  
    d_blockageId  
)  
=> g_hlObjectId / nil
```

Description

Adds a blockage to a highlight set.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>d_blockageId</i>	The blockage Id.

Value Returned

<i>g_hlObjectId</i>	The blockage is added to the set.
<i>nil</i>	The blockage is not added to the set.

Related Topics

[Highlight Functions](#)

geAddHighlightCircle

```
geAddHighlightCircle(  
    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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_center</i>	Specifies the center of the circle.
<i>n_radius</i>	Specifies the size of the circle.

Value Returned

<i>g_hlObjectId</i>	The circle is added to the set.
nil	The circle is not added to the set.

Examples

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.

```
geAddHighlightCircle( set1 0:0 10 )
```

Related Topics

[Highlight Functions](#)

geAddHighlightCurvedPath

```
geAddHighlightCurvedPath(  
    g_hlSetId  
    l_points  
    n_width  
)  
=> g_hlObjectId / nil
```

Description

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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_points</i>	List of points used to describe the curved path.
<i>n_width</i>	Width of the curved path.

Value Returned

<i>g_hlObjectId</i>	The curved path is added to the set.
nil	The curved path is not added to the set.

Examples

Adds a path to the highlight set and returns the object ID of the curved path.

```
hSet = geCreateHighlightSet(geGetEditCellView() list("metal" "drawing")  
hSet~>enable = t  
geAddHighlightCurvedPath(hSet list(-11.1:5.8 -11.1:-8.9 18.2:-8.9 18.2:-14.1) 1)
```

Related Topics

[Highlight Functions](#)

geAddHilighCurvedPolygon

```
geAddHilighPolygon(  
    g_hlSetId  
    l_points  
)  
=> g_hlObjectId / nil
```

Description

Adds a curved polygon to a highlight set.

The points in *l_points* specify the location of the curved polygon.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_points</i>	List of points used to describe the curved polygon.

Value Returned

<i>g_hlObjectId</i>	The curved polygon is added to the set.
<i>nil</i>	The curved polygon is not added to the set.

Examples

Adds a curved polygon to the highlight set and returns the object ID of the polygon.

```
hSet = geCreateHilighSet(geGetEditCellView() list("metal" "drawing")  
hSet~>enable = t  
geAddHilighCurvedPolygon(hSet list(10:10 10:15 15:15 list(16.5 12.5 "arc" "cw")  
15:10) )
```

Related Topics

[Highlight Functions](#)

geAddHilightDistance

```
geAddHilightDistance(  
    h_hlSetId  
    l_points [l_points]  
)  
=> h_hlObjectId
```

Description

Adds a distance, such as a ruler, to the given highlight set or group.

Arguments

<i>h_hlSetId</i>	Unique ID of a specific highlight Set or Group.
<i>l_points</i>	The highlight distance in terms of points.
<i>[l_points]</i>	Additional highlight distance in terms of points.

Value Returned

<i>h_hlObjectId</i>	Returns the ID of the created highlight object.

Examples

```
geAddHilightDistance(hgroupId list(-5:-3.6 2:-3.6) list(-5:0 2:-3.6))
```

Related Topics

[Highlight Functions](#)

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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_center</i>	Specifies the center of the donut.
<i>n_outR</i>	Specifies the outer radius of the donut.
<i>n_holeR</i>	Specifies the inner radius of the donut.

Value Returned

<i>g_hlObjectId</i>	The donut is added to the set.
<i>nil</i>	The donut is not added to the set.

Examples

Adds a donut to the highlight set `set1` whose center point is 0:0, outer radius is 10, and inner radius is 2. Returns the object ID of the donut.

```
geAddHilightDonut( set1 0:0 10 2 )
```

Related Topics

[Highlight Functions](#)

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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_point</i>	Point used to locate the dot.
<i>n_width</i>	Width of the dot.
<i>n_height</i>	Height of the dot.

Value Returned

<i>g_hlObjectId</i>	The dot is added to the set.
nil	The dot is not added to the set.

Examples

Adds a 5x5 dot to the highlight set `set1` with the center point at 0:0, and returns the object ID of the dot.

```
geAddHilightDot( set1 list(0:0) 5 5 )
```

Related Topics

Highlight Functions

geAddHighlightEllipse

```
geAddHighlightEllipse(  
    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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_bBox</i>	List of points used to describe the ellipse.

Value Returned

<i>g_hlObjectId</i>	The ellipse is added to the set.
<i>nil</i>	The ellipse is not added to the set.

Examples

Adds an ellipse to the highlight set `set1` and returns the object ID of the ellipse.

```
geAddHighlightEllipse( set1 list(0:0 10:10) )
```

Related Topics

[Highlight Functions](#)

geAddHighlightFig

```
geAddHighlightFig(  
    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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>d_figId</i>	Database ID of the figure added to the highlight set.
<i>g_drawAll</i>	Specifies whether all figures in the instance are drawn. Valid Values: <i>t</i> , <i>nil</i>
<i>l_hierPath</i>	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 (<i>dbInst memInst row column</i>).
<i>l_vertex</i>	Specifies the Boolean list of vertices of the figure to highlight. This option can be used to partially highlight a figure.

Value Returned

<i>g_hlObjectId</i>	The figure is added to the set.
<i>nil</i>	The figure is not added to the set.

Examples

The following example adds *fig3* to the highlight set *set1* and does not draw the objects. Returns the object ID of *fig3*.

```
geAddHighlightFig( set1 fig3 nil )
```

The following example adds *instId* to the highlight set *set1* and highlights the objects inside *instId*.

```
geAddHighlightFig(set1 instId t)
```

The following example adds *figId* to the highlight set *set1*, located in the hierarchy at the given *hierpath*, *inst1/inst2*.

```
geAddHighlightFig(set1 figId nil list(list(inst1 0 0 0) list(inst2 0 0 0)))
```

The following example adds *rect* to the highlight set *set1*, but only highlights vertices 2 and 3.

```
geAddHighlightFig(set1 rect nil nil list(nil t t nil))
```


Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Highlight Functions

geAddHighlightLabel

```
geAddHighlightLabel(  
    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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_origin</i>	Specifies the location of the label.
<i>t_label</i>	Specifies the text of the label.
<i>t_just</i>	Specifies the justification of the label. Valid Values: upperLeft, upperCenter, upperRight, centerLeft, centerCenter, centerRight, lowerLeft, lowerCenter, lowerRight
<i>t_orient</i>	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)
<i>t_font</i>	Specifies the font style used for the label. Valid Values: europe, gothic, roman, script, stick
<i>n_height</i>	Specifies the height of the label.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

<i>g_draftingP</i>	Determines how you can rotate text. <i>t</i> allows the text to rotate 90 degrees, <i>nil</i> allows rotation in all directions.
<i>g_overBar</i>	Determines whether the label has an over-bar or not. Valid Values: <i>t</i> , <i>nil</i> . Default: <i>nil</i> .

Value Returned

<i>g_hlObjectId</i>	The label is added to the set.
<i>nil</i>	The label is not added to the set.

Examples

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. It returns the object ID of the label.

```
set1 = geCreateHighlightSet( gec() list("Metall" "drawing") nil )
set1->enable=t
geAddHighlightLabel(set1 list(0 0) "mytext" "lowerLeft" "R0" "stick" 2 nil)
=>hl:0x3aa9e030
```

Related Topics

[Highlight Functions](#)

geAddHighlightLine

```
geAddHighlightLine(  
    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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_points</i>	List of points used to describe the line.

Value Returned

<i>g_hlObjectId</i>	The line is added to the set.
<i>nil</i>	The line is not added to the set.

Examples

Adds a line to the highlight set `set1` and returns the object ID of the line.

```
geAddHighlightLine( set1 list(0:0 10:0 10:10) )
```

Related Topics

[Highlight Functions](#)

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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_points</i>	List of points used to describe the polygon.
<i>n_width</i>	Width of the path.

Value Returned

<i>g_hlObjectId</i>	The path is added to the set.
nil	The path is not added to the set.

Examples

Adds a path to the highlight set `set1` and returns the object ID of the path.

```
geAddHilightPath( set1 list(0:0 10:10 5:10 0:5) 5 )
```

Related Topics

[Highlight Functions](#)

geAddHighlightPathSeg

```
geAddHighlightPathSeg(  
    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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_beginPt</i>	The start point of the path segment.
<i>l_endPt</i>	The end point of the path segment.
<i>n_width</i>	Width of the path.
<i>t_beginStyle</i>	The style for the beginning of the path segment.
<i>t_endStyle</i>	The style for the end of the path segment.
<i>l_extValue</i>	Extension value of a pathSeg. User needs to specify the value set of <i>n_beginExt</i> , <i>n_beginLeftDiagExt</i> , <i>n_beginRightDiagExt</i> , <i>n_beginRightHalfWidth</i> , and/or <i>n_endExt</i> , <i>n_endLeftDiagExt</i> , <i>n_endRightDiagExt</i> , <i>n_endRightHalfWidth</i> in the argument <i>l_extValue</i> .

Value Returned

<i>g_hlObjectId</i>	The pathSeg is added to the set.
<i>nil</i>	The pathSeg is not added to the set.

Examples

```
cv = geGetEditCellView()
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
=> db:0x4522df92
hl = geCreateHighlightSet(cv list("text" "drawing"))
=> hl:0x110a5018
hl->enable = t
=> t
geAddHighlightPathSeg(hl list(0.01 4.975) list(2.005 4.975) 0.12 "extend" "extend")
=> hl:0x110a4018
```

Related Topics

[Highlight Functions](#)

geAddHighlightPolygon

```
geAddHighlightPolygon(  
    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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_points</i>	List of points used to describe the polygon.

Value Returned

<i>g_hlObjectId</i>	The polygon is added to the set.
<i>nil</i>	The polygon is not added to the set.

Examples

Adds a polygon to the highlight set `set1` and returns the object ID of the polygon.

```
geAddHighlightPolygon( set1 list(0:0 10:10 5:10 0:5) )
```

Related Topics

[Highlight Functions](#)

geAddHighlightRow

```
geAddHighlightRow(  
    g_hlSetId  
    d_siteDef  
    l_point  
    x_numSites  
    [ t_siteOrient ]  
    [ t_rowOrient ]  
    => g_hlObjectId / nil
```

Description

Adds a row to a highlight set.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>d_siteDef</i>	The <code>siteDef</code> to be used for this row.
<i>l_point</i>	The origin point for the row.
<i>x_numSites</i>	The number of site placements in the row.
<i>t_siteOrient</i>	Specifies the orientation of the site placements of the row. Default value: R0
<i>t_rowOrient</i>	Specifies the orientation of the row. Default value: R0

Value Returned

<i>g_hlObjectId</i>	The row is added to the set.
<i>nil</i>	The row is not added to the set.

geAddHighlightRuler

```
geAddHighlightRuler(  
    g_hlSetId  
    l_points  
    => g_hlObjectId / nil
```

Description

Adds a ruler to a highlight set.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_points</i>	List of points used to describe the ruler.

Value Returned

<i>g_hlObjectId</i>	The ruler is added to the set.
<i>nil</i>	The ruler is not added to the set.

Examples

Adds a ruler to the highlight set `hset` and returns the object ID of the ruler.

```
geAddHighlightRuler(hset list(0:0 5:5) )
```

Related Topics

[Highlight Functions](#)

geAddHighlightSlicedCircle

```
geAddHighlightSlicedCircle(  
    g_hlSetId  
    l_center  
    n_radius  
    f_startAngle  
    f_StopAngle  
)  
=> g_hlObjectId / nil
```

Description

Adds a sliced circle to a highlight set.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_center</i>	Specifies the center of the sliced circle.
<i>n_radius</i>	Specifies the size of the sliced circle.
<i>f_startAngle</i>	Specifies the start angle of the sliced circle.
<i>f_StopAngle</i>	Specifies the stop angle of the sliced circle.

Value Returned

<i>g_hlObjectId</i>	The sliced circle is added to the set.
<i>nil</i>	The sliced circle is not added to the set.

Examples

```
hl = geCreateHighlightSet(geGetEditCellView() list("y0" "drawing"))  
hl->enable = t  
geAddHighlightSlicedCircle(hl (10:7) 6 45.0 125.0)
```

Related Topics

[Highlight Functions](#)

geAddHighlightSlicedDonut

```
geAddHighlightSlicedDonut (
    g_hlSetId
    l_center
    n_radius
    f_startAngle
    f_StopAngle
)
=> g_hlObjectId / nil
```

Description

Adds a sliced donut to a highlight set.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_center</i>	Specifies the center of the sliced donut.
<i>n_radius</i>	Specifies the size of the sliced donut.
<i>f_startAngle</i>	Specifies the start angle of the sliced donut.
<i>f_StopAngle</i>	Specifies the stop angle of the sliced donut.

Value Returned

<i>g_hlObjectId</i>	The sliced donut is added to the set.
<i>nil</i>	The sliced donut is not added to the set.

Examples

```
hl = geCreateHighlightSet(geGetEditCellView() list("y0" "drawing"))
hl->enable = t
geAddHighlightSlicedDonut(hl (1:-20) 6 3 45.0 190.0)
```

Related Topics

[Highlight Functions](#)

geAddHilighTrackPattern

```
geAddHilighTrackPattern(  
    g_hlSetId  
    d_trackPatternId  
)  
=> g_hlObjectId / nil
```

Description

Adds a track pattern to a highlight set.

Arguments

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>d_trackPatternId</i>	The track pattern Id.

Value Returned

<i>g_hlObjectId</i>	The track pattern is added to the set.
<i>nil</i>	The track pattern is not added to the set.

Examples

```
win = window(3)  
cv=geGetEditCellView(win)  
tp = nth(0 cv~>trackPatterns)  
hilSet = geCreateHilighSet(cv ' ("hilite" "drawing")) |  
hilSet~>enable=t  
geAddHilighTrackPattern(hilSet tp)
```

Related Topics

[Highlight Functions](#)

geAddHighlightRectangle

```
geAddHighlightRectangle(  
    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

<i>g_hlSetId</i>	Database ID of the highlight set.
<i>l_bBox</i>	List of points used to describe the rectangle.

Value Returned

<i>g_hlObjectId</i>	The rectangle is added to the set.
<i>nil</i>	The rectangle is not added to the set.

Examples

Adds a rectangle to the highlight set `set1` and returns the object ID of the rectangle.

```
geAddHighlightRectangle( set1 list(0:0 10:10) )
```

Related Topics

[Highlight Functions](#)

geChangeHighlightPacketName

```
geChangeHighlightPacketName(  
    h_hlSetId  
    g_packetName  
)  
=> t / nil
```

Description

Changes the packet on which the given highlight set is drawn.

Arguments

<i>h_hlSetId</i>	The database ID of the highlight set.
<i>g_packetName</i>	The name of the packet.

Value Returned

t	The name of the packet is changed.
nil	The highlight set ID is invalid.

Examples

```
geChangeHighlightPacketName(hsetId "annotate")
```

Related Topics

[Highlight Functions](#)

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

<i>h_hlSetId</i>	The database ID of the highlight set.
<i>l_hlObjects</i>	Optional argument that lists highlight objects to be placed in the group container being created.

Value Returned

<i>hlGroupId</i>	The database ID of the group container.
<i>nil</i>	The highlight set ID is invalid.

Examples

```
geCreateHilightGroup( hsetId list(hlObject1 hlObject2))
```

Related Topics

[Highlight Functions](#)

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 are 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

<i>d_cellview</i>	Database ID of the cellview in which to create the highlight set.
<i>l_layerPurposePair</i>	Layer-purpose pair used to create the highlight objects.
<i>g_notGlobal</i>	Specifies whether the highlight set is added to the global list. Valid Values: <i>t</i> , <i>nil</i>

Value Returned

<i>g_hlSetId</i>	The highlight set is created.
<i>nil</i>	The highlight set is not created.

Examples

Creates a highlight set in *cell13* on *layer1* with *purpose1* and adds the set to the global list.

```
geCreateHilightSet( cell13 list("layer1" "purpose1") nil )
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Highlight Functions

geCreateWindowHighlightSet

```
geCreateWindowHighlightSet(  
    w_windowId  
    l_layerPurposePair | t_packetName  
    [ g_notGlobal ]  
)  
=> h_highlightSetId / nil
```

Description

Creates a 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

<i>w_windowId</i>	The ID of the window.
<i>l_layerPurposePair</i> <i>t_packetName</i>	The layer purpose pair or a packet name.
<i>g_notGlobal</i>	Optional argument that specifies whether the highlight set is added to the global list.
	Valid Values: <code>t</code> , <code>nil</code>

Value Returned

<i>h_highlightSetId</i>	The database ID of the highlight set is returned.
<code>nil</code>	The LPP or packet name is invalid.

Examples

Creates a highlight set on the `metal 1` drawing LPP in `window(3)`.

```
geCreateWindowHighlightSet( window(3) list("metal1" "drawing") nil)
```

Related Topics

[Highlight Functions](#)

geDeleteAllHighlightSet

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

Description

Deletes all highlight sets from the specified window.

Arguments

<i>d_cellview</i>	Database ID of the highlight set.

Value Returned

t	All sets are deleted.
nil	Sets are not deleted.

Related Topics

[Highlight Functions](#)

geDeleteAllWindowsHighlightSet

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

Description

Deletes all the window highlight sets registered on the given window.

Arguments

<i>w_windowId</i>	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.

Examples

Deletes all the window highlight sets in `window(3)`.

```
geDeleteAllWindowsHighlightSet(window(3))
```

Related Topics

[Highlight Functions](#)

geDeleteHighlightGroup

```
geDeleteHighlightGroup(  
    hl_groupId  
    [ g_deleteAll ]  
)  
=> t / nil
```

Description

Deletes the specified highlight group.

Arguments

<i>hl_groupId</i>	The database ID of the highlight group.
<i>g_deleteAll</i>	Optional argument. When <code>t</code> , each highlight object in the group is also deleted unless it belongs to another group. When <code>nil</code> , each highlight object in the group is placed back in the highlight set object list unless it belongs to another group. The default is <code>nil</code> .

Value Returned

<code>t</code>	The specified highlight group is deleted.
<code>nil</code>	An error occurred and the highlight group is not deleted.

Examples

```
geDeleteHighlightGroup(hlGroupId t)
```

Related Topics

[Highlight Functions](#)

geDeleteHighlightObject

```
geDeleteHighlightObject(  
    g_hlObjectId  
)  
=> t / nil
```

Description

Deletes the highlight object from the highlight set.

Arguments

<i>g_hlObjectId</i>	Database ID of the highlight object.

Value Returned

t	The object is deleted.
nil	The object is not deleted.

Examples

Deletes object2.

```
geDeleteHighlightObject( object2 )
```

Related Topics

[Highlight Functions](#)

geDeleteHighlightSet

```
geDeleteHighlightSet (
    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

<i>g_hlSetId</i>	Database ID of the highlight set.

Value Returned

t	The set is deleted.
nil	The set is not deleted.

Examples

Deletes set2.

```
geDeleteHighlightSet( set2 )
```

Related Topics

[Highlight Functions](#)

geDeleteHighlightSetInWindows

```
geDeleteHighlightSetInWindows (
    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

<i>hl_groupId</i>	The database ID of the highlight group.
<i>l_windowList</i>	List of ID for windows in which the highlight set is to be deleted.

Value Returned

<i>t</i>	The specified highlight group is deleted.
<i>nil</i>	An error occurred and the highlight group is not deleted.

Examples

```
geDeleteHighlightSetInWindows(hlSetId list(window(3) window(5)))
```

Related Topics

[Highlight Functions](#)

geDrawHighlightSet

```
geDrawHighlightSet (
    [ 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

<i>w_windowId</i>	The window ID.
<i>g_hlSetId</i>	Database ID of the highlight set.

Value Returned

<i>t</i>	The set is redrawn.
<i>nil</i>	The set is not redrawn.

Related Topics

[Highlight Functions](#)

geGetCurrentHighlightSet

```
geGetCurrentHighlightSet(  
    d_cellview  
)  
=> g_hlSetId / nil
```

Description

Retrieves the current highlight set.

The highlight stack in the cellview is specified by *d_cellview*.

Arguments

<i>d_cellview</i>	Database ID of the cellview containing the highlight stack.

Value Returned

<i>g_hlSetId</i>	The highlight set is retrieved.
<i>nil</i>	The cellview is not valid or the stack is empty.

Examples

Returns the highlight set ID of the current highlight set in *cell13*.

```
geGetCurrentHighlightSet( cell13 )
```

Related Topics

[Highlight Functions](#)

geGetCurrentWindowHighlightSet

```
geGetCurrentWindowHighlightSet (  
    w_windowId  
)  
=> d_hlSetId / nil
```

Description

Returns the window highlight set which is at the top of the window highlight stack.

Arguments

<i>w_windowId</i>	Database ID of the window.

Value Returned

<i>d_hlSetId</i>	The database ID of the current window highlight set is returned.
<i>nil</i>	The window ID is invalid.

Examples

```
geGetCurrentWindowHighlightSet (window(3))
```

Related Topics

[Highlight Functions](#)

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

<i>g_hlSetId</i>	Database ID for the highlight set.
<i>d_figId</i>	Database ID for the object.

Value Returned

<i>g_hlObjectId</i>	The highlight object ID for the specified object.
nil	The highlight object ID is not found.

Examples

Returns the highlight object ID for the object *fig2* in the highlight set *set1*.

```
geGetFigHilightObjectId( set1 fig2 )
```

Related Topics

[Highlight Functions](#)

geGetHilightObjectFigId

```
geGetHilightObjectFigId(  
    h_hlObjectId  
)  
=> d_figId / nil
```

Description

Returns the figure ID of the highlighted object.

Arguments

<i>h_hlObjectId</i>	Database ID of the highlighted object.

Value Returned

<i>d_figId</i>	The figure ID of the highlighted object is returned.
<i>nil</i>	The highlight object ID is invalid.

Examples

```
geGetHilightObjectFigId(hlObjectId)
```

Related Topics

[Highlight Functions](#)

geInsertHighlightObjectsInGroup

```
geInsertHighlightObjectsInGroup(  
    h_hlGroupId  
    [ l_hlObjectId ]  
)  
=> t / nil
```

Description

Groups all given highlighted objects together.

Arguments

<i>h_hlGroupId</i>	Database ID of the group in which to insert the highlighted object.
<i>l_hlObjectId</i>	Optional argument. Database ID of the highlighted object.

Value Returned

<i>t</i>	The highlight objects are grouped together.
<i>nil</i>	An error occurred.

Examples

```
geInsertHighlightObjectsInGroup( hlGroup2 list(hlObject1 hlObject2 hlGroup1))
```

Related Topics

[Highlight Functions](#)

geIsValidHighlightGroup

```
geIsValidHighlightGroup(  
    h_hlGroupId  
)  
=> t / nil
```

Description

Determines whether the highlighted group is valid.

Arguments

<i>h_hlGroupId</i>	The database ID of the highlight group.

Value Returned

<i>t</i>	The highlight group is valid.
<i>nil</i>	The highlight group is not valid.

Examples

```
geIsValidHighlightGroup(hlGroup)
```

Related Topics

[Highlight Functions](#)

geIsValidHighlightId

```
geIsValidHighlightId(  
    h_hlId  
)  
=> t / nil
```

Description

This function determines whether a highlight ID is valid.

Arguments

<i>h_hlId</i>	The database ID of the highlighted group, set, or object.

Value Returned

<i>t</i>	The highlight ID is valid.
<i>nil</i>	The highlight ID is not valid.

Examples

```
geIsValidHighlightId(hlSet)  
geIsValidHighlightId(hlGroup)  
geIsValidHighlightId(hlObject)
```

Related Topics

[Highlight Functions](#)

geIsValidHighlightObject

```
geIsValidHighlightObject(  
    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 `geAddHighlight` functions.

Arguments

<i>h_hlObjectId</i>	Any highlighted object ID returned from any of the <code>geAddHighlight</code> functions.

Value Returned

t	The highlighted object is valid.
nil	The highlighted object is not valid.

Examples

The following example returns t.

```
hlObjectId = geAddHighlightPolygon( set1  
list( 0:0 10:0 10:10 0:10 ) )  
geIsValidHighlightObject( hlObjectId )
```

The following example returns nil.

```
geIsValidHighlightObject( geGetEditCellView( ) )
```

Related Topics

[Highlight Functions](#)

[geAddHighlightArc](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

geAddHighlightBlockage

geAddHighlightCircle

geAddHighlightCurvedPath

geAddHighlightCurvedPolygon

geAddHighlightDistance

geIsValidHighlightSet

```
geIsValidHighlightSet(  
    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 `geCreateHighlightSet` function.

Arguments

<i>h_hlSetId</i>	Any highlight set ID returned from <code>geCreateHighlightSet</code> .

Value Returned

t	The highlight set is valid.
nil	The highlight set is not valid.

Examples

The following example returns t.

```
hlSetId = geCreateHighlightSet( geGetEditCellView( ) list( "hilite" "drawing" ) )  
geIsValidHighlightSet( hlSetId )
```

The following example returns nil.

```
geIsValidHighlightSet( geGetEditCellView( ) )
```

Related Topics

[Highlight Functions](#)

[geCreateHighlightGroup](#)

[geCreateHighlightSet](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

geCreateWindowHighlightSet

gePopHighlightStack

```
gePopHighlightStack(  
    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

<code>d_cellview</code>	Database ID of the cellview containing the highlight stack.

Value Returned

<code>nil</code>	The highlight set is not popped out of the stack.

Examples

Pops the current highlight set out of the stack in `cell1`.

```
gePopHighlightStack( cell1 )
```

Related Topics

[Highlight Functions](#)

gePopWindowHighlightStack

```
gePopWindowHighlightStack(  
    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

<i>w_windowId</i>	Window ID of the current window.

Value Returned

<i>t</i>	The window highlight set is popped out of the stack.
<i>nil</i>	The window highlight set is not popped out of the stack

Examples

```
gePopWindowHighlightStack(window(3))
```

Related Topics

[Highlight Functions](#)

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

<i>g_hlSetId</i>	Database ID of the highlight set.

Value Returned

<i>t</i>	The highlight set is pushed into the highlight stack.
<i>nil</i>	The highlight set is not pushed into the highlight stack

Examples

Pushes highlight set `set13` into the stack.

```
gePushHilightStack( set13 )
```

Related Topics

[Highlight Functions](#)

geResetHighlightSetAlphaBlendingParameters

```
geResetHighlightSetAlphaBlendingParameters(  
    g_hSetId  
)  
=> t / nil
```

Description

Resets the alpha blending values to maximum values (1000) for the given highlight set.

Arguments

<i>g_hSetId</i>	The highlight set ID.

Value Returned

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

Examples

The following example resets the alpha blending values for a highlight set.

```
hSetId = geCreateHighlightSet(geGetEditCellView() list("Metall" "drawing"))  
geAddHighlightRectangle(hSetId list(0:0 5:5))  
hSetId->enable=t  
geSetHighlightSetAlphaBlendingParameters(hSetId 650 650 650)  
hiRedraw()  
geResetHighlightSetAlphaBlendingParameters(hSetId)  
hiRedraw()
```

Related Topics

[Highlight Functions](#)

geResetHilighSetHaloParameters

```
geResetHilighSetHaloParameters(  
    h_hlSetId  
)  
=> t / nil
```

Description

Resets the halo parameters set on a highlight set.

Arguments

<i>h_hlSetId</i>	The database ID of the highlight set.

Value Returned

<i>t</i>	Resets the halo parameters set on a highlight set.
<i>nil</i>	The window ID is invalid.

Examples

```
geResetHilighSetHaloParameters(highlighSet)
```

Related Topics

[Highlight Functions](#)

geSetHighlightSetAlphaBlendingParameters

```
geSetHighlightSetAlphaBlendingParameters (  
    g_hSetId  
    g_halo  
    g_fill  
    g_outLine  
)  
=> t / nil
```

Description

This function sets the alpha blending values for the specified highlight set.

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 `geSetHighlightSetHaloParameters` and `geSetHighlightSetAlphaBlendingParameters` 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

<i>g_hSetId</i>	The highlight set ID.
<i>g_halo</i>	The new halo value.
<i>g_fill</i>	The new fill value.
<i>g_outLine</i>	The new outline value.

Value Returned

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

Examples

The following example changes the default alpha blending values for the specified highlight set.

```
hSetId = geCreateHighlightSet(geGetEditCellView() list("Metall" "drawing"))  
geAddHighlightRectangle(hSetId list(0:0 5:5))
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
hSetId->enable=t  
geSetHighlightSetAlphaBlendingParameters(hSetId 650 650 650)  
hiRedraw()
```

Related Topics

[Highlight Functions](#)

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>h_hlSetId</i>	The database ID of the highlight set.
<i>g_haloPosition</i>	The position of the halo. Valid values: over, under
<i>g_haloType</i>	The halo type. Valid values: none, plain, glow, fadeout, random
<i>g_haloThickNess</i>	The halo thickness. Valid values: normal, thin, thick
<i>g_haloTransparency</i>	The transparency value. Valid values: [0 100]
<i>g_drawStroke</i>	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.

Examples

```
geSetHilightSetHaloParameters(hlsetId "under" "glow" "normal" 50 nil )
```

Related Topics

[Highlight Functions](#)

geShareWindowHighlightSet

```
geShareWindowHighlightSet(  
    h_hlSetId  
    l_windowList  
)  
=> t / nil
```

Description

Shares the given window highlight set with other windows.

Arguments

<i>h_hlSetId</i>	The database ID of the highlight set.
<i>l_windowList</i>	List of windows ID in which to share the highlight set.

Value Returned

<i>t</i>	The window highlight set is shared with the specified list of windows.
<i>nil</i>	The highlight set ID or window IDs are invalid.

Examples

```
geShareWindowHighlightSet(hlSetId list(window(4) window(5)))
```

Related Topics

[Highlight Functions](#)

Background Cellview Functions

You can use the background cellview functions to manage the display of the background cellview for a window. You can enable or disable the display of the background cellview. You can set and reset the background cellview and zoom to fit it in the window.

Related Topics

[geBackgroundCellView](#)

[geDisableBackgroundCellView](#)

[geEnableBackgroundCellView](#)

[geGetBackgroundCellView](#)

[geResetBackgroundCellView](#)

[geSetBackgroundCellView](#)

[geZoomToBackgroundAndEditCV](#)

[geZoomToBackgroundCV](#)

geBackgroundCellView

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

Description

Opens the Background CellView form which is used to superimpose 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.

Examples

Opens the Background CellView form.

```
geBackgroundCellView()
```

Related Topics

[Background Cellview Functions](#)

geDisableBackgroundCellView

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

Description

Disables the display of the background cellview for the given window. The current window is used by default.

Arguments

<i>w_windowId</i>	The ID of the window.

Value Returned

<i>t</i>	The display of the background cellview is disabled.
<i>nil</i>	The window ID is invalid.

Examples

Disables the display of the background cellview for the current window.

```
geDisableBackgroundCellView()
```

Related Topics

[Background Cellview Functions](#)

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

<i>w_windowId</i>	The database ID of the window in which to display the background cellview.
<i>g_dimmed</i>	Specifies if the background cellview is displayed dimmed. Valid Values: <code>t</code> , <code>nil</code>

Value Returned

<code>t</code>	The display of the background cellview is successful.
<code>nil</code>	The window ID is invalid.

Examples

Displays the background cellview as dimmed in the current window.

```
geEnableBackgroundCellView(hiGetCurrentWindow() t)
```

Related Topics

[Background Cellview Functions](#)

geGetBackgroundCellView

```
geGetBackgroundCellView(  
    [ w_windowId ]  
)  
=> l_cellViewId / l_transform
```

Description

Returns the cellview used for the background cellview.

Arguments

w_windowId Optional argument. ID of the window.

Value Returned

l_cellViewId l_transform List containing the cellview ID and the transform used for the background.

Examples

Returns the cellview used for the background cellview.

```
win = hiGetCurrentWindow()  
geGetBackgroundCellView(win)  
  (db:0x18b9d39a  
    ((-25.0 -25.0) "R0") 1.0  
  )
```

Related Topics

[Background Cellview Functions](#)

geResetBackgroundCellView

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

Description

Resets the background cellview on the given window.

Arguments

<i>w_windowId</i>	Optional argument. ID of the window.

Value Returned

<i>t</i>	Resets the background cellview on the given window.
<i>nil</i>	The window ID is invalid.

Examples

Resets the background cellview in window 3.

```
geResetBackgroundCellView(window(3))
```

Related Topics

[Background Cellview Functions](#)

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

<i>w_windowId</i>	Window ID.
<i>d_cellViewId</i> <i>l_libName</i> , <i>cellName</i> , <i>viewName</i>	Cellview ID or a list specifying the library, cell, or view name.
<i>l_transform</i>	Optional argument that will place the background cellview at a specified location and rotation.

Value Returned

<i>t</i>	The background is set to the specified cellview.
<i>nil</i>	The cellview ID or specified list is invalid.

Examples

Places the background cellview at location 5:10 with a rotation R90.

```
geSetBackgroundCellView(window(3) list( "myLib" "myCell" "layout")  
geSetBackgroundCellView(winId list(libname cellname viewname) list(5:10 "R0"))  
  
or  
  
geSetBackgroundCellView(winId cvId list(5:10 "R0"))
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

[Background Cellview Functions](#)

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

<i>w_windowId</i>	Optional argument. ID of the window.

Value Returned

<i>t</i>	Zooms to fit the background/edit cellview.
<i>nil</i>	The specified window does not exist.

Examples

Zooms to fit the background cellview and edit cellview in window 3.

```
geZoomToBackgroundAndEditCV(window(3))
```

Related Topics

[Background Cellview Functions](#)

geZoomToBackgroundCV

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

Description

When a background cellview is set on the window, zooms to fit the background cellview.

Arguments

<i>w_windowId</i>	Optional argument. ID of the window.

Value Returned

<i>t</i>	Zooms to fit the background window.
<i>nil</i>	The specified window does not exist.

Examples

Zooms to fit the background cellview in window 3.

```
geZoomToBackgroundCV(window(3))
```

Related Topics

[Background Cellview Functions](#)

Marker Functions

The marker functions enable you to work with markers. You can create, delete, and find markers in a cellview. You can also perform tasks such as display the marker owner, marker reason, and the text associated with a marker.

Related Topics

[geCreateMarker](#)

[geCreateMarkerByBBox](#)

[geCreateMarkerByPoints](#)

[geDeleteAllMarker](#)

[geGetMarkerDisplayText](#)

[geGetMarkerOwner](#)

[geGetMarkerReason](#)

[geGetMarkerType](#)

[geHasMarker](#)

[geHiCommonFindMarker](#)

[geHiDeleteAllMarker](#)

[geHiDeleteMarker](#)

[geHiExplainMarker](#)

[geHiFindMarker](#)

[gelsMarkerShape](#)

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_severity* 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.

<i>figId</i>	The bounding box of the shape is used for the marker.
<i>instId</i>	The bounding box of the instance is used for the marker.
<i>instHeaderId</i>	The bounding box of the first instance for the instance header is used for the marker.
<i>instTermId</i>	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.
<i>netId</i>	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.
<i>pinId</i>	The bounding box of the figure representing the pin is used for the marker.
<i>termId</i>	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.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

describes the problem causing the marker. The *t_displayText* value is a brief warning/error message that is displayed with the marker shape.

Arguments

<i>d_dbobjectId</i>	Database ID of the cellview to contain the marker.
<i>t_severity</i>	Severity of the marker. Valid Values: <code>warning</code> , <code>error</code>
<i>t_owner</i>	Owner of the marker. Valid values include any valid owner name.
<i>t_reason</i>	Reason for the violation.
<i>t_displayText</i>	Text associated with the marker.

Value Returned

<i>d_shapeId</i>	The shape representing the marker, if the marker is created.
<code>nil</code>	The marker is not created.

Examples

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.

```
geCreateMarker( cell2 "warning" "PDV" "physical rule violation" "mytext" )
```

Related Topics

Marker Functions

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

<i>d_cellview</i>	Database ID of the cellview to contain the marker.
<i>t_type</i>	Severity of the marker. Valid Values: <code>warning</code> , <code>error</code>
<i>t_owner</i>	Owner of the marker. Valid Values: <code>DLE</code> , <code>PDV</code> , or any marker owner
<i>t_reason</i>	Reason for violation.
<i>t_displayText</i>	Message displayed with the marker.
<i>l_bBox</i>	Bounding box that defines the marker shape.

Value Returned

<i>d_shapeId</i>	The shape representing the marker, if the marker is created.
<code>nil</code>	The marker is not created.

Examples

Creates a marker in the cellview `cell12` using the coordinates in the list. The marker severity is `warning` and the marker owner is `PDV`. Poly-poly spacing violation is the reason for the marker. `Mytext` is displayed with the marker. Returns the shape ID of the marker shape.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
geCreateMarkerByBBox( cell2 "warning" "PDV" "poly-poly spacing violation" "mytext"  
list(12:10 20:18) )
```

Related Topics

Marker Functions

geCreateMarkerByPoints

```
geCreateMarkerByPoints(  
    d_cellview  
    t_type  
    t_owner  
    t_reason  
    t_displayText  
    l_points  
)  
=> d_shapeId / nil
```

Description

Creates a polygon-shaped marker.

Arguments

<i>d_cellview</i>	Database ID of the cellview to contain the marker.
<i>t_type</i>	Severity of the marker. Valid Values: warning, error
<i>t_owner</i>	Owner of the marker. Valid Values: any marker owner
<i>t_reason</i>	Reason for violation.
<i>t_displayText</i>	Message displayed with the marker.
<i>l_points</i>	List of points defining the marker.

Value Returned

<i>d_shapeId</i>	The shape representing the marker, if the marker is created.
<i>nil</i>	The marker is not created.

Examples

Creates a marker in the cellview `cell12` 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.

```
geCreateMarkerByPoints( cell12 "warning" "PDV" "metall-metall spacing violation"  
"mytext" list(12:10 20:10 20:18 10:18 10:14 12:14) )
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Marker Functions

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

<i>d_cellview</i>	Database ID of the cellview containing the markers.
<i>l_owner</i>	List of marker owners. Valid Values: any valid owner name
<i>l_severity</i>	List of marker severities. Valid Values: <i>warning</i> , <i>error</i>
<i>g_traverse</i>	Specifies the traversal hierarchy, starting from the cellview. Specify this value as <i>nil</i> if you do not want the function to traverse the hierarchy.

Value Returned

<i>t</i>	The markers are deleted.
<i>nil</i>	The markers are not deleted.

Examples

In the following example, the function deletes all markers in *cell13* that are of severity *error* and owned by *DLE* or *PDV* but does not traverse the hierarchy because the last argument is *nil*.

```
geDeleteAllMarker( cell13 list("DLE" "PDV") "error" nil )
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

In the following example, the function deletes the markers only in the specified cellview, `cvId`.

```
geDeleteAllMarker( cvId nil nil nil)
```

Related Topics

[Marker Functions](#)

geGetMarkerDisplayText

```
geGetMarkerDisplayText(  
    d_figId  
)  
=> t_string / nil
```

Description

Displays the text associated with a marker.

Arguments

<i>d_figId</i>	Database ID of the marker.

Value Returned

<i>t_string</i>	The text associated with the marker, if the marker text is found.
<i>nil</i>	The marker text is not found.

Examples

Returns the text associated with `marker2`.

```
geGetMarkerDisplayText( marker2 )
```

Related Topics

[Marker Functions](#)

geGetMarkerOwner

```
geGetMarkerOwner(  
    d_figId  
)  
=> t_string / nil
```

Description

Displays the owner of a marker.

Arguments

<i>d_figId</i>	Database ID of the marker.

Value Returned

<i>t_string</i>	The name of the marker owner.
<i>nil</i>	The marker is not found.

Examples

Returns DLE, the name of the marker owner.

```
geGetMarkerOwner( d_figId )
```

Related Topics

[Marker Functions](#)

geGetMarkerReason

```
geGetMarkerReason(  
    d_dbobjectId  
)  
=> t_reason / nil
```

Description

Displays the reason for a marker.

Arguments

<i>d_dbobjectId</i>	Database ID of the marker.

Value Returned

<i>t_reason</i>	The marker reason.
<i>nil</i>	The marker is not found.

Examples

Returns the reason for marker2.

```
geGetMarkerReason( marker2 )
```

Related Topics

[Marker Functions](#)

geGetMarkerType

```
geGetMarkerType (
    d_figId
)
=> t_type / nil
```

Description

Displays the type of a marker.

Arguments

<i>d_figId</i>	Database ID of the marker.

Value Returned

<i>t_type</i>	The marker type, either warning or error.
<i>nil</i>	The marker is not found.

Examples

Returns the marker type of marker2.

```
geGetMarkerType ( marker2 )
```

Related Topics

[Marker Functions](#)

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

<i>d_cellview</i>	Database ID of the cellview queried.
<i>l_owner</i>	Owner of the markers to query. Valid Values: any marker owner
<i>l_type</i>	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.

Examples

Checks whether *cell12* contains any markers owned by PDV of the type *error*.

```
geHasMarker( cell12 "PDV" "error" )
```

Related Topics

[Marker Functions](#)

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.

Examples

```
geHiCommonFindMarker()
```

Related Topics

[Marker Functions](#)

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.

Examples

```
geHiDeleteAllMarker()
```

Related Topics

[Marker Functions](#)

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.

Examples

```
geHiDeleteMarker()
```

Related Topics

[Marker Functions](#)

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.

Examples

```
geHiExplainMarker()
```

Related Topics

[Marker Functions](#)

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.

Examples

```
geHiFindMarker()
```

Related Topics

[Marker Functions](#)

geIsMarkerShape

```
geIsMarkerShape(  
    d_dbobjectId  
)  
=> t / nil
```

Description

Checks whether a shape was created as a marker.

Arguments

<i>d_dbobjectId</i>	Database ID of the cellview containing the shape.

Value Returned

<i>t</i>	The shape is a marker.
<i>nil</i>	The shape is not a marker.

Examples

Checks if *shape2* is a marker.

```
geIsMarkerShape( shape2 )
```

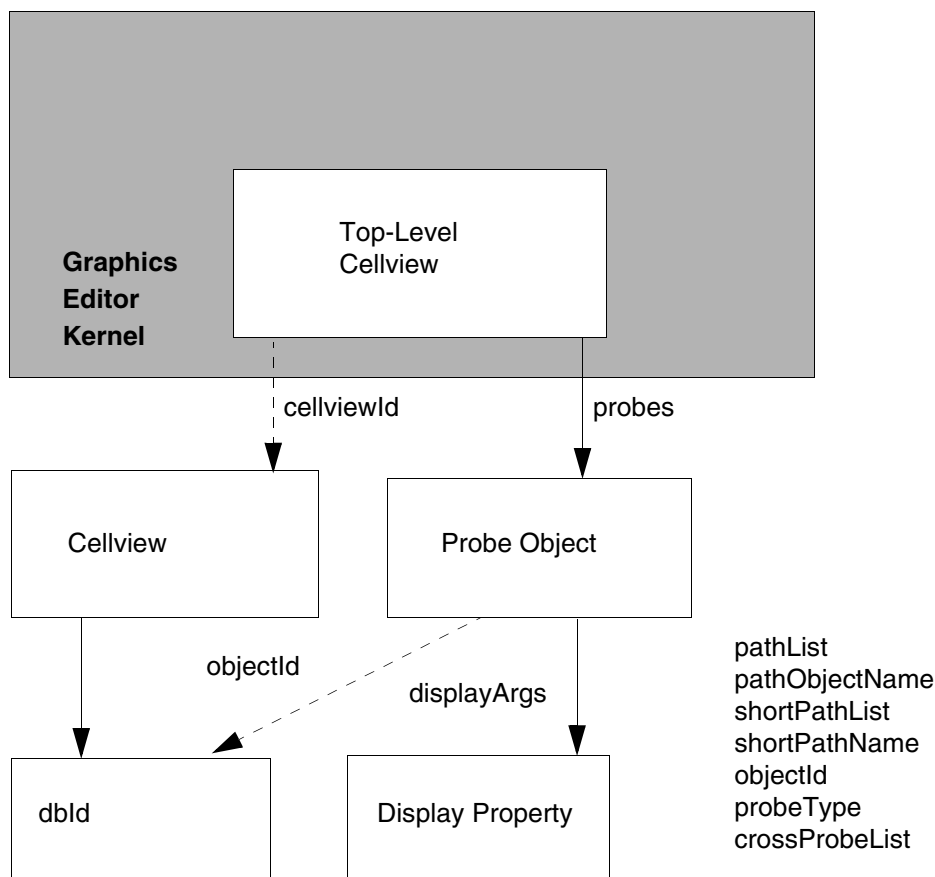
Related Topics

[Marker Functions](#)

Probe Functions

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.

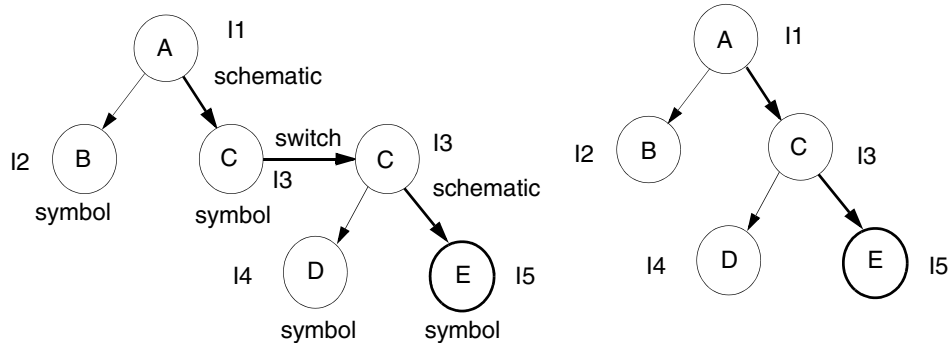
The following is a graphic description of the logical data model.



Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

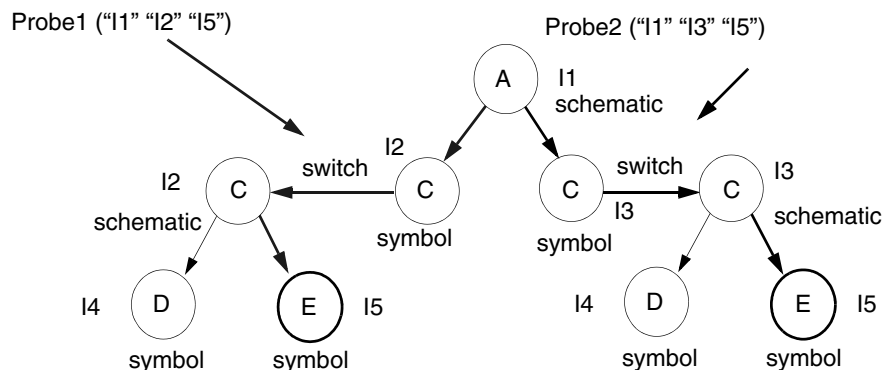
The path to the probe is an important concept. Two probes are the same if the *memInst* text string path to the probes are equal and if the top-level cellviews of each probe are the same.



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).



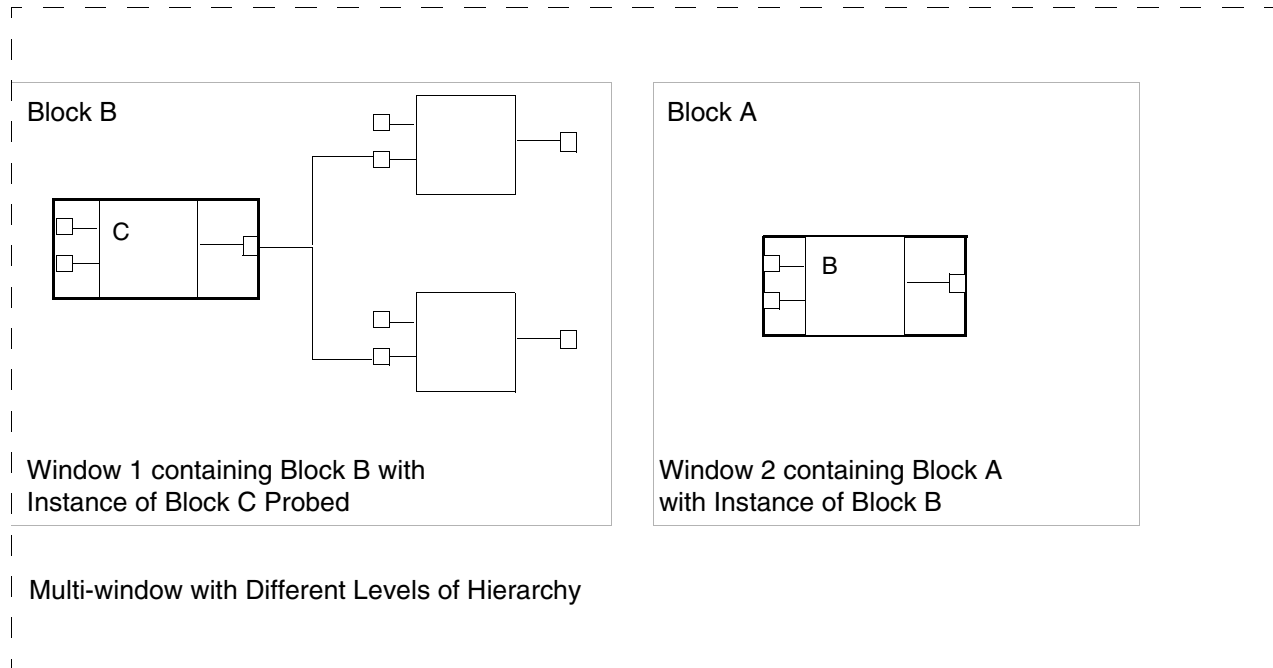
Probe1 and Probe2 are not equal even though the same physical object was probed because the paths to the object are different.

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

than you are currently displaying, the instance containing the object probed is highlighted so you know which instances contain probes.



Related Topics

[Probe Types](#)

[Probe Object](#)

[Probe Function SKILL Interface](#)

[Probe File Format](#)

[geAddInstProbe](#)

[geEnterAddInstProbe](#)

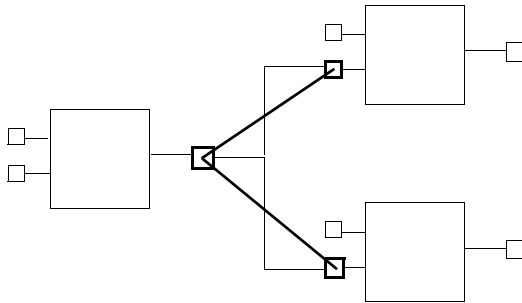
[geEnterDeleteInstProbe](#)

Probe Types

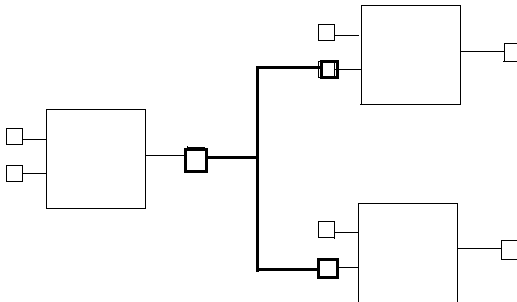
There are four types of probes: net, instance, path, and terminal.

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.



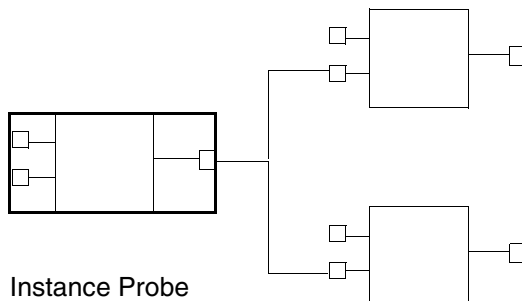
Net Probe with Flight line Display



Net Probe with Object Display

Instance Probe

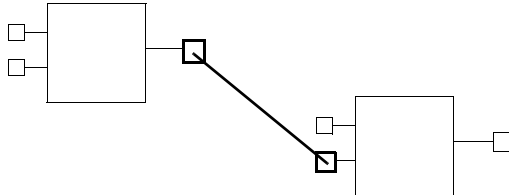
The instance probe highlights the user-specified instance. When you are at a level above this instance, the instances in all equivalent windows containing the chosen instance are highlighted to show that they contain a probe.



Instance Probe

Path Probe

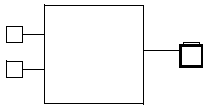
The path probe draws a flight line between two user specified terminals. These terminals do not have to be on the same net. This also means that the terminals do not have to be physically connected.



Path Probe

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

Related Topics

[Probe Functions](#)

[Probe Object](#)

[Probe Function SKILL Interface](#)

Probe Object

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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.

The 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
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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

- 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.
- 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.

Related Topics

Probe Functions

Probe Types

Probe Function SKILL Interface

Probe Function SKILL Interface

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*, *probeArgType*, *probeColor*, *probeLabel*, *probeLabelColor*, *probeLabelOffset*, *probeLabelFont*, *probeLabelFontSize*, *probeLabelJust*, *probeLabelOrient*, *probeOnColorWhenDisplay*, *probeDisplayStyle*.

Related Topics

Probe Functions

Probe Object

Probe Types

Probe File Format

There is a probe file format written and read by the `geSaveProbe` and `geLoadProbe` functions, respectively. In addition, an application you integrate into the Cadence framework

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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")
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
(probeType "instance" pathList "/I0" pathObjectName "I2"
 probeArgType "probeColor")
(probeType "net" pathObjectName "I0.Y")
(probeType "net" pathObjectName "I1.Y" probeArgType
 "probeColor" probeColor ("y4" "drawing"))
(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)
```

Related Topics

Probe Functions

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>w_windowId</i>	Database ID of the window in which to add the probe. The current window is used if <i>w_windowId</i> is defaulted.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.
<i>lt_probeSpec</i>	Location in user units to probe at, or a string describing the probed object.

Value Returned

<i>o_probe</i>	The probe object created.

Examples

Prompts you for a point, and creates an instance probe in `window(3)` using layer (`y0` drawing).

```
m = geAddInstProbe( window(3) list( "y0" "drawing" ) )
```

Related Topics

[Probe Functions](#)

[geSetProbeUserLPP](#)

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.

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 display of the net probe is affected by the net stop level.

You can select the specific bit of a bus by specifying the probePromptBusBit environment variable.

Arguments

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is defaulted.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.
<i>lt_probeSpec</i>	Location in user units to probe at, or a string describing the probed object.

Value Returned

<i>o_probe</i>	Probe object created.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Examples

Prompts for a point and creates a net probe in `window(3)` using layer `y0` drawing.

```
m = geAddNetProbe( window(3) list( "y0" "drawing" ) )
```

Related Topics

Probe Functions

geSetProbeUserLPP

geAddNetProbeFromTo

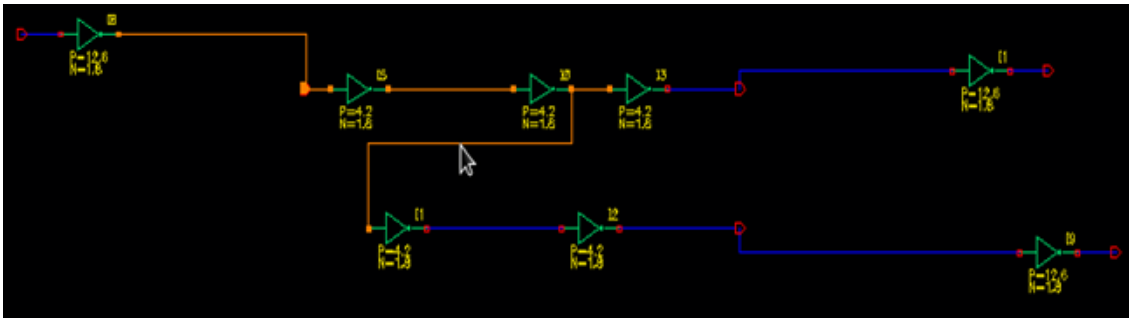
```
geAddNetProbeFromTo(  
    [ w_windowId ]  
    [ l_lpp ]  
)  
=> t / nil
```

Description

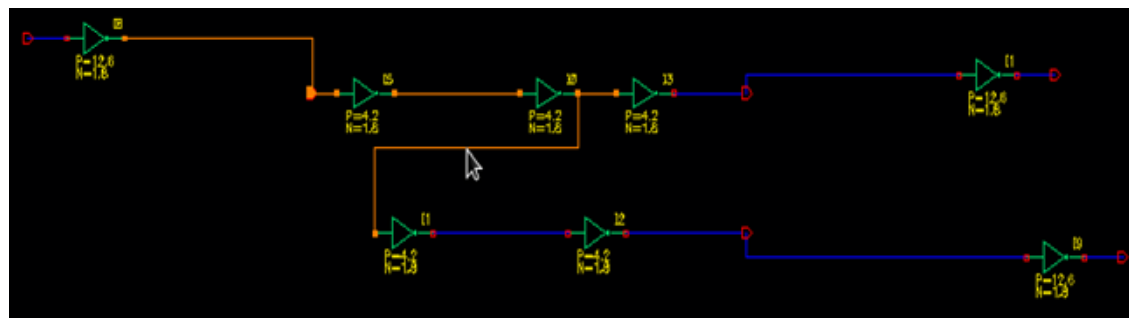
Adds a probe between two points where there is a connection between two different nets, such as nets separated by feedthrough or nets on different hierarchy but physically connected by pins.

The figure below depicts the impact on inverter 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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>w_windowId</i>	Database ID of the window in which to add the probe.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.

Value Returned

<i>t</i>	The probe was created.
<i>nil</i>	The probe was not created.

Examples

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 is created and printed on the CIW.

```
geAddNetProbeFromTo()
```

Related Topics

[Probe Functions](#)

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 *"/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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is defaulted.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.
<i>lt_probeSpec1</i>	Location in user units to look for the first terminal of the path at, or a string describing the first terminal.
<i>lt_probeSpec2</i>	Location in user units at which to look for the second terminal of the path, or a string describing the second terminal.

Value Returned

<i>o_probe</i>	Probe object created.

Examples

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.

```
m = geAddPathProbe( window(3) )
```

Related Topics

[Probe Functions](#)

[geSetProbeUserLPP](#)

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.

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 *"/instance/instance/ ... /objectName."* If *lt_probeSpec* is not specified, you are prompted to select a terminal to probe.

Arguments

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.
<i>lt_probeSpec</i>	Location in user units to probe at, or a string describing the probed object.

Value Returned

<i>o_probe</i>	The probe object created.

Examples

Prompts for a point and creates a terminal probe in *window(3)* using the next layer in the probe layer rotation sequence.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
m = geAddTermProbe( window(3) )
```

Related Topics

Probe Functions

geSetProbeUserLPP

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

<i>d_cellview</i>	Database ID of the cellview for which the probe net filter settings will be cleared.

Value Returned

<i>t</i>	The probe net name filter settings were cleared for the specified cellview ID.
<i>nil</i>	The command failed.

Examples

Clears the probe net filter previously saved in the current cellview.

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetProbeNetFilter(cellView "Hide" list("VDD*" "VSS*"))
geSaveProbeNetFilter(cellView)
geGetProbeNetFilter(cellview t)
geClearProbeNetFilter(cellview)
geGetProbeNetFilter(cellview t)
```

Related Topics

Probe Functions

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

<i>w_windowId</i>	Database ID of the window from which to remove probes.
<i>g_crossProbeFlag</i>	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.

Examples

Deletes all probes in the current window.

```
geDeleteAllProbe( hiGetCurrentWindow( ) )
```

Related Topics

[Probe Functions](#)

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for an instance probe to delete, or a string describing the instance probe to delete.

Value Returned

<i>t</i>	An instance probe was found and deleted.
<i>nil</i>	An instance probe was not found and deleted.

Examples

Prompts for a point and deletes the instance probe under that point.

```
m = geDeleteInstProbe( window(3) )
```

Related Topics

[Probe Functions](#)

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for a net probe to delete, or a string describing a net probe to delete.

Value Returned

<i>t</i>	A net probe was found and deleted.
<i>nil</i>	A net probe was not found and deleted.

Examples

Deletes the net probe associated with NET10 in instance I1 of the top level of *window(3)*.

```
m = geDeleteNetProbe( window(3) "/I1/NET10" )
```

Related Topics

[Probe Functions](#)

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for a path probe to delete, or a string describing a path probe to delete.

Value Returned

<i>t</i>	A probe was found and deleted.
<i>nil</i>	A probe was not found and deleted.

Examples

Prompts for a point and deletes the path probe under that point in `window(3)`.

```
m = geDeletePathProbe( window(3) )
```

Related Topics

[Probe Functions](#)

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

<i>w_windowId</i>	Database ID of the window from which to remove the probe.
<i>o_userType</i>	Handle to the probe object.
<i>g_crossProbeFlag</i>	Indicates if the crossProbes associated with the probe should also be deleted. Valid Values: <i>t</i> , <i>nil</i> Default Value: <i>nil</i>

Value Returned

<i>t</i>	The probe is removed.
<i>nil</i>	The probe is not removed.

Examples

The probe is deleted from the current window.

```
geDeleteProbe( getCurrentWindow( ) probe )
```

Related Topics

[Probe Functions](#)

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for a terminal probe to delete, or a string describing a terminal probe to delete.

Value Returned

<i>t</i>	A probe was found and deleted.
<i>nil</i>	A probe was not found and deleted.

Examples

Prompts for a point and deletes the terminal probe under that point in `window(3)`.

```
geDeleteTermProbe( window(3) )
```

Related Topics

[Probe Functions](#)

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

<i>w_windowId</i>	Database ID of the window in which to add the probe. The current window is used if <i>w_windowId</i> is defaulted.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.
<i>lt_probeSpec</i>	Location in user units to probe at, or a string describing the probed object.
<i>g_repeatCmd</i>	SKILL expression specifying the probe selection mode, single select versus multiple post-select.

Value Returned

<i>o_probe</i>	The probe object created.
<i>l_probes</i>	A list of all the probes in <i>w_windowId</i> .

Examples

Prompts you for a point and creates an instance probe in the current window using layer *y0* drawing.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
m = geEnterAddInstProbe( hiGetCurrentWindow() list( "y0" "drawing" ) nil t )
```

Related Topics

Probe Functions

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.

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is defaulted.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.
<i>lt_probeSpec</i>	Location in user units to probe at, or a string describing the probed object.
<i>g_repeatCmd</i>	SKILL expression specifying the probe selection mode, single select versus multiple post-select.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Value Returned

<i>o_probe</i>	Probe object created.
<i>l_probes</i>	A list of all the probes in <i>w_windowId</i> .

Examples

```
m = geEnterAddNetProbe( hiGetCurrentWindow() list( "y0" "drawing" ) nil t )
```

Related Topics

Probe Functions

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 “/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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>l_lpp</i>	Layer-purpose pair on which to display the probe.
<i>lt_probeSpec</i>	Location in user units to probe at, or a string describing the probed object.
<i>g_repeatCmd</i>	SKILL expression specifying the probe selection mode, single select versus multiple post-select.

Value Returned

<i>o_probe</i>	Probe object created.
<i>l_probes</i>	A list of all the probes in <i>w_windowId</i> .

Examples

```
m = geEnterAddTermProbe( hiGetCurrentWindow() list( "y0" "drawing" ) nil t )
```

Related Topics

Probe Functions

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for an instance probe to delete, or a string describing the instance probe to delete.
<i>g_repeatCmd</i>	SKILL expression specifying the probe selection mode, single select versus multiple post-select.

Value Returned

<i>t</i>	An instance probe was found and deleted.
<i>nil</i>	An instance probe was not found and deleted.

Examples

Prompts for a point and deletes the instance probe under that point.

```
m = geEnterDeleteInstProbe( hiGetCurrentWindow() nil t)
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Probe Functions

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for a net probe to delete, or a string describing a net probe to delete.
<i>g_repeatCmd</i>	SKILL expression specifying the probe selection mode, single select versus multiple post-select.

Value Returned

<i>t</i>	A net probe was found and deleted.
<i>nil</i>	A net probe was not found and deleted.

Examples

```
m = geEnterDeleteNetProbe( hiGetCurrentWindow() nil t )
```

Related Topics

Probe Functions

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

<i>w_windowId</i>	Database ID of the window to probe. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for a terminal probe to delete, or a string describing a terminal probe to delete.
<i>g_repeatCmd</i>	SKILL expression specifying the probe selection mode, single select versus multiple post-select.

Value Returned

<i>t</i>	A probe was found and deleted.
<i>nil</i>	A probe was not found and deleted.

Examples

Deletes the terminal probe from the current window.

```
geEnterDeleteTermProbe( hiGetCurrentWindow() nil t )
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Probe Functions

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.

The information for the file shown is put into a temporary file in */tmp*.

Arguments

<i>w_windowId</i>	Database ID of the window containing the probes you want explained. The current window is used if <i>w_windowId</i> is not specified.
<i>lt_probeSpec</i>	Location in user units to look at for probes to explain, or a string describing a probe to explain.

Value Returned

<i>t</i>	A probe was found and explained.
<i>nil</i>	A probe was not found and explained.

Examples

Explains any probes under the point 10:10 in *window(3)*.

```
geExplainProbe( window(3) 10:10 )
```

Related Topics

Probe Functions

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

<i>w_winId</i>	Window ID used to find the top-most point of the hierarchy for name reduction.
<i>t_pathList</i>	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.
<i>d_cellview</i>	Database Id of a cellview to be used as the top-level cellview specified by <i>w_winId</i> . If this value is specified, the minimized name is relative to this cellview, rather than the topmost cellview. Specifying the value returned by <code>geGetTopCellViewName</code> as this third argument is the same as omitting the third argument.
<i>t_viewnameList</i>	A string specifying views to switch into when descending the hierarchy specified in <i>t_pathList</i> . If not specified, the default value <code>schematic</code> is used.

Value Returned

<i>t_adjustedPathList</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Examples

Assuming the currently displayed cellview has a net called `gnd`, the following example 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"`.

```
netpath = buildString(foreach(mapcar i
geGetHierMemInst(hiGetCurrentWindow()) car(i)->name) "/" )
netPath = strcat("/", netPath "/" "gnd")
geGetAdjustedPath( hiGetCurrentWindow() netPath)
=> "/gnd"
```

Related Topics

Probe Functions

geGetAllProbe

```
geGetAllProbe(  
    [ w_windowId ]  
)  
=> l_probes / nil
```

Description

Gets a list of all the probes in a window.

Arguments

<i>w_windowId</i>	Database ID of the window containing the probes.

Value Returned

<i>l_probes</i>	A list of all the probes in <i>w_windowId</i> .
<i>nil</i>	There are no probes in the window.

Examples

Gets a list of all the probes in the current window.

```
probe_list = geGetAllProbe( GetCurrentWindow( ) )
```

Related Topics

[Probe Functions](#)

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

<i>t_viewTypeName</i>	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: <code>all</code> , <code>none</code>

Value Returned

<i>g_value</i>	The cross probe value.
<code>nil</code>	The cross probe value is not set.

Examples

Returns `t`.

```
geGetCrossProbeValue( "graphic" )
```

Related Topics

[Probe Functions](#)

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

<i>d_cellview</i>	Database ID of the cellview.
<i>g_fromDesign</i>	When <code>t</code> , it returns the net filtering settings for probe that are currently saved in the design.

Value Returned

<i>l_filterSettings</i>	The net filtering settings of the Highlight Options form.
<i>nil</i>	The net name filtering settings were not found.

Examples

In the following example, only `net1` is specified to be filtered in the Highlight Options form.

```
window = hiGetCurrentWindow()  
cellView = geGetEditCellView(window)  
netNameFilters = geGetProbeNetFilter(cellView)  
netNameFilters  
(nil mode "Hide" netNames ("VDD" "VSS"))  
netNameFilters->mode  
"Hide"  
netNameFilters->netNames  
("net1")
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Probe Functions

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.

Examples

Returns 20.

```
myValue = geGetProbeNetStopLevel( )
```

Related Topics

[Probe Functions](#)

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

<code>l_lpp</code>	The layer-purpose pair to be used for probing operations.
<code>nil</code>	No layer-purpose pair exists.

Examples

Returns ("marker" "error")

```
geGetProbeUserLPP( )
```

Related Topics

[Probe Functions](#)

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

<i>w_windowId</i>	Database ID of the window.

Value Returned

t	The Load Probes form is displayed.

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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

<i>w_windowId</i>	Database ID of the window.

Value Returned

t	The Save Probes form is displayed.

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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

<code>w_windowId</code>	Database ID of the window containing the probes you want loaded.
<code>p_port</code>	Port to load the probe information through.
<code>s_prop</code>	Property to be examined to decide whether to load probe.
<code>s_cond</code>	Condition that must be satisfied if the probe is to be loaded.
<code>g_value</code>	Value of the property tested against the condition to decide whether to load the probe.
<code>s_loadCount</code>	Variable containing the index of the probe to load.

Value Returned

<code>t</code>	A probe was found and loaded.
<code>nil</code>	A probe was not found and loaded.

Examples

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
m = infile( "/tmp/myProbes.out" )
ct = 0
geLoadNextProbe( window(3) m 'myProp 'equal "critical" 'ct)
```

Related Topics

[Probe Functions](#)

[geLoadProbe](#)

[Probe File Format](#)

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

<i>w_windowId</i>	Database ID of the window containing the probes you want loaded. If not specified, the current window is used.
<i>p_port</i>	Port to load the probe information through. If <i>p_port</i> is not specified, the probes are read from the file <code>probe.out</code> .
<i>s_prop</i>	Property to be examined in deciding whether to load probe.
<i>s_cond</i>	Condition that must be satisfied if the probe is to be loaded.
<i>g_value</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Examples

Loads all probes in file `/tmp/myProbes.out` that have the user-defined property `myProperty` equal to the string `critical`.

```
m = infile( "/tmp/myProbes.out" )
geLoadProbe( window(3) m 'myProperty 'equal "critical")
```

Related Topics

[Probe Functions](#)

[geUnRegCrossProbeTrigger](#)

[Probe File Format](#)

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

<i>?window w_windowId</i>	Database ID of the window to contain the probe.
<i>?name t_name</i>	Name of the probe.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

`?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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Value Returned

o_probeObject Database ID of the new probe object.

Examples

The following example 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( "y0" "drawing")  
?object list( "I1" "I2")  
?probeType "instance")
```

The following example 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.

```
probe = geMakeProbeWithColor(?name "myProbe"  
?color list( "marker" "error")  
?object list( "/I1/I5/NET10")  
?probeType "net")
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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. The default location of the label is calculated based on the type of 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 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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

?message *t_messageString*

String passed to the C-level *crossProbe* functions.

?labelName *t_labelName*

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 the font to use for label.

?labelFontSize *f_labelFontSize*

Height in user units for the probe label.

?labelJust *t_labelJust*

Justification of the probe label.

?labelOrient *t_labelOrient*

Orientation of the probe label.

Value Returned

o_probeObjectId Database ID of the new probe object.

Examples

The following example probes terminal A1 on instance I2 of the current window.

```
probe = geMakeProbeWithLabel(  
  ?window getCurrentWindow( )  
  ?name "probe2"  
  ?object list("I2" "A1")  
  ?labelName "labelName"  
  ?labelColor list("y0" "drawing")  
  ?probeType "terminal"  
  ?displayStyle "object"  
)
```

The following example has the same effect as the above example.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

```
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")
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

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.

Value Returned

o_probe_objectId Database ID of the new probe object.

Examples

Probes instance I2 contained within instance I1 within the top level of the current window.
The probe is drawn on layer y0 drawing.

```
probe = geMakeProbeWithOnColor(?name "myProbe"  
?onColor list( "y0" "drawing")  
?object list( "I1" "I2")  
?probeType "instance")
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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 be 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

<code>?window w_windowId</code>	Database ID of the window to contain the probe.
<code>?name t_name</code>	Name of the probe.
<code>?object g_object</code>	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 <i>memInst</i> 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 <code>w_windowId</code> to the object.
<code>?otherObject g_otherObject</code>	Second object specified only for path probes, which require two terminal objects.
<code>?probeType t_probeType</code>	Type of the probe. Valid Values: net, path, terminal, instance.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

?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

Probes instance `I2` contained within instance `I1` within the top level of the current window. The probe is drawn on layer `y0` drawing.

```
probe = geMakeProbeWithoutColor(  
  ?name "myProbe"  
  ?object list( "I1" "I2")  
  ?probeType "instance")
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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

<i>w_windowId</i>	Database ID of the window from which you want a list of probes. The default is the current window.
<i>p_port</i>	SKILL port printed out to. Default: <i>p_port</i>

Value Returned

<i>t</i>	The list of probes is created.
<i>nil</i>	There are no probes in the window.

Examples

Prints the attributes of all probes in the current window to a text file called `myFile` in the current directory.

```
port = outfile( "myFile")  
gePrintAllProbe( hiGetCurrentWindow( ) port)
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

geProbeP

```
geProbeP(  
    o_probe  
)  
=> t / nil
```

Description

Checks if the value passed in is a probe.

Arguments

<i>o_probe</i>	Handle to a probe object.

Value Returned

<i>t</i>	The object passed in is a probe.
<i>nil</i>	The object passed is not a probe.

Examples

```
if( geProbeP( myObject) then  
warn( "Object %L is not a probe\n" myObject ) )
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

geRegCrossProbeTrigger

```
geRegCrossProbeTrigger(  
    g_ilFunc  
    x_priority  
    t_name  
)  
=> t / nil
```

Description

Registers a trigger function for cross probing.

Arguments

<i>g_ilFunc</i>	The symbol of the client's cross probe trigger function.
<i>x_priority</i>	An integer representing the trigger priority.
<i>t_name</i>	A string used for identification only.

Value Returned

<i>t</i>	Registers a trigger function for cross probing.
<i>nil</i>	Does not register a trigger function.

Examples

Registers the trigger function `myXProbeTrigger` as priority 1 for cross probing.

```
geRegCrossProbeTrigger('myXProbeTrigger 1 "DEMO")
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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

<i>w_windowId</i>	Database ID of the window containing the probes you want saved. If not specified, the current window is used.
<i>p_port</i>	Port to save the probe information through. If <i>p_port</i> is not specified, the probes are stored into the file <code>probe.out</code> .

Value Returned

<i>t</i>	A probe was found and saved.
<i>nil</i>	A probe was not found and saved.

Examples

Writes out all probes in `window(3)` to the file `/tmp/myProbes.out`.

```
m = outfile( "/tmp/myProbes.out" )  
geSaveProbe( window(3) m )
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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.

This function can be used only if the cellview is editable.

Arguments

<code>d_cellview</code>	Database ID of the cellview.

Value Returned

<code>t</code>	The net filtering is saved.
<code>nil</code>	The net filtering is not saved.

Examples

Saves the probe net filtering settings, where probe highlights for VDD and VSS nets are not shown.

```
window = hiGetCurrentWindow()
cellView = geGetEditCellView(window)
geSetProbeNetFilter(cellView "Hide" list("VDD" "VSS"))
geSaveProbeNetFilter(cellView)
```

Related Topics

Probe Functions

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

<i>t_viewTypeName</i>	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 <i>all</i> or <i>none</i> .
<i>g_value</i>	Value assigned to crossProbe.

Value Returned

<i>t</i>	Returns <i>t</i> by default.
<i>nil</i>	If <i>t_viewTypeName</i> is <i>all</i> or <i>none</i> , this value is ignored.

Examples

Set to cross probe `graphic` view types by default.

```
geSetCrossProbeValue( "graphic" t )
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

geSetLimitedProbe

```
geSetLimitedProbe(  
    o_probeObject  
    t_dir  
)  
=> t / nil
```

Description

Sets the specified probe as a directed probe.

Arguments

<i>o_probeObject</i>	Database ID of the probe.
<i>t_dir</i>	An integer representing the direction. Valid Values: 0-inputOutput; 1-input; 2-output

Value Returned

<i>t</i>	The probe is set as a directed probe.
<i>nil</i>	The probe is not set as a directed probe.

Examples

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)  
            )  
        )  
    )  
)
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Probe Functions

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	The maximum number of probes could not be set.

Examples

Sets the maximum number of probes to 1000.

```
geSetMaxCreatedProbe(1000)
```

Related Topics

[Probe Functions](#)

geSetProbeDirection

```
geSetProbeDirection(  
    o_probeObject  
    t_dir  
)  
=> t / nil
```

Description

Sets the specified probe as a directed probe.

Arguments

<i>o_probeObject</i>	Database ID of the probe.
<i>t_dir</i>	An integer representing the direction. Valid Values: 0-inputOutput; 1-input; 2-output

Value Returned

<i>t</i>	The probe is set as a directed probe.
<i>nil</i>	The probe is not set as a directed probe.

Examples

The following example depicts how to probe the first net within the current window, edited cellview and define this probe as a Fadeln 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)  
            )  
        )  
    )  
)
```


Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Probe Functions

geSetProbeNetFilter

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

Description

Sets a net filtering for probe on the given cellview.

Arguments

<i>d_cellview</i>	Database ID of the cellview on which to set the filter settings.
<i>t_mode</i>	The mode, <code>Hide</code> or <code>Show</code> .
<i>l_netNames</i>	The list of net names handled by Highlight Options filtering. If the <i>t_mode</i> is set to <code>Show</code> , during the next redraw only net probes that have nets in the provided list of net names are displayed. If <i>t_mode</i> is set to <code>Hide</code> , on the next redraw corresponding net probes will not be displayed.

Value Returned

<i>t</i>	The probe net filtering is set.
<i>nil</i>	The probe net filtering is not set.

Examples

Sets net filtering to display probes only for VDD and VSS nets.

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

Related Topics

Probe Functions

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.

Examples

Causes all net probes to be displayed to the lowest level of hierarchy.

```
geSetProbeNetStopLevel( 20 )
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

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

<code>l_lpp</code>	Layer-purpose pair to use for successive probing operations.

Value Returned

<code>t</code>	The layer was set.
<code>nil</code>	The layer was not set.

Examples

Sets the default probe layer to be the `error marker` layer.

```
geSetProbeUserLPP( list( "marker" "error" ) )
```

Related Topics

[Probe Functions](#)

[Probe File Format](#)

geUnRegCrossProbeTrigger

```
geUnRegCrossProbeTrigger(  
    s_ilFunc  
)  
=> t / nil
```

Description

Removes the cross probe trigger, if one is registered.

Arguments

<i>s_ilFunc</i>	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.

Related Topics

[Probe Functions](#)

Icon Creation and Loading Functions

You can use SKILL functions to create and load icons from a SKILL file. You can also save a cellview as a SKILL icon file.

Related Topics

[geCellViewToDlist](#)

[geLoadIcon](#)

[geSaveIcon](#)

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

<i>d_cellview</i>	Database ID of the cellview whose contents are to be represented as a list.

Value Returned

<i>l_dlist</i>	Returns a list representing the contents of <i>d_cellview</i> .
<i>nil</i>	Returns <i>nil</i> if the cellview contains no objects.

Examples

Returns a list made from the objects in the cellview being edited in `window(2)`.

```
geCellViewToDlist( window(2)->editCellView )
```

Related Topics

[Icon Creation and Loading Functions](#)

[geLoadIcon](#)

[geSaveIcon](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>t_iconName</i>	Name given to the icon.
<i>t_fileName</i>	File in which the icon data is to be found.
<i>x_width</i>	Pixel width of the new icon.
<i>x_height</i>	Pixel height of the new icon.
<i>x_bgColor</i>	Background pixel value of the new icon.

Value Returned

<i>t</i>	The icon was successfully loaded.
<i>nil</i>	The icon was not loaded successfully.

Examples

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.

```
geLoadIcon( "myIcon" "myIconFile" 40 40 2 )
```

Related Topics

[Icon Creation and Loading Functions](#)

[geCellViewToDlist](#)

[geSavelcon](#)

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

<i>w_windowId</i>	Database ID of the window containing the cellview to be saved as an icon file.
<i>t_iconName</i>	Name given to the icon.
<i>t_fileName</i>	File in which to save the icon data.

Value Returned

t	The icon was successfully written.
nil	The icon was not written successfully.

Examples

Writes icon data derived from the edit cellview in `window(2)` to the file `myIconFile` and names the icon `myIcon`.

```
geSaveIcon( window(2) "myIcon" "myIconFile" )
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Icon Creation and Loading Functions

geCellViewToDlist

geLoadIcon

Utility Functions

You can use the utility functions to perform tasks such as:

- Open a design in a new window, verify the database ID of a window, and create a copy of a cellview.
- Display the properties of each object in the selected set.
- Determine whether a path or polygon is self-intersecting.
- Search a point list in a cellview for collinear points and delete the intermediate collinear points.
- Create or update a DFII display packet.

Related Topics

Icon Creation and Loading Functions

geAskOverwriteMode

geClosePolygon

geCompressPointArray

geCoordp

geCreatePacket

geGetInstTransform

geGetLockInfo

geInstIdToFileSpec

gelsPathReentrant

gelsPolygonReentrant

geOpen

geQuerySelSet

geResetCellView

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

geTransformUserBBox

geTransformUserPoint

geValidateWindow

geWriteCellView

geAskOverwriteMode

```
geAskOverwriteMode(  
    d_cellviewId  
)  
=> t / nil
```

Description

Prompts the user to determine whether or not to overwrite the already open cellview.

Arguments

<code>d_cellviewId</code>	Database ID of the cellview.

Value Returned

<code>t</code>	The cellview is overwritten.
<code>nil</code>	The cellview is not overwritten.

Examples

Asks the user whether or not to overwrite the cellview `cell13` when the cellview is already open.

```
geAskOverwriteMode( cell13 )
```

Related Topics

[Utility Functions](#)

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

<code>d_cellview</code>	Database ID of the cellview to contain the polygon.
<code>l_points</code>	List of points used to create the polygon.
<code>t_snapMode</code>	Specifies how the cursor snaps when a polygon is created. For example, some possible snap modes are <code>orthogonal</code> , <code>diagonal</code> , or <code>L90YFirst</code> .

Value Returned

<code>l_pointsPolygon</code>	Returns a list of points that describe a resulting closed polygon.
<code>nil</code>	Returns <code>nil</code> when a closed polygon could not be created.

Examples

```
geClosePolygon(geGetEditCellView() list(0:5 1:6 1:6) "orthogonal")  
=> ((1.0 6.0)  
    (1.0 5.0)  
    (0.0 5.0)  
    )
```

Related Topics

Utility Functions

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

<i>d_cellview</i>	Database ID of the cellview to search.
<i>l_points</i>	List of points to search.
<i>g_closedList</i>	Either <code>t</code> or <code>nil</code> to specify whether the list is a closed list or not.

Value Returned

<i>l_pointList</i>	A list of coordinates representing the compressed point array.
<code>nil</code>	The intermediate collinear points are not deleted.

Examples

Deletes the vertex at the point 10:0 from the closed list in the cellview returned by `geGetEditCellView`.

```
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))
```

Related Topics

Utility Functions

geCoordp

```
geCoordp(  
    coord  
)  
=> t / nil
```

Description

Determines whether the data supplied by *coord* is a point coordinate.

Arguments

<i>coord</i>	Data to check. <i>coord</i> 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

The following example returns t.

```
geCoordp(10:10)
```

The following example returns nil.

```
geCoordp("myCell")
```

Related Topics

[Utility Functions](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>w_window</i>	Window used to get the display.
<i>t_name</i>	The name of the display packet to be created.
<i>t_stippleName</i>	The stipple name of the packet.
<i>t_lineStyleName</i>	The line style name of the packet.
<i>t_fillColorName</i>	The fill color name of the packet.
<i>t_outlineColorName</i>	The outline color name of the packet.
<i>d_lineThickness</i>	The line thickness of the packet.
<i>g_reuse</i>	When false, erases and re-creates an existing packet.
<i>t_fillStyleName</i>	The fill style name of the packet.

Value Returned

<i>t</i>	The display packet was created or updated.
<i>nil</i>	The display packet was not created or updated.

Examples

```
geCreatePacket(window(3) "pkt1" "solid" "solid" "green" "green" 4 nil)
```

Related Topics

[Utility Functions](#)

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. Cadence recommends you to use the `dbGetInstTransform` function instead.

Arguments

<code>d_InstId</code>	This can be a single <i>instId</i> , or a list of <i>instIds</i> in order from top to bottom which describes a hierarchical path to an instance at a lower level.

Value Returned

<code>xOffset yOffset</code>	The float values of the x and y offsets.
<code>rotation</code>	A string that is the value of the rotation.
<code>magnification</code>	The amount of magnification as a float.

Examples

```
geGetInstTransform(  
list(topInstId middleInstId bottomInstId))  
=> ((-3.1 0.9) "R0" 1.0)
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Utility Functions

geGetLockInfo

```
geGetLockInfo(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> t_userHostInfo / no data on lock available
```

Description

Returns the user and host from a .cdslock file.

Arguments

<i>t_libName</i>	String value, library name.
<i>t_cellName</i>	String value, cell name.
<i>t_viewName</i>	String value, view name.

Value Returned

<i>t_userHostInfo</i>	Returns the user and host information successful.
no data on lock available	No data is obtained from the .cdslock file for a cellview.

Examples

```
geGetLockInfo("myLib" "myCell" "myView")  
"user : yourName\nhost : trnsoph6"
```

Related Topics

Utility Functions

geInstIdToFileSpec

```
geInstIdToFileSpec(  
    d_instId  
    t_mode  
)  
=> l_fileSpec
```

Description

Creates a *fileSpec*, a disassociated property list, from the data in an instance and the given mode.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Arguments

<i>d_instId</i>	Database ID of the instance.
<i>t_mode</i>	Mode used in the <i>fileSpec</i> .
Valid Values: <i>r</i> (read), <i>a</i> (append), <i>w</i> (overwrite).	

Value Returned

<i>l_fileSpec</i>	The filled-out <i>fileSpec</i> . It 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 ""))

Examples

Creates a *fileSpec* of the instance *inst3* with append mode.

```
geInstIdToFileSpec( inst3 "a" )
```

Related Topics

Utility Functions

gelsPathReentrant

```
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

<i>d_cellview</i>	Database ID of the cellview containing the path to check.
<i>l_points</i>	List of points making up the path.
<i>n_width</i>	Width of the path.
<i>t_style</i>	String defining an optional path style. If no style is given, the default is <i>truncateExtend</i> . Valid Values: <i>truncateExtend</i> , <i>extendExtend</i> , <i>roundRound</i> , <i>varExtendExtend</i>
<i>n_beginExt</i>	Beginning path extension. Default: 0.0
<i>n_endExt</i>	Ending path extension. Default: 0.0

Value Returned

<i>t</i>	The path is self-intersecting.
<i>nil</i>	The path is not self-intersecting.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Examples

The following example returns `t`.

```
geIsPathReentrant( geGetEditCellView( ) list( 0:0 10:10 0:10 10:0 ) 3.0  
"truncateExtend" 1.0 0 )
```

The following example returns `nil`.

```
geIsPathReentrant( geGetEditCellView( ) list( 0:0 10:10 0:10 3.0 )
```

Related Topics

Utility Functions

gelsPolygonReentrant

```
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

<i>d_cellview</i>	Database ID of the cellview containing the polygon to check.
<i>l_points</i>	List of polygon points.

Value Returned

<i>t</i>	The polygon is self-intersecting.
<i>nil</i>	The polygon is not self-intersecting.

Examples

The following example returns *nil*. The polygon is self-touching, but not self-intersecting.

```
geIsPolygonReentrant( geGetEditCellView( ) list( 0:0 10:0 10:5 5:5 5:0 0:5 ) )
```

The following example returns *t*. The polygon is self-intersecting.

```
geIsPolygonReentrant( geGetEditCellView( ) list( 0:0 10:0  
    10:5 5:5 5:-5 0:-5 ) )
```

Related Topics

Utility Functions

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

<i>?window w_windowId</i>	Database ID of the window in which to open the cell.
<i>?lib t_lib</i>	Name of the library containing the cell.
<i>?cell t_cell</i>	Name of the cell.
<i>?view t_view</i>	View of the cell. For example: layout, symbol, or schematic.
<i>?viewType t_viewType</i>	Type of data to open. For example: maskLayout, schematic.
<i>?mode t_mode</i>	Access mode of the cell. Valid Values: r (read), a (append)

Value Returned

<i>w_windowId</i>	Returns the ID of the window in which the design is created or is opened successfully. As a special case, geOpen returns <i>t</i> instead of the window ID. For example, when the window in which you want to open a design is not a DE window.
<i>nil</i>	Returns <i>nil</i> if a window could not be created or a design could not be opened.

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Examples

Opens the layout view of mycell in window:3 in edit mode.

```
geOpen( ?window hiGetCurrentWindow( ) ?lib "mylib"  
?cell "mycell" ?view "layout" ?viewType "maskLayout" ?mode "a" )  
=> window:3
```

Related Topics

Utility Functions

geQuerySelSet

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

Description

Displays the properties of each object in the selected set.

Arguments

<i>w_windowId</i>	Database ID of the window containing the selected set. If no <i>w_windowId</i> is specified, the current window is used.

Value Returned

<i>t</i>	Selected objects are found.
<i>nil</i>	Selected objects are not found.

Examples

Displays a properties form for each object in the selected set in `window3`.

```
geQuerySelSet( window3 )
```

Related Topics

[Utility Functions](#)

geRegSaveAllTrigger

```
geRegSaveAllTrigger(  
    trigName  
)  
=> t / nil
```

Description

Registers a trigger name for the Save All form.

Arguments

<i>trigName</i>	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.

Examples

```
geRegSaveAllTrigger( "_geiSaveAllCVTrigger" )
```

Related Topics

Utility Functions

geResetCellView

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

Description

Re-displays the last saved version of a cellview.

Arguments

<i>d_cellview</i>	Database ID of the cellview to reset.

Value Returned

t	The cellview is reset.
nil	The cellview is not reset.

Examples

Reverts to the last saved cellview of `cell2`.

```
geResetCellView( cell2 )
```

Related Topics

[Utility Functions](#)

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

<i>l_bBox</i>	Bounding box to be transformed.
<i>l_transform</i>	List the offset, orientation, and magnification of the bBox.

Value Returned

<i>l_bBox_new</i>	The bounding box resulting from transformation.

Examples

```
geTransformUserBBox(list(lowerLeft upperRight) list(Offset Orient Mag))
```

```
geTransformUserBBox(list(2:3 4:8) list(1:0 "R0" 1.1))
```

Related Topics

[Utility Functions](#)

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

<i>l_point</i>	Point to be transformed.
<i>l_list</i>	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

<i>l_point</i>	A list of points is returned if the transformation occurs.
<i>nil</i>	No transformation occurs.

Examples

```
geTransformUserPoint((1.5:2.5) list(0:10 "R90"))  
=> (-2.5 11.5)
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Utility Functions

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

<i>w_windowId</i>	Database ID of the window.

Value Returned

<i>w_windowId</i>	The window exists.
<i>nil</i>	The window does not exist.

Examples

Returns `window:123456`, the database ID of `window10`.

```
geValidateWindow( window10 )
```

Related Topics

[Utility Functions](#)

geWriteCellView

```
geWriteCellView(  
    d_cellview  
    t_libName  
    t_cellName  
    t_viewName  
    [ t_version ]  
    [ d_contextId ]  
)  
=>t / nil
```

Description

Creates a copy of a cellview.

Arguments

<i>d_cellview</i>	Database ID of the window containing the design to copy.
<i>t_libName</i>	Library name in which the cellview is created.
<i>t_cellName</i>	New cell name for the design.
<i>t_viewName</i>	New view name for the design.
<i>t_version</i>	This argument is ignored in 4.4 and 5.0, but is left in for compatibility.
<i>d_contextId</i>	Database ID of the context for the cell.

Value Returned

<i>t</i>	The cellview is written.
<i>nil</i>	The cellview is not written.

Examples

Creates a copy of the cellview `cell11` in `mylib` with the name `mycell` and the view `layout`.

```
geWriteCellView( cell11 "mylib" "mycell" "layout" )
```

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Related Topics

Utility Functions

Microwave Object Functions

You can use microwave functions to create a transmission line, transmission line bend, and tapered transmission line in a cellview.

Related Topics

[geCreateBend](#)

[geCreateTaper](#)

[geCreateTrl](#)

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

<i>d_cellview</i>	Database ID of the cellview in which the transmission line bend is created.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair. For more information, see Database Access Function Argument Types .
<i>l_points</i>	List of points used to create the transmission line bend. The point list must contain three points.
<i>f_width1</i>	Specifies the width of the first end of the bend. Valid Values: any number greater than zero in user units
<i>f_width2</i>	Specifies the width of the second end of the bend. Valid Values: any number greater than zero in user units
<i>t_bendStyle</i>	Specifies the type of corner to be use for the bend. (Refer to the <i>Virtuoso Layout Suite L User Guide</i> for descriptions and illustrations of the bend styles available.) Default Value: bend Valid Values: bend, chamfer, or radial

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

<i>f_bendFac</i>	The maximum width allowed in a standard bend before the bend is automatically chamfered. Default Value: 1 Valid Value: any number
<i>f_radFac</i>	The ratio of the centerline bend width to the transmission line width for a radial bend. Valid Value: any number
<i>f_chamFac</i>	The fraction of the corner that is removed in a chamfer bend. Valid Value: a number between 0 and 1
<i>x_nPtsToPI</i>	The number of segments in a 180-degree radial bend. Default Value: 20 Valid Value: any integer greater than 1

Value Returned

<i>d_bendId</i>	The database ID of the dbLine that represents the transmission line bend if the transmission line bend is created.
<i>nil</i>	The transmission line is not created.

Examples

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.

```
geCreateBend( "cell2" "metal1R" list(4:4 8:12 4:20) 4.0 6.0 "radial"  
1 3.0 0.6 20.0)
```

Related Topics

[geCreateTaper](#)

[geCreateTrl](#)

[Microwave Commands](#)

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

<i>d_cellview</i>	Database ID of the cellview containing the transmission line taper.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair. For more information, see Database Access Function Argument Types .
<i>l_points</i>	Points used to create the transmission line taper. The point list must contain two points.
<i>f_width1</i>	Specifies the width of the first end of the taper. Valid Values: any number greater than zero in user units
<i>f_width2</i>	Specifies the width of the second end of the taper. Valid Values: any number greater than zero in user units
<i>t_taperStyle</i>	Specifies the type of taper to be created. Default Value: <code>linear</code> . Valid Values: <code>linear</code> and <code>exponential</code> .
<i>x_nPtsToTaper</i>	The number of segments to use in approximating the shape of the taper. Valid Value: any integer greater than 1

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Value Returned

<i>d_taperId</i>	The database ID of the dbLine that represents the tapered transmission line if the transmission line taper is created.
<i>nil</i>	The transmission line taper is not created.

Examples

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 that represents the tapered transmission line.

```
geCreateTaper( cell2 "metal1R" list(0:0 12:12) 4 6 linear 20)
```

Related Topics

[geCreateBend](#)

[geCreateTrl](#)

[Microwave Commands](#)

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

<i>d_cellview</i>	Database ID of the cellview in which the transmission line is created.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair. For more information, see Database Access Function Argument Types .
<i>l_points</i>	List of points used to create the transmission line.
<i>f_width</i>	Width of the transmission line. Valid Values: any number greater than zero in user units
<i>t_bendStyle</i>	Bend style for the transmission line. Default Value: bend Valid Values: bend, chamfer, and radial.
<i>f_bendFac</i>	The maximum width allowed in a standard bend before the bend is automatically chamfered. (Refer to the <i>Virtuoso Layout Suite L User Guide</i> for further details.) Default Value: 1 Valid Value: any number

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

<i>f_radFac</i>	The ratio of the centerline bend width to the transmission line width for a radial bend. Valid Value: any number
<i>f_chamFac</i>	The fraction of the corner that is removed in a chamfer bend. Valid Value: a number between 0 and 1
<i>x_nPtsToPI</i>	The number of segments in a 180-degree radial bend. Default Value: 20 Valid Value: any integer greater than 1

Value Returned

<i>d_trlId</i>	The database ID for the dbLine that represents the transmission line if the transmission line is created.
<i>nil</i>	The transmission line is not created.

Examples

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.

```
geCreateTrl( cell2 "metal1R" list(0:0 10:0 10:10) 4.0 "radial" 1.0 3.0 0.6 20)
```

Related Topics

[geCreateBend](#)

[geCreateTaper](#)

[Microwave Commands](#)

Mouse Functions

You can use mouse functions to change the mode of the point selection in the current window. These functions are set up to be used as mouse bindkeys.

Related Topics

[mouseAddPt](#)

[mouseAddSelectPt](#)

[mouseFinishPt](#)

[mousePopUp](#)

[mouseSingleSelectPt](#)

[mouseSubSelectPt](#)

[mouseUndoPt](#)

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

Related Topics

[mouseAddSelectPt](#)

[mouseFinishPt](#)

[mousePopUp](#)

[mouseSingleSelectPt](#)

[mouseSubSelectPt](#)

[mouseUndoPt](#)

mouseAddSelectPt

`mouseAddSelectPt ( )`

Description

Performs an add mode point selection at the point you click in the current window.

Arguments

None

Value Returned

None

Related Topics

[mouseAddPt](#)

[mouseFinishPt](#)

[mousePopUp](#)

[mouseSingleSelectPt](#)

[mouseSubSelectPt](#)

[mouseUndoPt](#)

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

Related Topics

[mouseAddPt](#)

[mouseAddSelectPt](#)

[mousePopUp](#)

[mouseSingleSelectPt](#)

[mouseSubSelectPt](#)

[mouseUndoPt](#)

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

Related Topics

[hiOpenWindow](#)

[hiCreateWindow](#)

[mouseAddPt](#)

[mouseAddSelectPt](#)

[mouseFinishPt](#)

[mouseSingleSelectPt](#)

[mouseSubSelectPt](#)

[mouseUndoPt](#)

mouseSingleSelectPt

`mouseSingleSelectPt ( )`

Description

Performs a single mode point selection at the point you click in the current window.

Arguments

None

Value Returned

None

Related Topics

[mouseAddPt](#)

[mouseAddSelectPt](#)

[mouseFinishPt](#)

[mousePopUp](#)

[mouseSubSelectPt](#)

[mouseUndoPt](#)

mouseSubSelectPt

`mouseSubSelectPt ( )`

Description

Performs a sub-mode point selection at the point you click in the specified window.

Arguments

None

Value Returned

None

Related Topics

[mouseAddPt](#)

[mouseAddSelectPt](#)

[mouseFinishPt](#)

[mousePopUp](#)

[mouseSingleSelectPt](#)

[mouseUndoPt](#)

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

Related Topics

[mouseAddPt](#)

[mouseAddSelectPt](#)

[mouseFinishPt](#)

[mousePopUp](#)

[mouseSingleSelectPt](#)

[mouseSubSelectPt](#)

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Virtuoso Studio Design Environment SKILL Reference

Graphics Editor Functions

Database Access Functions

This topic provides a list of Cadence® SKILL functions used to create, modify, and delete Cadence® database objects.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- Data Stored as Objects
 - ❑ Object Identifier (dbObject)
 - ❑ Object Types and Classes
 - ❑ Relationships Among Objects
 - ❑ Database Access Function Argument Types
- Inherited Connections Functions
- Undo and Redo Functions
- Fluid Shape Functions
- Pcell Abutment Support Functions
- Object Creation and Retrieval Functions
- Shape Conversion Functions
- Mosaic Creation and Retrieval Functions
- Property Creation and Modification Functions
- Group Creation and Modification Functions
- Shape Transformation Functions
- Bounding Box Functions
- Property Bag Functions

Virtuoso Studio Design Environment SKILL Reference

Database Access 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](#)
- [Trim Functions](#)
- [Patch Functions](#)
- [Marker Functions](#)
- [Query Functions](#)
- [Row 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](#)
- [Remaster SKILL Functions](#)

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.

Design Framework II supports two database types, CDB and OpenAccess.

Related Topics

[dbGetDatabaseType](#)

[Object Identifier \(DBObject\)](#)

[Object Types and Classes](#)

[Relationships Among Objects](#)

[Database Access Function Argument Types](#)

[dbIsId](#)

[dbValidP](#)

[dbobjectp](#)

[dbDeleteObject](#)

dbGetDatabaseType

```
dbGetDatabaseType (  
    )  
=> CDBA / OpenAccess
```

Description

Returns the database type supported by the current executable. The function returns `CDBA` when called from a regular CDB executable, and `OpenAccess` when called from an `OpenAccess` executable.

Arguments

None

Value Returned

<code>CDBA</code>	When called from a regular CDB executable.
<code>OpenAccess</code>	When called from <code>OpenAccess</code> .

Related Topics

[Data Stored as Objects](#)

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.

Related Topics

[Data Stored as Objects](#)

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.

Related Topics

[Object Identifier \(dbObject\)](#)

[Data Stored as Objects](#)

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.

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 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.

Related Topics

Attribute Retrieval and Modification Functions

Data Stored as Objects

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.

Related Topics

Data Stored as Objects

Database 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.

Argument Type	Description	Examples
<i>l_bBox</i>	A list of two <i>l_point</i> 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.	<code>list(list(0 0) list(10 100))</code> <code>list(0:0 10:100)</code>
<i>l_bBox</i>	A list of two <i>l_point</i> 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.	<code>list(list(0 0) list(10 100))</code> <code>list(0:0 10:100)</code>

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Argument Type	Description	Examples
<i>l_group</i>	A list of strings describing group type. The recognized strings and their definitions are: ■ <code>collection</code>: The group can contain a dbObject more than once. ■ <code>set</code>: The group can contain a dbObject only once. ■ <code>ordered</code>: 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. ■ <code>unordered</code>: The order of the group members is insignificant. ■ <code>nonUniqueName</code>: Many groups within the same cellview have the same name. ■ <code>uniqueName</code>: Each group within the same cellview has a unique name. ■ <code>deleteNever</code>: Do not delete the group automatically. ■ <code>deleteFirst</code>: Automatically delete the group when the first group member is deleted. <code>deleteLast</code>: Automatically delete the group when the last group member is deleted.	<code>list("set" "ordered")</code>
<i>l_LP</i>	A list containing the layer and purpose.	<code>list(tx_layer tx_purpose)</code>

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Argument Type	Description	Examples
<i>l_point</i>	A list of two coordinates describing an xy location. Each coordinate is defined as an integer. Virtuoso supports a maximum of 4000 points or vertexes.	<pre>list(0 100) 0:100 list(list(0 0) list(10 10) list(100 100)) list(0:0 10:10 100:100)</pre>
<i>;-string</i>	A list of strings. There is no restriction on the string values.	<pre>list("dog" "cat" "horse" "")</pre>
<i>l_transform</i>	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 <i>l_point</i>. ■ orient: The orientation of the object described as a <i>t_orient</i>. ■ 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.	<pre>list(10.0:2.0 "R90" 2.0) list(list(10.0 2.0) "R90")</pre>
<i>tx_layer</i>	A string or integer describing a layer.	<pre>"hilite" "metal" 254</pre>
<i>txl_layerPurpose</i>	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 <i>drawing</i> . 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 <i>drawing</i> .	<pre>list(68 "drawing")</pre>
<i>t_direction</i>	A string describing a terminal's direction. The recognized strings are <i>input</i> , <i>inputOutput</i> , <i>jumper</i> , <i>output</i> , <i>switch</i> , <i>unused</i> , <i>powerInput</i> , <i>powerOutput</i> , <i>bidir</i> , and <i>tristate</i> .	<pre>"input" "inputOutput"</pre>

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Argument Type	Description	Examples
<i>t_font</i>	A string describing a label's font. The recognized strings are <i>euroStyle</i> (or <i>Europe</i>), <i>gothic</i> , <i>math</i> , <i>roman</i> , <i>script</i> , <i>stick</i> , <i>swedish</i> , <i>fixed</i> , and <i>raster</i>	"gothic" "math"
<i>t_just</i>	A string describing label's justification. The recognized strings are <i>lowerLeft</i> , <i>lowerCenter</i> , <i>lowerRight</i> , <i>centerLeft</i> , <i>centerCenter</i> , <i>centerRight</i> , <i>upperLeft</i> , <i>upperCenter</i> , <i>upperRight</i> , <i>leftTop</i> , <i>leftCenter</i> , <i>leftBottom</i> , <i>centerTop</i> , <i>centerBottom</i> , <i>rightTop</i> , <i>rightCenter</i> , and <i>rightBottom</i> .	"centerCenter" "upperCenter"
<i>t_orient</i>	A string describing an object's orientation. The recognized strings are <i>0</i> , <i>90</i> , <i>180</i> , <i>270</i> , <i>sideways</i> , <i>sideways&90</i> , <i>upsideDown</i> , <i>sideways&270</i> , <i>R0</i> , <i>R90</i> , <i>R180</i> , <i>R270</i> , <i>MY</i> , <i>MYR90</i> , <i>MX</i> , and <i>MXR90</i> .	"R90" "upsideDown"
<i>t_pathStyle</i>	A string describing a path's style. The recognized strings are <i>extendExtend</i> , <i>roundRound</i> , <i>truncateExtend</i> , <i>varExtendExtend</i> , <i>squareFlush</i> , <i>squareOffset</i> , and <i>octagonEnded</i> .	"truncateExtend" "squareFlush"
<i>t_prop</i>	A string describing a property's type. The recognized strings are: <i>boolean</i> , <i>int</i> , <i>fileName</i> , <i>float</i> , <i>hierProp</i> , <i>ILExpr</i> , <i>ILList</i> , <i>NLPExpr</i> , <i>string</i> , <i>time</i> , <i>fixnum</i> , <i>integer</i> , <i>flonum</i> , <i>Boolean</i> , <i>ilExpr</i> , <i>nlpExpr</i> , <i>property</i> , and <i>ilList</i> .	" <i>ILExpr</i> " "boolean"
<i>tx_purpose</i>	A string or number describing a purpose.	"drawing" 252

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

dbIsId

```
dbIsId(  
    d_id  
)  
=> t / nil
```

Description

Specifies whether the argument is a valid database object ID.

Arguments

<i>d_id</i>	Database ID of an object.

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.

Examples

```
dbIsId( instID )
```

Related Topics

[dbValidP](#)

[Data Stored as Objects](#)

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

<i>d_id</i>	Database ID of an object.

Value Returned

<i>t</i>	The argument is a valid database object ID.
<i>nil</i>	The argument is not a valid database object ID.

Related Topics

[dbIsId](#)

[Data Stored as Objects](#)

dbobjectp

```
dbobjectp(  
    d_id  
)  
=> t / nil
```

Description

Checks whether argument is a database object.

Arguments

<i>d_id</i>	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 <i>w_windowId</i> to the object.

Value Returned

t	The argument is a valid database object
nil	The argument is not a valid database object.

Examples

```
dbobjectp( netId ) => t
```

Related Topics

[Data Stored as 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

<i>d_object</i>	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

<i>t</i>	The object is deleted.
<i>nil</i>	The object is not deleted.

Examples

```
dbDeleteObject(net)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Data Stored as Objects

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 signals and inherited terminals.

Related Topics

[dbCreateSigNetExpr](#)

[dbCreateTermNetExpr](#)

[dbCVHasInheritedSig](#)

[dbCVHasInheritedSig](#)

[dbCVHasNetSetProp](#)

[dbDeleteSigNetExpr](#)

[dbDeleteTermNetExpr](#)

[dbFix](#)

[dbGetCellViewInheritedSig](#)

[dbGetCellViewInheritedSig](#)

[dbGetSigNetExpr](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

dbGetTermNetExpr

dbIsValidSigNetExpr

dbIsValidTermNetExpr

dbIsSigInherited

dbIsTermInherited

dbParseSigNetExpr

dbParseTermNetExpr

dbReplaceSigNetExpr

dbReplaceTermNetExpr

Net and Terminal Name Syntax

Instance Name Syntax

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

<i>d_sigId</i>	Specifies a signal that has the inherited net expression attached to it.
<i>t_netExpression</i>	Specifies the inherited net expression that gets attached to the signal.

Value Returned

<i>t</i>	The inherited net expression is successfully attached to the signal.
<i>nil</i>	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.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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

<i>d_termId</i>	Specifies a terminal that has the inherited net expression attached to it.
<i>t_netExpression</i>	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.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbCVHasInheritedSig

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

Description

Checks if there is any inherited signal in the given cellview.

Argument

<i>d_cellViewId</i>	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.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbCVHasInheritedTerm

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

Description

Checks whether there is any inherited terminal in the given cellview.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

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.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbCVHasNetSetProp

```
dbCVHasNetSetProp(  
    d_cellviewId  
)  
=> t / nil
```

Description

Determines if the netSet property is set on a cellview.

Arguments

<code>d_cellviewId</code>	Database ID of the source cellview.

Value Returned

<code>t</code>	The property is set on the cellview.
<code>nil</code>	The property is not set.

Examples

```
cv = dbOpenCellViewByType( "netSetLib" "test1" "schematic" "schematic" "w")  
dbCVHasNetSetProp(cv) ==> t
```

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbDeleteSigNetExpr

```
dbDeleteSigNetExpr(  
    d_sigId  
)  
=> t / nil
```

Description

Given a signal ID in *d_sigId*, this function deletes any existing inherited net expression associated with this signal and removes it from the set of inherited signals.

Arguments

<i>d_sigId</i>	Database ID of a signal that has the inherited net expression attached to it.

Values Return

<i>t</i>	The inherited net expression exists on the object and is successfully deleted.
<i>nil</i>	No inherited net expression belongs to the object, or if the given signal cannot be found.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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

<i>d_termId</i>	Specifies a terminal that has the inherited net expression attached to it.

Value Returned

<i>t</i>	The inherited net expression exists on the object and is successfully deleted.
<i>nil</i>	No inherited net expression belongs to the object, or if the given terminal cannot be found.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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

<i>d_cellViewId</i>	Specifies the cellview ID.
<i>g_reBind</i>	Specifies whether rebinding is needed.

Value Returned

<i>t</i>	The data corruption was fixed.
<i>nil</i>	The data corruption was not fixed.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbGetCellViewInheritedSig

```
dbGetCellViewInheritedSig(  
    d_cellViewId  
)  
=> l_sigId / nil
```

Description

Retrieves from the given cellview all signals to which inherited net expressions are attached.

Arguments

<i>d_cellViewId</i>	Specifies the cellview ID.

Value Returned

<i>l_sigId</i>	A list of signals that have inherited expressions.
<i>nil</i>	There are no inherited signals in the given cellview.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbGetCellViewInheritedTerm

```
dbGetCellViewInheritedTerm(  
    d_cellViewId  
)  
=> l_termId / nil
```

Description

Retrieves from the given cellview all terminals to which inherited net expressions are attached.

Arguments

<i>d_cellViewId</i>	Specifies the cellview ID.

Value Returned

<i>l_termId</i>	A SKILL list of terminals that have inherited expressions.
<i>nil</i>	There are no inherited terminals in the given cellview.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbGetSigNetExpr

```
dbGetSigNetExpr(  
    d_sigId  
)  
=> t_netExpression / nil
```

Description

Retrieves any inherited net expression found attached to the signal.

Arguments

<i>d_sigId</i>	Database ID of a signal.

Value Returned

<i>t_netExpression</i>	The inherited net expression.
<i>nil</i>	No inherited net expression is found, or the given signal cannot be found.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbGetTermNetExpr

```
dbGetTermNetExpr(  
    d_termId  
)  
=> t_netExpression / nil
```

Description

Retrieves any inherited net expression found attached to the terminal.

Arguments

<i>d_termId</i>	A terminal.

Value Returned

<i>t_netExpression</i>	The inherited net expression.
<i>nil</i>	No inherited net expression is found, or the given terminal cannot be found.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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.

Examples

```
[@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.
```

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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.

Examples

```
[@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.
```

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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

<i>d_sigId</i>	Database ID of a signal.

Value Returned

<i>t</i>	An inherited net expression is found to be attached to the given signal.
<i>nil</i>	There is no inherited net expression attached to the signal, or the given signal cannot be found.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

dbIsTermInherited

```
dbIsTermInherited(  
    d_termId  
)  
=> t / nil
```

Description

Checks whether there is an inherited net expression attached to the given terminal.

Arguments

<i>d_termId</i>	Database ID of 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.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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

<i>t_netExpression</i>	The inherited net expression that gets attached to a signal.

Value Returned

<i>t_searchScope</i>	Search scope.
<i>t_propName</i>	The property name to be used in searching.
<i>t_formatString</i>	The format string.
<i>t_defaultNetName</i>	The default net name.
<i>nil</i>	The given inherited net expression is of illegal syntax.

Examples

Given the net expression found in the *schematic* view of the cell *inv*, [*@vdd: %: vdd!*], this function returns

```
list("@ "vdd" "%" "vdd!")
```

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Instance Name Syntax

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

<i>t_netExpression</i>	The inherited net expression that gets attached to a terminal.

Value Returned

<i>t_searchScope</i>	Search scope.
<i>t_propName</i>	Property name to be used in searching.
<i>t_formatString</i>	Format string.
<i>t_defaultNetName</i>	Default net name.
<i>nil</i>	The given inherited net expression is of illegal syntax.

Examples

```
list("@ "vdd" "%" "vdd!")
```

Given the net expression found in the schematic view of the cell *inv*, [*@vdd:%:vdd!*], this function returns

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Instance Name Syntax

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

<i>d_sigId</i>	Database ID of a signal.
<i>t_netExpression</i>	The inherited net expression that gets attached to a signal.

Value Returned

<i>t</i>	The inherited net expression exists on the object and is successfully replaced or created.
<i>nil</i>	The given signal cannot be found, or the given inherited net expression is of illegal syntax.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

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

<i>d_termId</i>	Database ID of a terminal.
<i>t_netExpression</i>	The inherited net expression that gets attached to a terminal.

Value Returned

<i>t</i>	The inherited net expressions exists on the object and is successfully replaced or created.
<i>nil</i>	The given terminal cannot be found, or if the given inherited net expression is of illegal syntax.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

[Instance Name Syntax](#)

Net and Terminal Name Syntax

This section describes the name syntax for net and terminal names. The name syntax is in Backus Nour Format (BNF).

[]	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 <i>or</i> .
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 <i>prefix repeat operator</i> . 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 <i>suffix repeat operator</i> . 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 <i>the entire expanded term</i> enclosed in parentheses.

Examples:

```
nameExpr ::= [ < *N > ] nameTerm { , nameExpr }
nameTerm ::= baseName [ < vectExpr > ] | < nameExpr >
vectExpr ::= vectTerm [ *N ] { , vectExpr }
vectTerm ::= N [ : N [ : N ] ] | < vectExpr >
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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>*.

Related Topics

Inherited Connections Functions

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Instance Name Syntax

Instance Name Syntax

Syntax of an instance name is in Backus-Naur Format (BNF) notation.

```
instName ::= baseName [ < number [ : number ] > ]
```

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.

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 parenthetical 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.

Related Topics

[Inherited Connections Functions](#)

[Net and Terminal Name Syntax](#)

Undo and Redo Functions

This topic provides a list of Cadence® SKILL functions associated with undo and redo handle functions.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at

Related Topics

[dbUndoToHandle](#)

[dbGetUndoHandles](#)

[dbGetRedoHandles](#)

[dbEnableUndo](#)

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

<i>t_handleName</i>	Name of the handle name that needs to be set as the undo handle.
<i>g_moveHandle</i>	Controls what happens if there is an existing undo checkpoint with the same name. ■ When set to <code>t</code>, 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 <code>nil</code>, if there is an existing undo checkpoint with the same name, an error is returned because the handle names must be unique.

Value Returned

<code>t</code>	The handle was set.
<code>nil</code>	The handle was not set.

Examples

Sets the undo handle, `checkpoint5` on the current undo checkpoint.

```
dbSetUndoHandle("checkpoint5")
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

```
dbSetUndoHandle("checkpoint5" t)
```

Related Topics

[Undo and Redo Functions](#)

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.

`dbUndoToHandle` depends on the HI event loop just like [hiUndo](#), and has the same limitations as `hiUndo`. For example, `dbUndoToHandle` cannot be called from within a SKILL procedure.

Arguments

<code>t_handleName</code>	Handle name that was set using <code>dbSetUndoHandle</code> .

Value Returned

<code>t</code>	The undo was invoked.
<code>nil</code>	The undo was not invoked.

Examples

Rolls back to the undo handle `checkpoint5`.

```
dbUndoToHandle ("checkpoint5")
dbUndoToHandle ()
```

Related Topics

[hiUndo](#)

[dbSetUndoHandle](#)

[Undo and Redo Functions](#)

dbGetUndoHandles

```
dbGetUndoHandles (  
    )  
=> l_handlelist
```

Description

Returns a list of handles in the undo stack.

Arguments

None

Value Returned

<i>l_handlelist</i>	The list of handles in the undo stack.

Examples

```
dbGetUndoHandles ()
```

Related Topics

[Undo and Redo Functions](#)

dbGetRedoHandles

```
dbGetRedoHandles (
    )
    => l_handlelist
```

Description

Returns a list of handles in the redo stack.

Arguments

None

Value Returned

<i>l_handlelist</i>	The list of handles in the redo stack.

Examples

```
dbGetRedoHandles () +
```

Related Topics

[Undo and Redo Functions](#)

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

<i>t_handleName</i>	The handle name.

Value Returned

t	The redo was invoked.
nil	The redo was not invoked.

Examples

```
dbRedoToHandle("my_checkpoint")
```

Related Topics

Undo and Redo Functions

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.

Examples

```
dbEnableUndo ()
```

Related Topics

[Undo and Redo Functions](#)

Fluid Shape Functions

This topic provides a list of Cadence® SKILL functions associated with fluid shapes.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbGetFluidShapeByName](#)

[dbGetFluidShapeName](#)

[dbGetFluidShapes](#)

[dbIsFluidShape](#)

[dbSetFluidShape](#)

dbGetFluidShapeByName

```
dbGetFluidShapeByName (
    d_cellViewId
    t_shapeName
)
=> d_fluidShapeId
```

Description

Returns the database ID for a specified fluid shape name in a given cellview.

Arguments

<i>d_cellViewId</i>	Database ID of the cellview.
<i>t_shapeName</i>	Name of the fluid shape.

Value Returned

<i>d_fluidShapeId</i>	Database ID of the fluid shape.

Examples

```
rect1 = dbGetFluidShapeByName(inst~>master "fluidShape2")
```

Related Topics

[Fluid Shape Functions](#)

dbGetFluidShapeName

```
dbGetFluidShapeName (
    d_shapeId
)
=> t_FluidName / nil
```

Description

Returns the fluid name of the specified shape if it is a fluid shape.

Arguments

<i>d_shapeId</i>	Database ID of a shape.

Return Value

<i>t_FluidName</i>	Fluid name of the specified fluid shape.
<i>nil</i>	The given shape is not a fluid shape.

Examples

```
name = dbGetFluidShapeName(rect1)
```

Returns the fluid name of the shape `rect1`.

Related Topics

[Fluid Shape Functions](#)

dbGetFluidShapes

```
dbGetFluidShapes (
    d_cvId
)
=> l_fluidShapes
```

Description

Returns all fluid shapes on the specified cellview `dbGetFluidShapes`.

Arguments

<i>d_cvId</i>	Database ID of a cellview.

Return Value

<i>l_fluidShapes</i>	List of all fluid shapes on the specified cellview.

Examples

Returns list of fluid shapes on master cellview.

```
fluidShapes = dbGetFluidShapes(inst~>master)
```

Related Topics

[Fluid Shape Functions](#)

dbIsFluidPcell

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

Description

Checks if submaster belongs to a Fluid Pcell supermaster.

Arguments

<i>d_cellViewId</i>	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.

Examples

Cellview *cvId* belongs to a Fluid Pcell supermaster.

```
dbIsFluidPcell(cvId) => t
```

Related Topics

[Fluid Shape Functions](#)

dbIsFluidShape

```
dbIsFluidShape(  
    d_shapeId  
)  
=> t / nil
```

Description

Checks if the specified shape is marked as a Fluid shape.

Arguments

<i>d_shapeId</i>	Database ID of a shape.

Return Value

t	Specified shape is a fluid shape.
nil	Specified shape is not a fluid shape.

Examples

```
when(dbIsFluidShape(rect2) printf("Shape: %L is a fluid shape.\n" rect2~>??)
```

If the shape `rect2` is a fluid shape, prints the following:

```
Shape: rect2 is a fluid shape.
```

Related Topics

[Fluid Shape Functions](#)

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

<i>d_shapeId</i>	Database ID of a shape to be marked as a fluid shape
<i>t_fluidName</i>	The name to mark the specified shape as a fluid shape.

Return Value

<i>t</i>	Specified name is set for the specified shape to mark it as a fluid shape.
<i>nil</i>	The operation failed.

Examples

Sets the name `fluidShape2` on the shape `rect2` to mark it as a fluid shape.

```
dbSetFluidShape(rect2 "fluidShape2") cv
```

Related Topics

[Fluid Shape Functions](#)

Pcell Abutment Support Functions

This topic provides a list of Cadence® SKILL functions that 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.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[dbAbutIsInteractive](#)

[dbAbutIsReAbutGroup](#)

[dbSetFigAbutName](#)

[dbSetFigAbutRefName](#)

[dbGetFigAbutName](#)

[dbGetFigAbutRefName](#)

[dbHasFigAbutName](#)

[dbHasFigAbutRefName](#)

[dbFindAbutFig](#)

[dbUnabutGroup](#)

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

<code>t</code>	Returns <code>t</code> if the abutment being called is interactive.
<code>nil</code>	Returns <code>nil</code> if the abutment being called is not interactive.

Examples

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.

Related Topics

[Pcell Abutment Support Functions](#)

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.

Examples

```
groupId = Database Id of the group going through un-abutment (event 3)  
dbAbutIsReAbutGroup(groupId)
```

Related Topics

[Pcell Abutment Support Functions](#)

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

<i>d_figId</i>	Specifies the ID associated with the figure to which you want to assign an abutment name
<i>t_abutName</i>	Specifies the abutment name to be assigned to the figure ID

Value Returned

<i>t</i>	Returns <i>t</i> if the abutment name is successfully assigned
<i>nil</i>	Returns <i>nil</i> if the abutment name is not assigned

Examples

```
dbSetFigAbutName (fig "dpolyRight")
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Pcell Abutment Support Functions

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

<i>d_figId</i>	Specifies the ID of the figure to which you want to assign an abutment reference name
<i>t_abutRefName</i>	Specifies the abutment reference name to be assigned to the figure ID

Value Returned

<i>t</i>	Returns <i>t</i> if the abutment reference name is successfully assigned
<i>nil</i>	Returns <i>nil</i> if the abutment reference name is not assigned

Examples

```
dbSetFigAbutRefName (fig "diffpolyRight")
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Pcell Abutment Support Functions

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

<i>d_figId</i>	Specifies the figure ID for which you want to retrieve the abutment name

Value Returned

<i>t_abutName</i>	Returns the abutment name assigned to the figure if it is successfully retrieved
<i>nil</i>	Returns <i>nil</i> if an abutment name is not assigned to the given figure ID or if the above conditions are not met

Examples

```
dbGetFigAbutName (fig) => "dpolyRight"
```

Related Topics

[Pcell Abutment Support Functions](#)

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

<i>d_figId</i>	Specifies the ID of the figure for which you want to retrieve the abutment reference name

Value Returned

<i>t_abutName</i>	Returns the abutment reference name assigned to the figure if it is successfully retrieved
<i>nil</i>	Returns <i>nil</i> if an abutment reference name is not assigned to the given figure ID or if the conditions specified above are not met

Examples

```
dbGetFigAbutRefName (fig) => "diffpolyRight"
```

Related Topics

Pcell Abutment Support Functions

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

<i>d_figId</i>	Specifies the ID of the figure for which you want to confirm if an abutment name was set

Value Returned

<i>t</i>	Returns <i>t</i> if an abutment name was set for the figure with the specified ID
<i>nil</i>	Returns <i>nil</i> if no abutment name was assigned to the specified figure ID or if the conditions specified above are not met

Examples

```
dbHasFigAbutName (fig) => t
```

Related Topics

[Pcell Abutment Support Functions](#)

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

<i>d_figId</i>	Specifies the ID of the figure for which you want to confirm if an abutment reference name was set

Value Returned

<i>t</i>	Returns <i>t</i> if an abutment reference name was set for the figure with the specified ID
<i>nil</i>	Returns <i>nil</i> if no abutment reference name was assigned to the specified figure or if the figure did not exist in the Pcell

Examples

```
dbHasFigAbutRefName(fig) => t
```

Related Topics

[Pcell Abutment Support Functions](#)

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

<i>d_cvId</i>	Specifies the ID of the submaster cell in which you want to search for the figure with the specified abutment name
<i>t_abutName</i>	Specifies the abutment name of the figure that you want to search

Value Returned

<i>d_figId</i>	Returns the ID of the figure you are searching for
<i>nil</i>	Returns <i>nil</i> 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

Examples

```
dbFindAbutFig(cv "dpolyRight") => fig
```

Related Topics

[Pcell Abutment Support Functions](#)

dbUnabutGroup

```
dbUnabutGroup(  
    d_groupID  
)  
=> t / nil
```

Description

Unabuts members of an abutment group.

Arguments

<i>d_groupID</i>	Database ID of the group going through unabutment.

Value Returned

<i>t</i>	Members of the specified abutment group are unabutted.
<i>nil</i>	The operation fails.

Examples

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)  
)
```

Related Topics

[Pcell Abutment Support Functions](#)

Object Creation and Retrieval Functions

This topic provides a list of Cadence® SKILL functions associated with object creation and retrieval.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCopyShape](#)

[dbCreateLine](#)

[dbCreatePath](#)

[dbCreatePolygon](#)

[dbCreateRect](#)

[dbCreateRuler](#)

[dbCreateEllipse](#)

[dbCreateArc](#)

[dbCreateDonut](#)

[dbCreateDot](#)

[dbCreateLabel](#)

[dbCreateTextDisplay](#)

[dbHasShape](#)

[dbMoveShape](#)

[dbSetTextDisplayNameValueVisible](#)

dbCopyShape

```
dbCopyShape (
    d_shapeId
    d_cellviewId
    [ l_transform ]
)
=> d_newShapeId / nil
```

Description

Copies a shape to the specified cellview with the given transformation.

Note: By default, coloring information is not copied. To copy coloring information, set the `cdba copyMPAttributes` environment variable to `t` before using `dbCopyShape`.

Arguments

<i>d_shapeId</i>	Database ID of the shape to be copied.
<i>d_cellviewId</i>	Database ID of a cellview.
<i>l_transform</i>	Optional list value to transform the copied shape. List includes: origin offset, rotation, and magnitude. Default is ' (0:0 "R0" 1)

Value Returned

<i>d_newShapeId</i>	Returns the database ID of the new shape.
<i>nil</i>	An error occurred during execution.

Examples

```
dbCopyShape(shapeId cvId list(10:10 "R0" 1)
```

Related Topics

[Object Creation and Retrieval Functions](#)

dbCreateLine

```
dbCreateLine(  
    d_cellView  
    txl_layerPurpose  
    l_points  
)  
=> d_line / nil
```

Description

Creates a line.

Arguments

<i>d_cellView</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_points</i>	Points list, must contain at least two points.

Value Returned

<i>d_line</i>	The <i>dbObject</i> of the line.
<i>nil</i>	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))
```

Related Topics

[Object Creation and Retrieval Functions](#)

[Database Access Function Argument Types](#)

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

<i>d_cellView</i>	The cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>t_purpose</i>	Purpose, if not specified, defaults to <i>drawing</i> .
<i>l_points</i>	Point list; must contain at least two points.
<i>n_width</i>	Path width specified in user units.
<i>t_pathStyle</i>	Path style, defaults to <code>truncateExtend</code> .

Value Returned

<i>d_path</i>	The <i>dbObject</i> of the path.
<code>nil</code>	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")
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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)
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

dbCreatePolygon

```
dbCreatePolygon(  
    d_cellView  
    txl_layerPurpose  
    l_points  
)  
=> d_polygon / nil
```

Description

Creates a polygon.

Arguments

<i>d_cellView</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_points</i>	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.

Value Returned

<i>d_polygon</i>	The <i>dbObject</i> of the polygon.
<i>nil</i>	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))
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

dbCreateRect

```
dbCreateRect(  
    d_cellView  
    txl_layerPurpose  
    l_bBox  
)  
=> d_rect / nil
```

Description

Creates a rectangle.

Arguments

<i>d_cellView</i>	The cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_bBox</i>	Defines the lower left and upper right corners of the bounding box.

Value Returned

<i>d_rect</i>	The <i>dbObject</i> of the rectangle.
<i>nil</i>	The rectangle is not created.

Examples

```
dbCreateRect(cv list(87 "pin") list(0:0 6.3:8.7))
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

dbCreateRuler

```
dbCreateRuler(  
    d_cellviewId  
    l_points  
)  
=> d_rulerId / nil
```

Description

Creates a dbRuler object in the design.

Arguments

<i>d_cellviewId</i>	Cellview Id of the design.
<i>l_points</i>	List of ruler points.

Value Returned

<i>d_rulerId</i>	The Id of the dbRuler object, if successful.
<i>nil</i>	The creation of ruler object failed.

Examples

```
dbCreateRuler (cvId list (list (0 0) list (10:0))
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

dbCreateEllipse

```
dbCreateEllipse(  
    d_cellView  
    txl_layerPurpose  
    l_bBox  
)  
=> d_ellipse / nil
```

Description

Creates an ellipse.

Arguments

<i>d_cellView</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_bBox</i>	Defines the bounding box of the ellipse.

Value Returned

<i>d_ellipse</i>	The <i>dbObject</i> of the ellipse.
<i>nil</i>	The ellipse is not created.

Examples

```
dbCreateEllipse(cv 1 list(0:0 2:3))
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

dbCreateArc

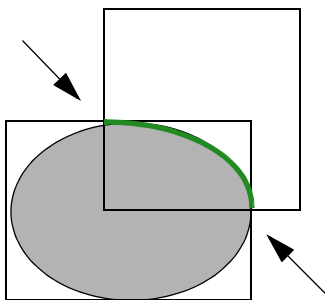
```
dbCreateArc(  
    d_cellView  
    txl_layerPurpose  
    l_ellipse_bBox  
    l_arc_bBox  
)  
=> d_arc / nil
```

Description

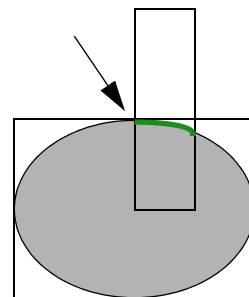
Creates an arc.

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

<i>d_cellView</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_ellipse_bBox</i>	Bounding box of the ellipse containing the arc.
<i>l_arc_bBox</i>	Bounding box of the arc.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_arc</i>	The <i>dbObject</i> of the arc.
<i>nil</i>	The arc is not created.

Examples

```
dbCreateArc(cv list("background" "drawing") list(0:0 2:2) list(1:1 2:2))
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

dbCreateDonut

```
dbCreateDonut(  
    d_cellView  
    txl_layerPurpose  
    l_point  
    x_outR  
    x_holeR  
)  
=> d_donut / nil
```

Description

Creates a donut (two concentric circles).

Arguments

<i>d_cellView</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_point</i>	Defines the center of the donut.
<i>x_outR</i>	Outer circle radius, must be larger than <i>x_holeR</i> .
<i>x_holeR</i>	Inner circle radius.

Value Returned

<i>d_donut</i>	The <i>dbObject</i> of the donut.
<i>nil</i>	The donut is not created.

Examples

```
dbCreateDonut(cv 1 0:0 100 50)C
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

dbCreateDot

```
dbCreateDot(  
    d_cellView  
    txl_layerPurpose  
    l_point  
)  
=> d_dot / nil
```

Description

Creates a dot.

Arguments

<i>d_cellView</i>	The cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_point</i>	Defines the location of the dot.

Value Returned

<i>d_dot</i>	The <i>dbObject</i> of the dot.
<i>nil</i>	The dot is not created.

Examples

```
dbCreateDot(cv "metal1" 1:1)
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

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

<i>d_cellView</i>	The cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair.
<i>l_point</i>	Defines the origin of the label.
<i>t_label</i>	Text string for the label.
<i>t_just</i>	Justification string for the label.
<i>t_orient</i>	Orientation string for the label.
<i>t_font</i>	Font style string for the label.
<i>x_height</i>	Height of the label, must be given as a positive number.

Value Returned

<i>d_label</i>	The <i>dbObject</i> of the label.
<i>nil</i>	The label is not created.

Examples

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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

```
dbCreateLabel(cv 4 1:1 "myLabel" "centerLeft" "R0" "roman" 2)
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

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.

Arguments

<i>d_associateId</i>	Associate of the text display. This can be a label in a master cellview, a property, or a <i>dbObject</i> whose attribute <i>t_attrName</i> 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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

<i>d_ownerId</i>	Owner of the text display. This can be any database object except properties or groups.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair. For more information, see Database Access Function Argument Types .
<i>l_displayFlags</i>	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, <i>t</i> must be specified here, followed by a specification for each of the display characteristics listed below.
<i>l_point</i>	Origin of the text display. This must be specified for properties and for object attribute associates. For more information, see Database Access Function Argument Types .
<i>t_just</i>	Justification string for the text display. This must be specified for properties and for object attribute associates. For more information, see Database Access Function Argument Types .

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

<i>t_orient</i>	Orientation string for the text display. This must be specified for properties and for object attribute associates. For more information, see dblsld.
<i>t_font</i>	Font style string for the text display. This must be specified for properties and for object attribute associates. For more information, see Database Access Function Argument Types.
<i>x_height</i>	Height of the text display that must be given as a positive number. This must be specified for properties and for object attribute associates.
<i>g_isDrafted</i>	Specifies whether or not drafting is set for the text display. The default is <code>nil</code>.
<i>g_isOverbar</i>	Specifies whether or not over-bar is set for the text display. The default is <code>nil</code>.
<i>g_isVisible</i>	Specifies whether or not the text display is visible. The default is <code>t</code>. When <i>g_isVisible</i> is set to <code>t</code>, g_isNameVisible and g_isValueVisible cannot be set to <code>nil</code>. For example, if a user tries to set <code>(g_isVisible g_isNameVisible g_isValueVisible)</code> to <code>(t nil nil)</code>, a warning is issued, and <i>g_isValueVisible</i> is set to <code>t</code>.
<i>g_isNameVisible</i>	Specifies whether or not the property or attribute name is visible. The default is <code>t</code>. This argument applies only to a text display with a property or attribute as its associate.
<i>g_isValueVisible</i>	Specifies whether or not the property or attribute value is visible. The default is <code>t</code>. This argument applies only to a text display with a property or attribute as its associate.
<i>t_attrOrParamName</i>	Name of the attribute or the CDF parameter of the associate object specified in <i>d_associateId</i>.
<i>g_isParamAssoc</i>	Specifies whether or not <i>t_attrOrParamName</i> is a CDF parameter name. The default is <code>nil</code>.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_textDisplay</i>	The <i>dbObject</i> of the text display.
<i>nil</i>	The text display is not created.

Examples

In this 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)
```

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

dbHasShape

```
dbHasShape (
    d_cellviewId
)
=> t / nil
```

Description

Checks whether the specified cellview contains any shapes.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

Value Returned

t	The cellview contains shapes.
nil	The cellview does not contain shapes.

Related Topics

[Object Creation and Retrieval Functions](#)

dbMoveShape

```
dbMoveShape (
    shapeId
    cellviewId
    [l_transform]
)
=> d_shapeId / nil
```

Description

Moves a shape to given cellview with given transformation.

Arguments

<i>d_shapeId</i>	Database ID of the shape to be moved.
<i>d_cellviewId</i>	Database ID of a cellview.
<i>l_transform</i>	Optional list value to transform the copied shape. List includes: origin offset, rotation and magnitude. Default is ' (0:0 "R0" 1)

Value Returned

<i>d_shapeId</i>	The database ID of the moved shape.
<i>nil</i>	An error occurred during execution.

Examples

```
dbMoveShape(shapeId cvId list(10:10 "R0" 1)
```

Related Topics

[Object Creation and Retrieval Functions](#)

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

<code>d_tdId</code>	The textDisplay ID.
<code>g_isNameVisible</code>	The <code>isNameVisible</code> flag, sets name visibility.
<code>g_isValueVisible</code>	The <code>isValueVisible</code> flag, sets value visibility.

Value Returned

<code>t</code>	The flags are set successfully.
<code>nil</code>	An error occurred during execution.

Related Topics

[Database Access Function Argument Types](#)

[Object Creation and Retrieval Functions](#)

Shape Conversion Functions

This topic provides a list of Cadence® SKILL functions that convert shapes from one form to another.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbConvertArcToLine](#)

[dbConvertDonutToPolygon](#)

[dbConvertEllipseToPolygon](#)

[dbConvertPathToPolygon](#)

[dbConvertPathSegToPolygon](#)

[dbConvertLineToPath](#)

[dbConvertRectToPolygon](#)

[dbCompressPointArray](#)

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

<i>d_arc</i>	The arc to be converted.
<i>x_nSides</i>	Number of segments to be created.

Value Returned

<i>d_line</i>	The <i>dbObject</i> of the line if the conversion is successful.
<i>nil</i>	The arc is not converted.

Examples

```
dbConvertArcToLine(arcId 5)
```

Related Topics

[Shape Conversion Functions](#)

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

<i>d_donut</i>	The donut to be converted.
<i>x_nSides</i>	Number of sides created for each circle.

Value Returned

<i>d_polygon</i>	The <i>dbObject</i> of the polygon if the conversion is successful.
<i>nil</i>	The donut is not converted.

Examples

```
dbConvertDonutToPolygon(donutId 7)
```

Related Topics

[Shape Conversion Functions](#)

dbConvertEllipseToPolygon

```
dbConvertEllipseToPolygon(  
    d_ellipse  
    x_nSides  
)  
=> d_polygon / nil
```

Description

Converts an ellipse to a polygon.

Arguments

<i>d_ellipse</i>	The ellipse to be converted.
<i>x_nSides</i>	Number of sides defining the polygon.

Value Returned

<i>d_polygon</i>	The <i>dbObject</i> of the polygon if the conversion is successful.
<i>nil</i>	The ellipse is not converted.

Examples

```
dbConvertEllipseToPolygon(ellipseId 7)
```

Related Topics

[Shape Conversion Functions](#)

dbConvertPathToPolygon

```
dbConvertPathToPolygon(  
    d_path  
)  
=> d_polygon / nil
```

Description

Converts a path to a polygon with the same number of points as the path.

Arguments

<i>d_path</i>	The path to be converted.

Value Returned

<i>d_polygon</i>	The <i>dbObject</i> of the polygon if the conversion is successful.
<i>nil</i>	The path is not converted.

Examples

```
dbConvertPathToPolygon(pathId)
```

Related Topics

[Shape Conversion Functions](#)

dbConvertPathSegToPolygon

```
dbConvertPathSegToPolygon(  
    d_pathSeg  
)  
=> d_polygon / nil
```

Description

Converts a pathseg to a polygon based on the boundary list.

Arguments

<i>d_pathSeg</i>	The pathseg to be converted.

Value Returned

<i>d_polygon</i>	The <i>dbObject</i> of the polygon if the conversion is successful.
<i>nil</i>	The pathseg is not converted.

Examples

```
dbConvertPathSegToPolygon(pathSegId)
```

Related Topics

[Shape Conversion Functions](#)

dbConvertLineToPath

```
dbConvertLineToPath(  
    d_line  
    x_width  
    [ t_pathStyle ]  
)  
=> d_path / nil
```

Description

Converts a line to a path.

Arguments

<i>d_line</i>	The line to be converted.
<i>x_width</i>	Path width of the path specified in user units.
<i>t_pathStyle</i>	Path style of the path, defaults to "truncateExtend".

Value Returned

<i>d_path</i>	The <i>dbObject</i> of the path if the conversion is successful.
<i>nil</i>	The line is not converted.

Examples

```
dbConvertLineToPath(lineId 4)
```

Related Topics

[Shape Conversion Functions](#)

dbConvertRectToPolygon

```
dbConvertRectToPolygon(  
    d_rect  
)  
=> d_polygon / nil
```

Description

Converts a rectangle into a polygon.

Arguments

<i>d_rect</i>	The rectangle object that needs to be converted to a polygon object.

Value Returned

<i>d_polygon</i>	The converted polygon object.
<i>nil</i>	The rectangle to polygon object conversion failed.

Examples

```
dbConvertRectToPolygon(d_rect)
```

Related Topics

[Shape Conversion Functions](#)

dbCompressPointArray

```
dbCompressPointArray(  
    l_pointArray  
    x_isClosed  
    [ f_DBUPerUU ]  
)  
=> l_newPointArray
```

Description

Compresses collinear points in a list of points.

Arguments

<i>l_pointArray</i>	Point array to be compressed.
<i>x_isClosed</i>	Either <code>t</code> or <code>nil</code> specifying that the figure is closed or not.
<i>f_DBUPerUU</i>	Number of database units in one user unit. The default is <code>1.0</code> .

Value Returned

<i>l_newPointArray</i>	Compressed point array.

Examples

```
l1 = list((0:0) (0:1) (0:2) (0:3))  
l2 = dbCompressPointArray(l1 nil)  
=> ((0:0) (0:3))
```

Related Topics

[Shape Conversion Functions](#)

Mosaic Creation and Retrieval Functions

This topic provides a list of Cadence® SKILL functions that create a simple mosaic in the database

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateParamSimpleMosaic](#)

[dbCreateParamSimpleMosaicByMasterName](#)

[dbCreateSimpleMosaic](#)

[dbFindMosaicByName](#)

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

<i>d_cellView</i>	The <i>dbObject</i> ID of the cellview.
<i>d_masterCellView</i>	The <i>dbObject</i> ID of the master cellview.
<i>t_name</i>	If specified, the mosaic is named <i>t_name</i> ; if <i>nil</i> , the program assigns a unique name to the mosaic.
<i>l_origin</i>	Origin of the lower left instance in the array.
<i>l_orient</i>	Orientation of the instance.
<i>x_rows</i>	Number of rows defining the size of the array.
<i>x_cols</i>	Number of columns defining the size of the array.
<i>n_rowSpacing</i>	Spacing between the origins of the mosaic elements in y-direction.
<i>n_colSpacing</i>	Spacing between the origins of the mosaic elements in x-direction.
<i>l_params</i>	List of <i>l_param</i> , the names, types, and values of parameters of the mosaic.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_mosaic</i>	The <i>dbObject</i> of the mosaic.
<i>nil</i>	The mosaic is not created.

Examples

```
dbCreateParamSimpleMosaic(cv masterTile "myMos" 0:0 "R0" 5 7 3 2 list(list("l"  
"float" 5.0)))
```

Related Topics

[Mosaic Creation and Retrieval Functions](#)

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

<i>d_cellView</i>	The <i>dbObject</i> ID of the cellview.
<i>t_libName</i>	Library name of the master cellview.
<i>t_cellName</i>	Cell name of the master cellview.
<i>t_viewName</i>	View name of the master cellview.
<i>t_Name</i>	Name of the mosaic. If <i>t_Name</i> is <i>nil</i> , the program assigns a unique name to the mosaic.
<i>l_origin</i>	Origin of the lower left instance in the array.
<i>l_orient</i>	Orientation of the instance.
<i>x_rows</i>	Number of rows defining the size of the array.
<i>x_cols</i>	Number of columns defining the size of the array.
<i>n_rowSpacing</i>	Spacing between the origins of the mosaic elements in y-direction.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

<i>n_colSpacing</i>	Spacing between the origins of the mosaic elements in x-direction.
<i>l_params</i>	List of <i>l_param</i> , the names, types, and values of parameters of the mosaic.

Value Returned

<i>d_mosaic</i>	The <i>dbObject</i> of the mosaic.
<i>nil</i>	The mosaic is not created.

Examples

```
dbCreateParamSimpleMosaicByMasterName(cv "masterLib" "masterCell" "masterView"  
"myMos" 0:0 "R0" 5 7 3 2 list(list("l" "float" 5.0)))
```

Related Topics

Mosaic Creation and Retrieval Functions

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

<i>d_cellView</i>	Database ID of a cellview.
<i>d_masterCellView</i>	Database ID of a master cellview.
<i>t_name</i>	If specified, the instance is named <i>t_name</i> ; if <i>nil</i> , the program assigns a unique name to the instance.
<i>l_point</i>	Origin of the lower left instance in the array. For more information, see Database Access Function Argument Types .
<i>t_orient</i>	Orientation of the instance.
<i>x_rows</i>	Number of rows defining the size of the array.
<i>x_cols</i>	Number of columns defining the size of the array.
<i>n_rowSpacing</i>	Spacing between the origins of the mosaic elements in y-direction.
<i>n_colSpacing</i>	Spacing between the origins of the mosaic elements in x-direction.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_mosaic</i>	The <i>dbObject</i> of the mosaic.
<i>nil</i>	The mosaic is not created.

Examples

```
dbCreateSimpleMosaic(cv masterTile "myMos" 0:0 "R0" 5 7 3 2)
```

Related Topics

Mosaic Creation and Retrieval Functions

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

<i>d_cellViewId</i>	Database ID of a cellview..
<i>t_name</i>	Name of a mosaic.

Value Returned

<i>d_mosaicId</i>	A mosaic ID if a mosaic with the given name is found in the cellview specified.
<i>nil</i>	Otherwise, <i>nil</i> is returned.

Related Topics

Mosaic Creation and Retrieval Functions

Property Creation and Modification Functions

This topic provides a list of Cadence® SKILL functions that create a simple mosaic in the database

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateProp](#)

[dbCreateRangeProp](#)

[dbCreateEnumProp](#)

[dbCreateHierProp](#)

[dbFindProp](#)

[dbGetPropByName](#)

[dbReplaceProp](#)

[dbReplaceRangeProp](#)

[dbReplaceEnumProp](#)

[dbReplaceHierProp](#)

[dbCopyProp](#)

[dbCopyPropList](#)

[dbResetPropList](#)

[dbDeletePropByName](#)

[dbCopySingleProp](#)

[dbMergeSingleProp](#)

[dbReplacePropList](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

dbSearchPropByName

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

<i>g_object</i>	Specifies an object the property is to be associated with. The object can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.
<i>t_propType</i>	Type of the property.
<i>g_value</i>	Value of the property (must be consistent with <i>t_propType</i>).

Value Returned

<i>d_prop</i>	A scalar property belonging to the database object;.
<i>nil</i>	A property named <i>t_name</i> already exists on the specified object or an error occurs.

Examples

```
dbCreateProp(cv "prop1" "boolean" t)  
dbCreateProp(cv "DBUpperUU" "float" 1.0)
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>g_object</i>	The database object which can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.
<i>t_propType</i>	Type of the property.
<i>g_value</i>	Value of the property.
<i>g_lBound</i>	Lower bound of the range of values.
<i>g_uBound</i>	Upper bound of the range of values.

Value Returned

<i>d_prop</i>	A range property belonging to the database object.
<i>nil</i>	A property named <i>t_name</i> already exists on the specified object, or if the value passed in is not within the specified range. Also returns <i>nil</i> if an error occurs, such as an invalid argument, or if <i>d_object</i> is not a hierarchical property.

Examples

```
dbCreateRangeProp(cv "prop1" "int" 5 3 8)  
dbCreateRangeProp(cv "prop1" "float" 100.0 0 "infinity")
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Property Creation and Modification Functions](#)

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

<i>g_object</i>	The database object which can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.
<i>t_value</i>	Value of the property.
<i>l_string</i>	Array of strings enumerating the possible values of the property, one of which must equal <i>t_value</i> .

Value Returned

<i>d_prop</i>	An enumerated property belonging to the database object.
<i>nil</i>	A property named <i>t_name</i> already exists on the specified object, or if the initial value is not in the enumerated property range passed in. Also returns <i>nil</i> if an error occurs.

Examples

```
dbCreateEnumProp(cv "prop1" "apple" list("apple" "pear" "peach"))  
dbCreateEnumProp(cv "prop1" "float" list("sink" "float" "beach" "pool"))
```

Related Topics

Property Creation and Modification Functions

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

<i>g_object</i>	The database object which can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.

Value Returned

<i>d_prop</i>	A hierarchical property belonging to the database object.
nil	A property named <i>t_name</i> already exists on the specified object or if <i>d_object</i> is invalid or is not a hierarchical property.

Examples

```
dbCreateHierProp(dbId "HProp")
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>d_object</i>	Can be a <i>dbId</i> , <i>propBagId</i> , or <i>ddId</i> .
<i>t_name</i>	Name of the property to be searched.

Value Returned

<i>d_propId</i>	<i>PropId</i> if property of the given name is found on the object
<i>nil</i>	Returns <i>nil</i> otherwise.

Examples

```
propId = dbFindProp(instId "myProp")
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>d_object</i>	Can be a <i>dbId</i> , <i>propBagId</i> , or <i>ddId</i> .
<i>t_name</i>	Name of the property to be searched.

Value Returned

<i>d_propId</i>	<i>PropId</i> if the property of the given name is found on the object.
<i>nil</i>	Returns <i>nil</i> otherwise.

Examples

```
propId = dbGetPropByName(instId "myProp")
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>g_object</i>	The database object which can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property. If <i>t_name</i> exists and is a range or enumerated property, it is replaced by a scalar property.
<i>t_propType</i>	Type of the property.
<i>g_value</i>	Value of the property; must be consistent with <i>t_propType</i> .

Value Returned

<i>d_prop</i>	A scalar property belonging to the database object.
<i>nil</i>	Any argument is in error, or <i>d_object</i> is not a hierarchical property.

Examples

```
dbReplaceProp(dbId "netNumber" "int" 12)  
dbReplaceProp(dbId "drawGrid?" "boolean" t)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Property Creation and Modification Functions](#)

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*.

Arguments

<i>g_object</i>	The database object which can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.
<i>t_propType</i>	Type of the property.
<i>g_value</i>	Value of the property. <i>g_value</i> , <i>g_lBound</i> , and <i>g_uBound</i> must be consistent with <i>t_propType</i> .
<i>g_lBound</i>	Lower bound of the range of values.
<i>g_uBound</i>	Upper bound of the range of values.

Value Returned

<i>d_prop</i>	A range property belonging to the database object.
<i>nil</i>	The value is not in the given range or if <i>d_object</i> is not a hierarchical property.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Examples

```
dbReplaceRangeProp(dbId "prop" "float" 120.0 0 nil)
dbReplaceRangeProp(dbId "prop" "int" 5 3 12)dbReplaceRangeProp
```

Related Topics

Property Creation and Modification Functions

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

<i>g_object</i>	The database object which can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.
<i>t_value</i>	Value of the property.
<i>l_string</i>	Array of strings enumerating the possible values of the property, one of which must equal <i>t_value</i> .

Value Returned

<i>d_prop</i>	An enumerated property belonging to the database object.
<i>nil</i>	The value is not in the enumerated property range or if <i>d_object</i> is not a hierarchical property.

Examples

```
dbReplaceEnumProp(dbId "prop" "grape" list("orange" "grape" "apricot"))
```

Related Topics

Property Creation and Modification Functions

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

<i>g_object</i>	The database object which can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.

Value Returned

<i>d_prop</i>	A hierarchical property belonging to the database object.
<i>nil</i>	<i>d_object</i> is not valid or <i>t_name</i> is not a string.

Examples

```
dbReplaceHierProp(inst "prop1")
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>g_object1</i>	Database object the properties are copied from. The object can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>g_object2</i>	Database object the properties of the first database object are copied to. The object can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .

Value Returned

<i>t</i>	The copy is successful.
<i>nil</i>	The properties are not copied or if they are only partially copied.

Examples

```
dbCopyProp(dbId1 dbId2)
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>d_dbId1</i>	Database object the properties are copied from. The object can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>d_dbId2</i>	Database object the properties of the first database object are copied to. The object can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .

Value Returned

<i>t</i>	The copy is successful.
<i>nil</i>	The properties are not copied or if they are only partially copied.

Examples

```
dbCopyPropList(i1 i2)
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>d_object</i>	Database ID of the object whose properties are being replaced.
<i>l_props</i>	List(propName propType propValue) List of properties. where, ■ propName Name of the property ■ propType Type of property ■ propValue Value of the property.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

t	List of properties on a given object are reset with the specified list of properties.
nil	Returned if an error occurs.

Examples

```
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)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Property Creation and Modification Functions

dbDeletePropByName

```
dbDeletePropByName (
    g_object
    t_name
)
=> t / nil
```

Description

Deletes a property from an object.

Arguments

<i>g_object</i>	The database object can be a <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .
<i>t_name</i>	Name of the property.

Value Returned

<i>t</i>	The property is not found or if the property is found and deleted.
<i>nil</i>	The property is not deleted.

Note: This function might return *t* even if an error occurs.

Examples

```
dbDeletePropByName (dbId1 "myProp")
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>d_srcPropId</i>	ID of the source property to be copied.
<i>g_dstObjId</i>	Destination object where the property is to be copied. This destination object can be a property, <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .

Value Returned

<i>d_dstPropId</i>	The property ID of the destination property.

Examples

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)
```

Related Topics

[Property Creation and Modification Functions](#)

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

<i>d_srcPropId</i>	Source property to be merged.
<i>g_dstObjId</i>	Destination object where the property is to be merged. The destination object can be a property, <i>dbId</i> , <i>ddId</i> , or <i>dbBagId</i> .

Value Returned

<i>d_dstPropId</i>	The property ID of the merged property.

Examples

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.

Related Topics

[Property Creation and Modification Functions](#)

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

<i>g_object</i>	Database object or property bag for which the given list of properties will be replaced.
<i>l_params</i>	List of properties. This list is a <i>list of sub-lists</i> . Each sub-list should contain three elements: a property name, a property type (acceptable to <code>dbCreateProp</code>) 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.

Examples

```
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"))))))
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Property Creation and Modification Functions](#)

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

<i>d_object</i>	Database object for which the given list of properties will be replaced.
<i>t_name</i>	List of properties.

Value Returned

<i>d_propId</i>	The properties are replaced successfully.
<i>nil</i>	The properties are not replaced successfully.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Property Creation and Modification Functions

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.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbAddObjectToGroup](#)

[dbChangeGroupType](#)

[dbCreateGroup](#)

[dbDeleteGroupByName](#)

[dbDeleteObjectFromGroup](#)

[dbGetGroupByName](#)

[dbGetGroupsWithDef](#)

[dbMoveGroupMember](#)

[dbDeleteGroupMemberFromGroup](#)

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

<i>d_group</i>	Database object of the group.
<i>d_object</i>	Database object to be added to the group.

Value Returned

<i>d_groupMem</i>	The <i>dbObject</i> of the group member if the database is successfully added to the group.
<i>nil</i>	The object is not added.

Examples

```
dbAddObjectToGroup(groupId objId)
```

Related Topics

Group Creation and Modification Functions

dbChangeGroupType

```
dbChangeGroupType (  
    d_groupId  
    l_groupType  
)  
=> t / nil
```

Description

Changes the type of a group.

Arguments

<i>d_groupId</i>	ID of the group to be changed.
<i>l_groupType</i>	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 <i>l_groupType</i> see Database Access Function Argument Types .

Value Returned

t	The group type is successfully changed
nil	The group type is not changed.

Related Topics

[Group Creation and Modification Functions](#)

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.

Arguments

<i>d_cellViewID</i>	The database object.
<i>t_groupName</i>	Name of the group to be created.
<i>l_groupType</i>	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 <i>l_groupType</i> see Database Access Function Argument Types .
<i>t_groupDefName</i>	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

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_group</i>	The <i>dbObject</i> of the group if the creation is successful.
<i>nil</i>	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.

Related Topics

[Group Creation and Modification Functions](#)

dbDeleteGroupName

```
dbDeleteGroupName(  
    d_cellView  
    t_name  
)  
=> t / nil
```

Description

Deletes a group and its members from a cellview.

Arguments

<i>d_cellView</i>	Cellview from which the group is to be deleted.
<i>t_name</i>	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.

Examples

```
dbDeleteGroupName(cv "GROUP")
```

Related Topics

[Group Creation and Modification Functions](#)

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

<i>d_group</i>	Database object of the group.
<i>d_object</i>	Database object to be deleted from the group.

Value Returned

t	The object is deleted.
nil	The object is not deleted.

Examples

```
dbDeleteObjectFromGroup(groupId objId)
```

Related Topics

Group Creation and Modification Functions

dbGetGroupName

```
dbGetGroupName (
    d_cellviewId
    t_groupName
)
=> d_groupId / nil
```

Description

Returns the database ID of the group specified by name.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_groupName</i>	The name of the group.

Value Returned

<i>d_groupId</i>	Returns the database ID of the group.
<i>nil</i>	The named group does not exist. An error occurred during execution.

Related Topics

Group Creation and Modification Functions

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

<i>d_cellViewID</i>	ID of the database object.
<i>t_groupDefName</i>	Name of the group definition.

Value Returned

<i>d_group</i>	List of groups is returned if the cellview with the specified ID exists.
<i>nil</i>	<i>nil</i> is returned if the cellview with the specified ID does not exist.

Examples

```
dbGetGroupsWithDef( )
```

Related Topics

[Group Creation and Modification Functions](#)

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

<i>d_groupMem1</i>	<i>dbObject</i> of the group member to be moved to a new position.
<i>d_groupMem2</i>	<i>dbObject</i> of the group member whose position is right before <i>d_groupMem1</i> when the move is completed. If not passed or nil, <i>d_groupMem1</i> is moved to the first position of the list.

Value Returned

t	The move was successful or if <i>d_groupMem1</i> equals <i>d_groupMem2</i> .
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)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Group Creation and Modification Functions

dbDeleteGroupMemberFromGroup

```
dbDeleteGroupMemberFromGroup(  
    d_groupMemId  
)  
=> t / nil
```

Description

Deletes a particular group member from the group to which it belongs.

Arguments

<i>d_groupMemId</i>	The group member ID.

Value Returned

t	The group member is deleted.
nil	The group member is not deleted.

Examples

```
dbDeleteGroupMemberFromGroup(groupMemId) => t
```

Related Topics

[Group Creation and Modification Functions](#)

Shape Transformation Functions

The following functions let you transform a shape or object by specifying modifications to the coordinates of the shape or object.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbTransformCellView](#)

[dbTransformPoint](#)

[dbTransformPointList](#)

[dbTransformBBox](#)

[dbConcatTransform](#)

[dbCanonicalizeAnyAngleTransform](#)

[dbConvertAnyAngleTransformFromDbToMY](#)

dbTransformCellView

```
dbTransformCellView(  
    d_cellViewId  
    f_mag  
    f_angle  
)  
=> t / nil
```

Description

Scales (magnifies) and/or rotates all shapes in a cellview.

Arguments

<i>d_cellViewId</i>	The database cellview ID.
<i>f_mag</i>	The magnification factor - 1.0 is a no-op.
<i>f_angle</i>	The rotation angle in degrees.

Value Returned

t	The function executed successfully.
nil	An error occurred.

Related Topics

[Shape Transformation Functions](#)

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

<i>l_point</i>	Coordinate pair to be transformed.
<i>l_list</i>	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

<i>l_point</i>	The coordinate pair of the location of the object after the transformation.
<i>nil</i>	The transformation is not successful.

Examples

```
dbTransformPoint( (1.5:2.5) list(0:10 "R90" 2.5) )  
=> (-6.250000 13.750000)
```

Related Topics

[Shape Transformation Functions](#)

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

<i>l_points</i>	The array of coordinate pairs to be transformed.
<i>l_transform</i>	<code>list(l_offset t_transform [f_scale])</code> ■ A transformation object. <code>l_offset: list(f_offsetX f_offsetY)</code> ■ An offset vector. <code>t_transform: A transformation name.</code> 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). <code>f_scale</code> Default value is 1.0.

Value Returned

<i>l_points</i>	The list of transformed points.
<i>nil</i>	The transformation was not successful.

Examples

```
dbTransformPointList(  
list((0:0) (0:1) (0:2) (0:3))  
list(0:0 "R90" 1.0)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

```
)  
=>  
( (0.0 0.0)  
  (-1.0 0.0)  
  (-2.0 0.0)  
  (-3.0 0.0) )
```

Related Topics

[Shape Transformation Functions](#)

dbTransformBBox

```
dbTransformBBox(  
    l_bBox  
    l_list  
)  
=> l_bBox_new
```

Description

Transforms a bounding box.

Arguments

<i>l_bBox</i>	A list representing the coordinates of a bounding box.
<i>l_list</i>	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

<i>l_bBox_new</i>	A list of the coordinates of the new bounding box.

Examples

```
dbTransformBBox(list(1.5:2.5 0:10) list(10:20 "R90"))  
=> ((0.0 21.5) (7.5 20.0))
```

Related Topics

[Shape Transformation Functions](#)

dbConcatTransform

```
dbConcatTransform(  
    l_transform1  
    l_transform2  
)  
=> l_transform_new
```

Description

Merges two individual transformations into a single new transformation.

Arguments

<i>l_transform1</i>	Defines the first transformation in terms of the displacement vector for the transformation, the rotation or reflection of the objects, and the magnification (optional).
<i>l_transform2</i>	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

<i>l_transform_new</i>	The new transformation resulting from the merged transformations.

Examples

```
dbConcatTransform(list(0:0 "R90") list(1.5:2.5 "R90"))  
=> ((1.500000 2.500000) "R180" 1.000000)
```

In the following example, figure F is moved to the same position and orientation as the two successive calls to dbMoveFig.

```
(dbMoveFig F cv (dbConcatTransform T1 T2))
```

The figures are moved in the following order:

```
(dbMoveFig F cv T1)  
(dbMoveFig F cv T2)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Shape Transformation Functions

dbCanonicalizeAnyAngleTransform

```
dbCanonicalizeAnyAngleTransform(  
    g_anyAngleTransform...  
)  
=> list(t_orient f_acuteAngleDeg)
```

Description

(Virtuoso MultiTech Framework) 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 [dbIsId](#).

Value Returned

```
list(t_orient f_acuteAngleDeg)
```

A list with two elements:

```
(t_orient f_acuteAngleDeg)
```

where,

- *t_orient*
A string representing orientation. See [dbIsId](#).
- *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)
```


Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Shape Transformation Functions

dbConvertAnyAngleTransformFromDbToMY

```
dbConvertAnyAngleTransformFromDbToMY(  
    t_orient  
    f_angleDeg  
)  
=> list(f_angleDeg g_MY)
```

Description

(Virtuoso MultiTech Framework) 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

<i>t_orient</i>	A string representing orientation. See dbIsId .
<i>f_angleDeg</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Examples

The following example checks the orientation is flipped:

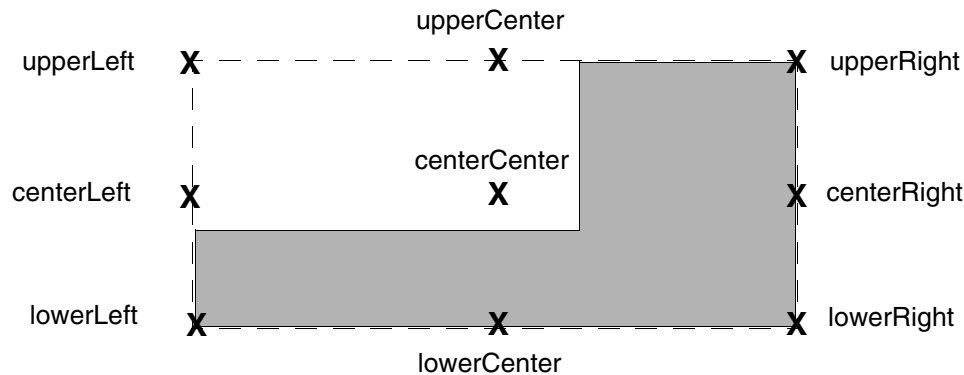
```
(defun flipsOrientation (t_orient)
  (cadr dbConvertAnyAngleTransformFromDbToMY(t_orient 0.0)))
```

Related Topics

[Shape Transformation Functions](#)

Bounding Box Functions

A bounding box has several points. The figure shows 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			

This topic provides a list of Cadence® SKILL functions associated with bounding box points.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[lowerLeft](#)

[upperRight](#)

[leftEdge](#)

[rightEdge](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

bottomEdge

topEdge

centerBox

isLocation

isBBox

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

<i>l_bBox</i>	A list representing a bounding box.

Value Returned

<i>l_list</i>	A list representing the lower left point of the bounding box.

Examples

```
lowerLeft( list( list(0 0) list(5 5)) => list(0 0)
```

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

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

<i>l_bBox</i>	A list representing a bounding box.

Value Returned

<i>l_list</i>	A list representing the upper right point of the bounding box.

Examples

```
upperRight( list( list(0 0) list(5 5)) => list(5 5)
```

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

leftEdge

```
leftEdge(  
    d_object  
)  
=> g_coordinate / nil
```

Description

Returns the lower left point's X coordinate of the figure's bounding box.

Arguments

<i>d_object</i>	Database ID of a figure.

Value Returned

<i>g_coordinate</i>	The X coordinate of the lower left point of the bounding box.
<i>nil</i>	The object is not a database figure or it does not have a bounding box.

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

rightEdge

```
rightEdge(  
    d_object  
)  
=> g_coordinate / nil
```

Description

Returns the upper right point's X coordinate of the figure's bounding box.

Arguments

<i>d_object</i>	Database ID of a figure.

Value Returned

<i>g_coordinate</i>	The X coordinate of the upper right point of the bounding box.
<i>nil</i>	The object is not a database figure or it does not have a bounding box.

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

bottomEdge

```
bottomEdge(  
    d_object  
)  
=> g_coordinate / nil
```

Description

Returns the lower left point's Y coordinate of the figure's bounding box.

Arguments

<i>d_object</i>	Database ID of a figure.

Value Returned

<i>g_coordinate</i>	The Y coordinate of the lower left point of the bounding box.
<i>nil</i>	The object is not a database figure or it does not have a bounding box.

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

topEdge

```
topEdge (
    d_object
)
=> g_coordinate / nil
```

Description

Returns the upper right point's Y coordinate of the figure's bounding box.

Arguments

<i>d_object</i>	A database figure.

Value Returned

<i>g_coordinate</i>	The Y coordinate of the upper right point of the bounding box.
<i>nil</i>	The object is not a database figure or it does not have a bounding box.

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

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

<i>l_bBox</i>	A list representing a bounding box.

Value Returned

<i>l_point</i>	A list representing the center point of the bounding box.
<i>nil</i>	The bounding box is not valid.

Examples

```
centerBox( list( 0:0 9:9 ) ) => ( 4 4 )
```

```
centerBox( list(0.0:0.0 9.0:9.0 ) ) => ( 4.5 4.5 )
```

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

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

<i>l_point</i>	A list representing a point.

Value Returned

t	The point is properly defined.
nil	The point is not properly defined.

Examples

```
isLocation( list( 0 0 )) => t
```

Related Topics

[Database Access Function Argument Types](#)

[Bounding Box Functions](#)

isBBox

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

Description

Validates that the given bounding box is properly defined. This function should not be used with Pcells.

Arguments

<i>l_bBox</i>	A list representing a bounding box.

Value Returned

t	The bounding box is properly defined.
nil	The bounding box is not properly defined.

Examples

```
isBBox( list( list( 0 0 ) list(5 5)) ) => t
```

Related Topics

[Bounding Box Functions](#)

Property Bag Functions

The property bag is a database object that contains only properties and is identified by *dbBagId*. This topic provides a list of Cadence® SKILL functions associated with the property bag. They can be used to create or modify properties in the property bag.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbOpenBag](#)

[dbReOpenBag](#)

[dbGetBag](#)

[dbGetBagOwner](#)

[dbCloseBag](#)

[dbPurgeBag](#)

[dbRefreshBag](#)

[dbMakeBagCurrent](#)

[dbSaveBag](#)

[dbIsBagReadable](#)

[dbIsBagWritable](#)

[dbGetBagTimeStamp](#)

[dbFindOpenBag](#)

[dbGetBagDdId](#)

[dbGetOpenBags](#)

dbOpenBag

```
dbOpenBag (
    g_ddObject
    t_mode
)
=> g_bagId / nil
```

Description

Opens the library, cell, or view property bag in the given mode.

Arguments

<i>g_ddObject</i>	<i>ddId</i> of the library, cell, or view.
<i>t_mode</i>	Mode in which the property bag will be opened. The mode can be one of the following values: ■ <i>r</i> Opens the property bag in read mode. The property bag must already exist. ■ <i>a</i> Opens the existing property bag in append mode. If the property bag does not exist, it is created. ■ <i>w</i> 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

<i>g_bagId</i>	<i>dbBagId</i> object if the property bag opens successfully.
<i>nil</i>	It fails to open the property bag.

Examples

```
dbOpenBag(libId "a")
```


Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Soft Connect Terminal Functions

dbReOpenBag

```
dbReOpenBag (
    g_bagId
    t_mode
)
=> t / nil
```

Description

Reopens the given *g_bagId* for the given mode.

Arguments

<i>g_bagId</i>	<i>g_bagId</i> to be reopened.
<i>t_mode</i>	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.

Examples

```
dbReOpenBag( bagId "a" )
```

Related Topics

[Soft Connect Terminal Functions](#)

dbGetBag

```
dbGetBag (
    g_ddObject
)
=> g_bagId / nil
```

Description

Opens the property bag in read mode for the given *ddId* object.

Arguments

<i>g_ddObject</i>	<i>ddId</i> of library, cell, or view.

Value Returned

<i>g_bagId</i>	The bag is opened successfully.
<i>nil</i>	If the property bag is newly created but not yet saved to disk.

Examples

```
dbGetBag( libId )
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<i>g_bagId</i>	The ID of a property bag.

Value Returned

<i>g_ddId</i>	The ID of the property bag owner.
<i>nil</i>	The function failed.

Examples

```
dbGetBagOwner( bagId )
```

Related Topics

[Soft Connect Terminal Functions](#)

dbCloseBag

```
dbCloseBag(  
    g_bagId  
    [t_mode]  
)  
=> t / nil
```

Description

Closes the property bag in the given mode. If closing for append or write, the function decrements the write count.

Arguments

<i>g_bagId</i>	The ID of the property bag.
<i>t_mode</i>	Mode to close the bag. If omitted, it closes in read mode.

Value Returned

<i>t</i>	The bag is closed successfully.
<i>nil</i>	It fails to close the bag.

Examples

```
dbCloseBag( bagId "a" )
```

Related Topics

[Soft Connect Terminal Functions](#)

dbPurgeBag

```
dbPurgeBag (
    g_bagId
)
=> t / nil
```

Description

Forces a property bag to close and removes it from virtual memory.

Arguments

<i>g_bagId</i>	Property bag to purge.

Value Returned

t	The property bag is purged.
nil	The property bag cannot be purged.

Examples

```
dbPurgeBag ( bagId )
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<i>g_bagId</i>	Property bag to be refreshed.

Value Returned

t	The bag is refreshed.
nil	The property bag cannot be refreshed.

Examples

```
dbRefreshBag( bagId )
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<i>g_bagId</i>	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.

Examples

```
dbMakeBagCurrent ( bagId )
```

Related Topics

[Soft Connect Terminal Functions](#)

dbSaveBag

```
dbSaveBag (
    g_bagId
)
=> t / nil
```

Description

Writes the virtual memory image of the property bag to disk.

Arguments

<i>g_bagId</i>	Property bag to be saved.

Value Returned

t	The bag is saved.
nil	The property bag cannot be written to disk.

Examples

```
dbSaveBag ( bagId )
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<i>g_ddObject</i>	<i>ddId</i> 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.

Examples

```
dbIsBagReadable( libId )
```

Related Topics

[Soft Connect Terminal Functions](#)

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 [`ddlIsObjWritable`](#).

Arguments

<i>g_ddObject</i>	<i>ddlId</i> of library, cell, or view.

Value Returned

<code>t</code>	The property bag already exists for the given <i>ddlId</i> 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 <i>ddlId</i> Object.
<code>nil</code>	Returns <code>nil</code> otherwise.

Examples

```
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
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

```
ddIsObjWritable(ddGetObj("libFolder"))  
=> nil
```

Related Topics

[Soft Connect Terminal Functions](#)

dbGetBagTimeStamp

```
dbGetBagTimeStamp(  
    g_bagId  
)  
=> t_time / nil
```

Description

Returns the last time that the bag was modified.

Arguments

<i>g_bagId</i>	<i>dbBagId</i> object.

Value Returned

<i>t_time</i>	Last time the bag was modified.
<i>nil</i>	The <i>bagId</i> is invalid.

Examples

```
dbGetBagTimeStamp
```

Related Topics

[Soft Connect Terminal Functions](#)

dbFindOpenBag

```
dbFindOpenBag (
    d_ddId
)
=> g_bagId / nil
```

Description

Finds an opened bag in virtual memory for the specified *d_ddId*.

Arguments

<i>d_ddId</i>	The object ID.

Value Returned

<i>g_bagId</i>	The ID of the object's property bag.
<i>nil</i>	The <i>g_bagId</i> is invalid.

Examples

```
dbFindOpenBag ( ddId )
```

Related Topics

[Soft Connect Terminal Functions](#)

dbGetBagDdId

```
dbGetBagDdId(  
    g_bagId  
)  
=> d_ddId / nil
```

Description

Returns the *ddId* associated with the specified *bagId*.

Arguments

<i>g_bagId</i>	<i>dbBagId</i> object.

Value Returned

<i>d_ddId</i>	The property bag's dd ID.
nil	The given <i>g_bagId</i> is invalid.

Examples

```
dbGetBagDdId( bagId )
```

Related Topics

[Soft Connect Terminal Functions](#)

dbGetOpenBags

```
dbGetOpenBags (  
    )  
=> d_bags / nil
```

Description

Returns a list of the currently open property bags.

Arguments

None

Value Returned

<i>d_bags</i>	A list of currently open property bags.
<i>nil</i>	There are no currently open property bags.

Related Topics

[Soft Connect Terminal Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

dbIsBagId

```
dbIsBagId(  
    g_bagId  
)  
=> t / nil
```

Description

Returns `t` if the specified ID is a valid *bagId*, otherwise, it returns `nil`.

Arguments

<i>g_bagId</i>	<i>dbBagId</i> object.

Value Returned

<code>t</code>	The specified ID is a valid <i>bagId</i> .
<code>nil</code>	The <i>bagId</i> is invalid.

Examples

```
dbIsBagId( bagId )
```

Related Topics

[Soft Connect Terminal Functions](#)

Layer Boolean Functions

Layer Boolean SKILL functions let you create certain geometries using Boolean operators, such as `AND` and perform relational selection operations such as `ENCLOSE` on existing geometries.

These SKILL functions take input from one or two lists of database objects of shapes or polygons while the output of the function is created as database objects on a specified output layer. The object IDs of the created database objects are returned.

This topic provides a list of Cadence® SKILL functions associated with the layer Boolean functions.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbLayerAnd](#)

[dbLayerAndNot](#)

[dbLayerOr](#)

[dbLayerXor](#)

[dbLayerSize](#)

[dbLayerTile](#)

[dbLayerStraddle](#)

[dbLayerNoHoles](#)

[dbLayerHoles](#)

[dbLayerInside](#)

[dbLayerOutside](#)

[dbLayerEnclose](#)

dbLayerAnd

```
dbLayerAnd(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    l_dbIdList2  
    [ x_numVertices ]  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list 1.
<i>l_dbIdList2</i>	List of objects from input list 2.
<i>x_numVertices</i>	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects derived from the overlap of objects on the input lists.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Examples

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`.

```
list1 = geGetSelSet(cvId1)
list2 = geGetSelSet(cvId2)
m3Out = dbLayerAnd(cvId "metal3" list1 list2)
```

Related Topics

[Layer Boolean Functions](#)

dbLayerAndNot

```
dbLayerAndNot (
    d_cellViewId
    t_layerName
    l_dbIdList1
    l_dbIdList2
    [ x_numVertices ]
    [ g_mfgGrid ]
)
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list 1.
<i>l_dbIdList2</i>	List of objects from input list 2.
<i>x_numVertices</i>	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerAndNot(cvId "metal3" m1Rect m2Rect)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Layer Boolean Functions](#)

dbLayerOr

```
dbLayerOr(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    [ l_dbIdList2 ]  
    [ x_numVertices ]  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list 1.
<i>l_dbIdList2</i>	List of objects from input list 2.
<i>x_numVertices</i>	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerOr(cvId "metal3" m1Rect m2Rect)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Layer Boolean Functions](#)

dbLayerXor

```
dbLayerXor(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    l_dbIdList2  
    [ x_numVertices ]  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list 1.
<i>l_dbIdList2</i>	List of objects from input list 2.
<i>x_numVertices</i>	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerXor(cvId "metal3" m1Rect m2Rect)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Layer Boolean Functions](#)

dbLayerSize

```
dbLayerSize(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    n_value  
    [ x_numVertices ]  
    [ g_mfgGrid ]  
    [ x_conicSides ]  
)  
=> l_dbIdList
```

Description

Performs an oversizing or undersizing operation on the object specified in the input list.

Arguments

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list 1.
<i>n_value</i>	Sizing value.
<i>x_numVertices</i>	Target vertex count of output objects. If you do not specify this number, the default vertex count (195) is used.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.
<i>x_conicSides</i>	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

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerSize(cvId "metal3" m1Rect 0.5)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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`.

```
M2Out = dbLayerSize(cvId "Metal2" geGetSelSet() 0.25 2000 0.001 100)
```

Related Topics

[Layer Boolean Functions](#)

dbLayerTile

```
dbLayerTile(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

Fractures edges of the specified object on the input list into vertical trapezoids.

Arguments

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list 1.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerTile(cvId "metal3" m1Rect)
```

Related Topics

[Layer Boolean Functions](#)

dbLayerStraddle

```
dbLayerStraddle(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    l_dbIdList2  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview in which to create the output objects.
<i>t_layerName</i>	The layer on which output objects are to be created.
<i>l_dbIdList1</i>	List of objects from input layer 1.
<i>l_dbIdList2</i>	List of objects from input layer 2.
<i>g_mfgGrid</i>	The manufacturing grid to which the output objects must be snapped.

Value Returned

<i>l_dbIdList</i>	List of objects resulting from the layer operation.

Examples

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`.

```
m3Out = dbLayerStraddle(cvId "metal3" m1Rect m2Rect)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Layer Boolean Functions](#)

dbLayerNoHoles

```
dbLayerNoHoles(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerNoHoles(cvId "metal3" m1Rect m2Rect)
```

Related Topics

[Layer Boolean Functions](#)

dbLayerHoles

```
dbLayerHoles(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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 `m3Out`.

```
m3Out = dbLayerHoles(cvId "metal3" m1Rect m2Rect)
```

Related Topics

[Layer Boolean Functions](#)

dbLayerInside

```
dbLayerInside(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    l_dbIdList2  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<code>d_cellViewId</code>	Database ID of the cellview in which to create the output objects.
<code>t_layerName</code>	Layer name to be used for creating the output objects.
<code>l_dbIdList1</code>	List of objects from input layer 1.
<code>l_dbIdList2</code>	List of objects from input layer 2.
<code>g_mfgGrid</code>	The manufacturing grid to which the output objects must be snapped.

Value Returned

<code>l_dbIdList</code>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerInside(cvId "metal3" m1Rect m2Rect)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Layer Boolean Functions](#)

dbLayerOutside

```
dbLayerOutside(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    l_dbIdList2  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview in which to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input layer 1.
<i>l_dbIdList2</i>	List of objects from input layer 2.
<i>g_mfgGrid</i>	The manufacturing grid to which the output objects must be snapped.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerOutside(cvId "metal3" m1Rect m2Rect)
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[Layer Boolean Functions](#)

dbLayerEnclose

```
dbLayerEnclose(  
    d_cellViewId  
    t_layerName  
    l_dbIdList1  
    l_dbIdList2  
    [ g_mfgGrid ]  
)  
=> l_dbIdList
```

Description

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>t_layerName</i>	Layer name to be used for creating the output objects.
<i>l_dbIdList1</i>	List of objects from input list 1.
<i>l_dbIdList2</i>	List of objects from input list 2.
<i>g_mfgGrid</i>	The manufacture grid to snap the points of output shapes.

Value Returned

<i>l_dbIdList</i>	List of objects from the resulting layer operation.

Examples

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`.

```
m3Out = dbLayerEnclose(cvId "metal3" m1Rect m2Rect)
```

Related Topics

Layer Boolean Functions

Point Array Functions

This topic provides a list of Cadence® SKILL functions associated with point arrays.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbPointArrayAnd](#)

[dbPointArrayAndNot](#)

[dbPointArrayOr](#)

[dbPointArraySize](#)

[dbPointArrayXor](#)

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>l_pointList1</i>	Input list 1 with the list of point lists.
<i>l_pointList2</i>	Input list 2 with the list of point lists.
<i>x_maxPoints</i>	Maximum number of points each point array in the output point list can contain.

Value Returned

<i>l_pointList</i>	List of point lists derived from the overlap of shapes in the input lists.

Examples

```
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)
```

Related Topics

[Point Array Functions](#)

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>l_pointList1</i>	Input list 1 with the list of point lists.
<i>l_pointList2</i>	Input list 2 with the list of point lists.
<i>x_maxPoints</i>	Maximum number of points each point array in the output point list can contain.

Value Returned

<i>l_pointList</i>	List of points derived from non-overlapping areas from the input lists.

Examples

```
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)
```

Related Topics

Point Array Functions

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>l_pointList1</i>	Input list 1 with the list of point lists.
<i>l_pointList2</i>	Input list 2 with the list of point lists.
<i>x_maxPoints</i>	Maximum number of points each point array in the output point list can contain.

Value Returned

<i>l_pointList</i>	List of points derived from the union of shapes from the input lists.

Examples

```
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)
```

Related Topics

Point Array Functions

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>l_pointList</i>	Input list with the list of point lists.
<i>n_value</i>	The sizing value.
<i>x_maxPoints</i>	Maximum number of points each point array in the output point list can contain.

Value Returned

<i>l_pointList</i>	List of points derived after an over size or under size operation on the input list.

Examples

```
pointArray = '( ((1 1) (1 6) (6 6) (6 1)) )  
dbPointArraySize( geGetEditCellView() pointArray 1.0)
```

Related Topics

[Point Array Functions](#)

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

<i>d_cellViewId</i>	Database ID of the cellview to create the output objects.
<i>l_pointList1</i>	Input list 1 with the list of point lists.
<i>l_pointList2</i>	Input list 2 with the list of point lists.
<i>x_maxPoints</i>	Maximum number of points each point array in the output point list can contain.

Value Returned

<i>l_pointList</i>	List of points in both input lists that do not overlap points in the other input list.

Examples

```
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)
```

Related Topics

Point Array Functions

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.

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`.

This topic provides a list of Cadence® SKILL functions associated with soft connect terminals.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[dbSetTermSoftConnect](#)

[dbGetSoftConnectTermConnectToLayer](#)

[dbGetSoftConnectTermPinlessLayer](#)

[dbIsTermSoftConnect](#)

[dbSetSoftConnectTermConnectToLayer](#)

[dbSetSoftConnectTermPinlessLayer](#)

[dbIsTermSoftConnect](#)

[dbSetSoftConnectTermConnectToLayer](#)

[dbSetSoftConnectTermPinlessLayer](#)

[dbSoftConnectTermHasConnectToLayer](#)

[dbSoftConnectTermHasPinlessLayer](#)

dbSetTermSoftConnect

```
dbSetTermSoftConnect (
    d_termId
    g_value
)
=> t / nil
```

Description

Sets the `softConnect` attribute of the specified terminal to the given value.

Arguments

<code>d_termId</code>	The database ID of the terminal.
<code>g_value</code>	Value of the <code>softConnect</code> attribute. If the <code>softConnect</code> attribute of the terminal is set to <code>nil</code> , the associated <code>pinlessTerminalLayer</code> and <code>connectToLayer</code> attributes are removed (if already set).

Value Returned

<code>t</code>	The <code>softConnect</code> attribute has been successfully set.
<code>nil</code>	The <code>softConnect</code> attribute has not been set.

Examples

```
dbSetTermSoftConnect( term t)
```

Related Topics

[Soft Connect Terminal Functions](#)

dbGetSoftConnectTermConnectToLayer

```
dbGetSoftConnectTermConnectToLayer (
    d_termId
)
=> x_layer / nil
```

Description

Returns the `connectToLayer` of the `softConnect` terminal as a layer number.

Arguments

<code>d_termId</code>	The database ID of the terminal.

Value Returned

<code>x_layer</code>	The layer number of the <code>connectToLayer</code> .
<code>nil</code>	When <code>connectToLayer</code> is not set for the terminal, an error is reported and <code>nil</code> is returned.

Examples

```
dbGetSoftConnectTermConnectToLayer( term )=> 3
```

Related Topics

[Soft Connect Terminal Functions](#)

dbGetSoftConnectTermPinlessLayer

```
dbGetSoftConnectTermPinlessLayer (
    d_termId
)
=> tx_layer / nil
```

Description

Returns the `pinlessTerminalLayer` of the `softConnect` terminal as a layer name or number.

Arguments

<code>d_termId</code>	The database ID of the terminal.

Value Returned

<code>tx_layer</code>	The layer name or number of the <code>pinlessTerminalLayer</code> .
<code>nil</code>	When <code>pinlessTerminalLayer</code> is not set for this terminal, an error is reported and <code>nil</code> is returned.

Examples

```
dbGetSoftConnectTermPinlessLayer( term )=> 20
```

Related Topics

[Soft Connect Terminal Functions](#)

dbIsTermSoftConnect

```
dbIsTermSoftConnect (
    d_termId
)
=> t / nil
```

Description

Checks the `softConnect` attribute value of the specified explicit, single-bit terminal.

Arguments

<code>d_termId</code>	The database ID of a explicit single-bit terminal.

Value Returned

<code>t</code>	The <code>softConnect</code> attribute of the specified terminal is set.
<code>nil</code>	The <code>softConnect</code> attribute of the specified terminal is not set or the terminal is an implicit or a multi-bit terminal.

Examples

```
dbIsTermSoftConnect( term )=> t
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<code>d_termId</code>	The database ID of the terminal.
<code>tx_layer</code>	The layer name or layer number of the terminal.

Value Returned

<code>t</code>	The <code>connectToLayer</code> on the specified <code>softConnect</code> terminal has been successfully set.
<code>nil</code>	An error occurred and the <code>connectToLayer</code> on the specified <code>softConnect</code> terminal could not be set.

Examples

```
dbSetSoftConnectTermConnectToLayer( term "metal2")=> t
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<code>d_termId</code>	The database ID of the terminal.
<code>tx_layer</code>	The layer name or layer number of the terminal.

Value Returned

<code>t</code>	The <code>pinlessTerminalLayer</code> on the specified <code>softConnect</code> terminal has been successfully set.
<code>nil</code>	An error occurred and the <code>pinlessTerminalLayer</code> on the specified <code>softConnect</code> terminal could not be set.

Examples

```
dbSetSoftConnectTermPinlessLayer( term "metall")=> t
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<code>d_termId</code>	The database ID of the terminal.

Value Returned

<code>t</code>	The terminal has the <code>softConnect connectToLayer</code> attribute set.
<code>nil</code>	The terminal is not a <code>softConnect</code> terminal or does not have the <code>connectToLayer</code> attribute set.

Examples

```
dbSoftConnectTermHasConnectToLayer( term )=> t
```

Related Topics

[Soft Connect Terminal Functions](#)

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

<code>d_termId</code>	The database ID of the terminal.

Value Returned

<code>t</code>	The terminal has the <code>softConnect pinlessTerminalLayer</code> attribute set.
<code>nil</code>	The terminal is not a <code>softConnect</code> terminal or does not have the <code>pinlessTerminalLayer</code> attribute set.

Examples

```
dbSoftConnectTermHasPinlessLayer( term )=> t
```

Related Topics

[Soft Connect Terminal Functions](#)

Information Retrieval Functions

This topic provides a list of Cadence® SKILL functions associated with information retrieval.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbDebugStats2](#)

[dbGetBuildInfo](#)

[dbGetFileDatabaseType](#)

[dbGetVersion](#)

[dbSetNIMsgMode](#)

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

<i>d_cellViewId</i>	The database cellview ID.
<i>x_segNum</i>	The segment number.
<i>g_objName</i>	The object name.
<i>g_outputFileName</i>	Output file name.

Value Returned

t	The function executed successfully.
nil	An error occurred.

Related Topics

[Information Retrieval Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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

<i>t_dbBuildInfo</i>	The release number, tarkit revision number, and the date and machine on which the database was built.

Examples

```
dbGetBuildInfo()  
"@(#)$OpenAccess: 02.00 s092 06/11/03 00:01 (cds12106) $"
```

Related Topics

[Information Retrieval Functions](#)

dbGetFileDatabaseType

```
dbGetFileDatabaseType(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> t_dbFileDatabaseType
```

Description

Returns the *t_libName*, *t_cellName*, and *t_viewName* database type.

Arguments

<i>t_libName</i>	The library name.
<i>t_cellName</i>	The cell name.
<i>t_viewName</i>	The view name.

Value Returned

<i>t_dbFileDatabaseType</i>	The library/cell/view database type.

Examples

```
dbGetFileDatabaseType("newTech18" "M3xM2xG" "layout")  
"OpenAccess"
```

Related Topics

[Information Retrieval Functions](#)

dbGetVersion

```
dbGetVersion(  
    )  
=> t_dbVersion
```

Description

Returns the OpenAccess database version number.

Arguments

None

Value Returned

t_dbVersion The OpenAccess database version number.

Examples

```
dbGetVersion()  
59
```

Related Topics

[Information Retrieval Functions](#)

dbSetNIMsgMode

```
dbSetNIMsgMode (  
    g_arg  
)  
=> t / nil
```

Description

Turns on/off the `not implemented` message.

Arguments

<i>g_arg</i>	Turns the <code>not implemented</code> message on or off. Valid Values: <code>t</code> (on), <code>nil</code> (off)

Value Returned

<code>t</code>	The function is executed successfully.
<code>nil</code>	<i>g_arg</i> was not set to <code>t</code> or <code>nil</code> .

Related Topics

[Information Retrieval Functions](#)

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
```

This topic provides a list of Cadence® SKILL functions associated with blockages.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

[dbCreateAreaBlockage](#)

[dbCreateAreaHalo](#)

[dbCreateLayerBlockage](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>l_points</i>	Specifies the list of points for the blockage.
<i>d_owner</i>	Optional argument indicating the owner ID of the blockage. Default is <code>nil</code> .

Value Returned

<i>d_areaBlockageId</i>	When successful, the blockage database ID is returned.
<code>nil</code>	<code>nil</code> is returned if the cellview ID is invalid.

Related Topics

[Database Access Function Argument Types](#)

[Blockage Functions](#)

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

<i>d_ownerId</i>	The owner ID of the areaHalo blockage.
<i>n_topOffset</i>	Integer or floating-point number indicating the top offset of the area halo.
<i>n_bottomOffset</i>	Integer or floating-point number indicating the bottom offset of the area halo.
<i>n_leftOffset</i>	Integer or floating-point number indicating the left offset of the area halo.
<i>n_rightOffset</i>	Integer or floating-point number indicating the right offset of the area halo.

Value Returned

<i>d_areaHaloId</i>	When successful, the area halo database ID is returned.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

Blockage Functions

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>tx_layer</i>	Name or number of the layer to be assigned to the blockage.
<i>t_type</i>	String argument indicating the blockage type. Valid Values: routing, viaRouting, wiring, fill, slot, pin, feedthru, Screen.
<i>l_points</i>	The list of points for the blockage.
<i>d_owner</i>	Optional argument indicating the owner of the blockage. Default is <code>nil</code> .

Value Returned

<i>d_layerBlockageId</i>	The blockage database ID if the creation is successful.
<code>nil</code>	<code>nil</code> is returned if the cellview ID is invalid.

Related Topics

[Database Access Function Argument Types](#)

[Blockage Functions](#)

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

<i>d_ownerId</i>	The database ID of the layer halo owner. The owner can only be an instance or a prBoundary.
<i>tx_layer</i>	Name or number of the layer which the layer halo is on.
<i>t_type</i>	Type of layer halo. Valid values: routing placement wiring fill slot pin feedthru Screen
<i>n_top</i>	Integer or floating-point number indicating the top offset of the layer halo.
<i>n_bottom</i>	Integer or floating-point number indicating the bottom offset of the layer halo.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

<i>n_left</i>	Integer or floating-point number indicating the left offset of the layer halo.
<i>n_right</i>	Integer or floating-point number indicating the right offset of the layer halo.

Value Returned

<i>d_layerHalo</i>	Returns the database ID of the layer halo.
<i>nil</i>	An error occurred during execution.

Related Topics

[Database Access Function Argument Types](#)

[Blockage Functions](#)

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`.

This topic provides a list of Cadence® SKILL functions associated with boundary.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateAreaBoundary](#)

[dbCreateClusterBoundary](#)

[dbCreateCoverObstruction](#)

[dbCreatePRBoundary](#)

[dbCreateSnapBoundary](#)

[dbFindAreaBoundaryByName](#)

[dbFindClusterBoundaryByName](#)

[dbSetBoundaryEdge](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>l_points</i>	Specifies the list of points for the boundary.
<i>t_name</i>	Optional argument that specifies the name of the area boundary.
<i>lt_edgeNames</i>	Optional argument that list the names of boundary edges.

Value Returned

<i>d_areaBoundaryId</i>	Returns the database ID of the boundary if the creation is successful.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

[Database Access Function Argument Types](#)

[Boundary Functions](#)

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

<i>d_clusterId</i>	The database ID of the cluster in which to create the boundary.
<i>t_name</i>	Name of the cluster boundary.
<i>l_points</i>	Specifies the list of points for the boundary.
<i>lt_edgeNames</i>	Optional argument that list the names of boundary edges.

Value Returned

<i>d_boundaryId</i>	Returns the database ID of the boundary if the creation is successful.
<i>nil</i>	<i>nil</i> is returned if the cluster ID is invalid.

Related Topics

[Database Access Function Argument Types](#)

[Boundary Functions](#)

dbCreateCoverObstruction

```
dbCreateCoverObstruction(  
    d_prBoundaryId  
    x_topLayerMaskNumber  
    [ g_allowPGNet ]  
)  
=> d_coverObstructionId / nil
```

Description

Creates a cover obstruction object associated with the specified prBoundary object.

Arguments

<i>d_prBoundaryId</i>	The database ID of the prBoundary object.
<i>x_topLayerMaskNumber</i>	Specifies the mask number of the top layer up to which the obstruction will be created.
<i>g_allowPGNet</i>	Optional argument, which indicates whether power and ground nets can be routed over the areas defined by the cover obstruction.

Value Returned

<i>d_coverObstructionId</i>	Returns the database ID of the cover obstruction object if the creation is successful.
<i>nil</i>	<i>nil</i> is returned if the prBoundary ID is invalid.

Related Topics

[Boundary Functions](#)

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

<i>d_cellViewId</i>	The database ID of the cellview.
<i>l_points</i>	Specifies the points of the prBoundary.
<i>lt_edgeNames</i>	Optional argument that list the names of edges.

Value Returned

<i>d_boundaryId</i>	The database ID of the prBoundary if the creation is successful.
<i>nil</i>	If a prBoundary already exists for the cellview, a warning is generated and <i>nil</i> is returned.

Related Topics

[Database Access Function Argument Types](#)

[Boundary Functions](#)

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

<i>d_cellViewId</i>	The database ID of the cellview.
<i>l_points</i>	Specifies the points of the snap boundary.
<i>lt_edgeNames</i>	Optional argument that list the names of edges.

Value Returned

<i>d_boundaryId</i>	The database ID of the snap boundary if the creation is successful.
<i>nil</i>	If a snap boundary already exists for the cellview, a warning is generated and <i>nil</i> is returned.

Related Topics

Database Access Function Argument Types

Boundary Functions

dbFindAreaBoundaryByName

```
dbFindAreaBoundaryByName (
    d_cellviewId
    t_name
)
=> d_areaBoundaryId / nil
```

Description

Finds the area boundary specified by name and cellview ID.

Arguments

<i>d_cellviewId</i>	The cellview ID in which to search for the specified area boundary.
<i>t_name</i>	Name of the area boundary to be found.

Value Returned

<i>d_areaBoundaryId</i>	The database ID of the area boundary when successful.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

[Boundary Functions](#)

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

<i>d_clusterId</i>	The cluster ID in which to search for the specified cluster boundary.
<i>t_name</i>	Name of the cluster boundary to be found.

Value Returned

<i>d_boundaryId</i>	The database ID of the boundary when successful.
<i>nil</i>	<i>nil</i> is returned if the cluster ID is invalid.

Related Topics

[Boundary Functions](#)

dbSetBoundaryEdge

```
dbSetBoundaryEdge(  
    d_boundaryId  
    l_points  
    [ lt_edgeNames ]  
)  
=> t / nil
```

Description

Sets the points (edges) and edge names for the specified boundary.

Arguments

<i>d_boundaryId</i>	The database ID of the boundary to be set.
<i>l_points</i>	Specifies a list of points for the boundary.
<i>lt_edgeNames</i>	Optional argument that list the names of edges.

Value Returned

<i>t</i>	The set of the boundary edges successfully.
<i>nil</i>	<i>nil</i> is returned if the boundary ID is invalid.

Related Topics

[Database Access Function Argument Types](#)

[Boundary Functions](#)

Bus Definition and Bit Order Functions

This topic provides a list of Cadence® SKILL functions associated with bus definition and bit order.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateBusNetDef](#)

[dbCreateBusTermDef](#)

[dbCreateVectorInstDef](#)

[dbFindBusNetDefByName](#)

[dbFindBusTermDefByName](#)

[dbFindVectorInstDefByName](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_baseName</i>	Specifies the base name for the bus net definition.
<i>t_bitOrder</i>	The bit order for the bus net definition.
Valid Values: none, ascendingBitOrder, and descendingBitOrder	

Value Returned

<i>d_busNetDefId</i>	When successful, the ID of the bus net definition is returned.
<i>nil</i>	If an explicit oaBusNetDef with the specified name already exists, a warning is issued and <i>nil</i> is returned.

Examples

This creates a busNet definition with base name "in" with ascending bit order.

```
dbCreateBusNetDef(cv "in" "ascendingBitOrder")
```

Related Topics

[Bus Definition and Bit Order Functions](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_baseName</i>	Specifies the base name for the bus terminal definition.
<i>t_bitOrder</i>	The bit order for the bus terminal definition.
Valid Values: none, ascendingBitOrder, and descendingBitOrder	

Value Returned

<i>d_busTermDefId</i>	When successful, the ID of the busTermDef is returned.
<i>nil</i>	If an explicit bus terminal definition with the specified name already exists, a warning is issued and <i>nil</i> is returned.

Examples

This creates a bus term definition with base name "B" with ascending bit order.

```
dbCreateBusTermDef(cv "B" "ascendingBitOrder")
```

Related Topics

[Bus Definition and Bit Order Functions](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_baseName</i>	Specifies the base name for the vector instance.
<i>t_bitOrder</i>	The bit order for the vector instance.
	Valid Values: none, ascendingBitOrder, and descendingBitOrder

Value Returned

<i>d_vectorInstDefId</i>	When successful, the ID of the vector instance is returned.
<i>nil</i>	If an explicit vector instance with the specified name already exists, a warning is issued and <i>nil</i> is returned.

Examples

This creates a vectorInst definition with base name "A" and ascending bit order.

```
dbCreateVectorInstDef(cv "A" "ascendingBitOrder")
```

Related Topics

[Bus Definition and Bit Order Functions](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_baseName</i>	Specifies the base name for the bus net definition.

Value Returned

<i>d_busNetDefId</i>	If successful, the ID of the bus net definition is returned.
<i>nil</i>	<i>nil</i> is returned if the bus net definition does not exist.

Examples

```
dbFindBusNetDefByName (cv "in")
```

Related Topics

[Bus Definition and Bit Order Functions](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_baseName</i>	Specifies the base name for the bus term definition.

Value Returned

<i>d_busNetDefId</i>	If successful, the ID of the bus term definition is returned.
<i>nil</i>	<i>nil</i> is returned if the bus term definition does not exist.

Examples

```
dbFindBusTermDefByName (cv "B")
```

Related Topics

[Bus Definition and Bit Order Functions](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_baseName</i>	Specifies the base name for the vector instance definition.

Value Returned

<i>d_busNetDefId</i>	If successful, the ID of the vector instance definition is returned.
<i>nil</i>	<i>nil</i> is returned if the vector instance definition does not exist.

Examples

```
dbFindVectorInstDefByName (cv "A")
```

Related Topics

[Bus Definition and Bit Order Functions](#)

Cluster 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.

This topic provides a list of Cadence® SKILL functions associated with clusters.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateCluster](#)

[dbFindClusterByName](#)

dbCreateCluster

```
dbCreateCluster(  
    d_cellViewId  
    t_clusterName  
    t_clusterType  
)  
=> d_clusterId / nil
```

Description

Creates a cluster in the specified cellview.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.
<i>t_clusterName</i>	Name of the cluster.
<i>t_clusterType</i>	Indicates the type of cluster. Valid values: <i>inclusive</i> : Instances other than the instances in this cluster may be placed in the cluster's region. <i>exclusive</i> : Only instances in this cluster may be placed in the cluster's region. <i>suggested</i> : Instances in this cluster may be placed outside the cluster's region.

Value Returned

<i>d_clusterId</i>	The database ID of the cluster if the creation is successful.
<i>nil</i>	The creation fails. Failure can occur if either another cluster having the name specified already exists in the cellview, the specified <i>clusterType</i> is invalid, or the specified cellview is invalid

Related Topics

[Cluster Functions](#)

dbFindClusterByName

```
dbFindClusterByName(  
    d_cellViewId  
    t_clusterName  
)  
=> d_clusterId / nil
```

Description

Searches for a cluster in the specified cellview with the specified name.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.
<i>t_clusterName</i>	Name of the cluster.

Value Returned

<i>d_clusterId</i>	The database ID of the cluster when successful.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid or the specified cluster is not found.

Related Topics

[Cluster Functions](#)

Counter Functions

This topic provides a list of Cadence® SKILL functions associated with counters.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbDisableCellViewCounter](#)

[dbEnableCellViewCounter](#)

[dbGetCellViewCounter](#)

[dbIsConnCurrent](#)

[dbSetConnCurrent](#)

[dbSetInstHeadersCounter](#)

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.

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

<code>d_cellViewId</code>	The database ID of the cellview.

Value Returned

<code>t</code>	The updates were disabled.
<code>nil</code>	The cellview ID is invalid.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Examples

`dbDisableCellViewCounter(cv)`

Related Topics

[dbSave](#)

[Counter Functions](#)

dbEnableCellViewCounter

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

Description

Enables updating of the cellview's timestamp counter.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.

Value Returned

t	The updating was enabled.
nil	nil is returned if the cellview ID is invalid.

Related Topics

[Counter Functions](#)

dbGetCellViewCounter

```
dbGetCellViewCounter(  
    d_cellViewId  
)  
=> x_counter / nil
```

Description

Gets the timestamp counter value for the cellview.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.

Value Returned

<i>x_counter</i>	The counter of the cellview.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

[Counter Functions](#)

dbGetFileTimeStamp

```
dbGetFileTimeStamp(  
    d_cellViewId  
)  
=> t_time / nil
```

Description

Gets the last saved time for the cellview file.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.

Value Returned

<i>t_time</i>	The time the file was last saved.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

[Counter Functions](#)

dbIsConnCurrent

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

Description

Tests whether the value of the connectivity current property is equal to the cellview timestamp counter value.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.

Value Returned

<i>t</i>	The value of the connectivity current property is equal to the cellview's timestamp counter value.
<i>nil</i>	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.

Related Topics

[Counter Functions](#)

dbSetConnCurrent

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

Description

Sets the value of the connectivity current property to the cellview timestamp counter value.

Arguments

<i>d_cellViewId</i>	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.

Related Topics

[Counter Functions](#)

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 timestamp counter value.

Arguments

<i>d_cellViewId</i>	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.

Related Topics

[Counter Functions](#)

Interconnect Functions

This topic provides a list of Cadence® SKILL functions associated with interconnections.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateGuide](#)

[dbCreatePathSeg](#)

[dbCreateRoute](#)

[dbCreateVia](#)

[dbCreateCustomViaVariant](#)

[dbCreateStdViaVariant](#)

[dbFindViaVariantByName](#)

[dbGetViaLayer](#)

[dbIsCellViewStdViaMaster](#)

[dbSetPathSegPoints](#)

[dbSetViaParamDefault](#)

[dbSetViaShapePurpose](#)

[dbGetRouteAuthor](#)

[dbSetRouteAuthor](#)

[dbGetRouteSubAuthor](#)

[dbSetRouteSubAuthor](#)

[xdvReplaceInstWithVias](#)

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

<i>d_cellViewId</i>	Database ID of the cellview.
<i>l_startPoint</i>	List which specifies the starting point of the guide.
<i>l_endPoint</i>	List which specifies the ending point of the guide.
<i>t_startLayerName</i>	Optional argument. The names of the layers in which the guide starts.
<i>t_endLayerName</i>	Optional argument. The names of the layers in which the guide ends.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_guideId</i>	When successful, returns the database ID of the created guide.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

Interconnect Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layer or layer-purpose pair. For more information, see Database Access Function Argument Types .
<i>l_beginPt</i>	The start point of the path segment.
<i>l_endPt</i>	The end point of the path segment.
<i>n_width</i>	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 <i>n_width</i>. 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.
<i>t_beginStyle</i>	The beginning and ending style of the path segment.
<i>t_endStyle</i>	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 (<i>endExt</i>) distance past the end point. You need to specify the value of <i>n_beginExt</i> and/or <i>n_endExt</i>. ■ 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 <i>n_beginExt</i>, <i>n_beginLeftDiagExt</i>, <i>n_beginRightDiagExt</i>, <i>n_beginRightHalfWidth</i>, and/or <i>n_endExt</i>, <i>n_endLeftDiagExt</i>, <i>n_endRightDiagExt</i>, <i>n_endRightHalfWidth</i> in the argument <i>l_extValue</i>.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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_beginExt` is valid only when `t_beginStyle` is set to `custom` or `variable`.

`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`.

`n_endExt` is valid only when `t_endStyle` is set to `custom` or `variable`.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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)
```

This value is useful only when at least one of the *t_beginStyle* or *t_endStyle* is set to *custom*. If the *l_extValue* argument is specified and *t_beginStyle* is not *custom*, the first three values are ignored, and if *t_endStyle* is not *custom*, the last three values are ignored.

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*.
- *n_beginRightDiagExt*: right 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, $n\_beginRightDiagExt * \sqrt{2}$) to create a pathSeg with the desired beginning right diagonal extension *n_beginRightDiagExt*.

- *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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_pathSegId</i>	When successful, returns the database ID of the created path segment.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Examples

```
; Two ways of creating the same pathSeg with zero extension on both ends
dbCreatePathSeg(cv "Metall" 0:0 1:0 0.1 "truncate" "truncate")
dbCreatePathSeg(cv "Metall" 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 "Metall" 0:0 1:0 0.1 "truncate" "extend")
dbCreatePathSeg(cv "Metall" 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 "Metall" 0:0 1:0 0.1 "extend" "truncate")
dbCreatePathSeg(cv "Metall" 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 "Metall" 0:0 1:0 0.1 "chamfer" "truncate")
dbCreatePathSeg(cv "Metall" 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 "Metall" 0:0 1:0 0.1 "truncate" "chamfer")
dbCreatePathSeg(cv "Metall" 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 "Metall" 0:0 2:2 diagWidth "variable" "variable")
dbCreatePathSeg(cv "Metall" 0:0 2:2 diagWidth "extend" "extend")
```


Virtuoso Studio Design Environment SKILL Reference

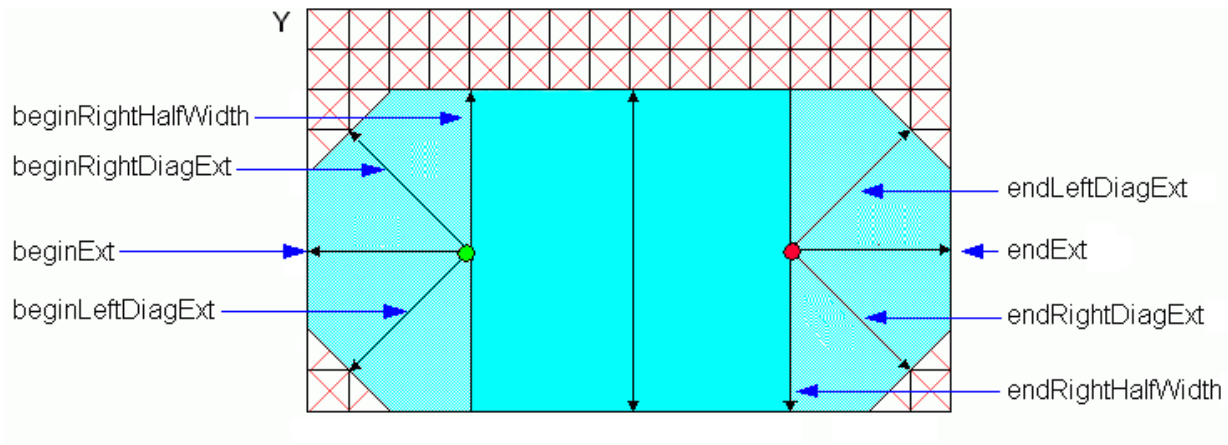
Database Access Functions

```
; Example for creating a diagonal pathSeg with custom ends
halfDiag=diagWidth/2
halfWidth=width/2
dbCreatePathSeg(cv "Metall1" 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
dbCreatePathSeg(cv "Metall1" 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.

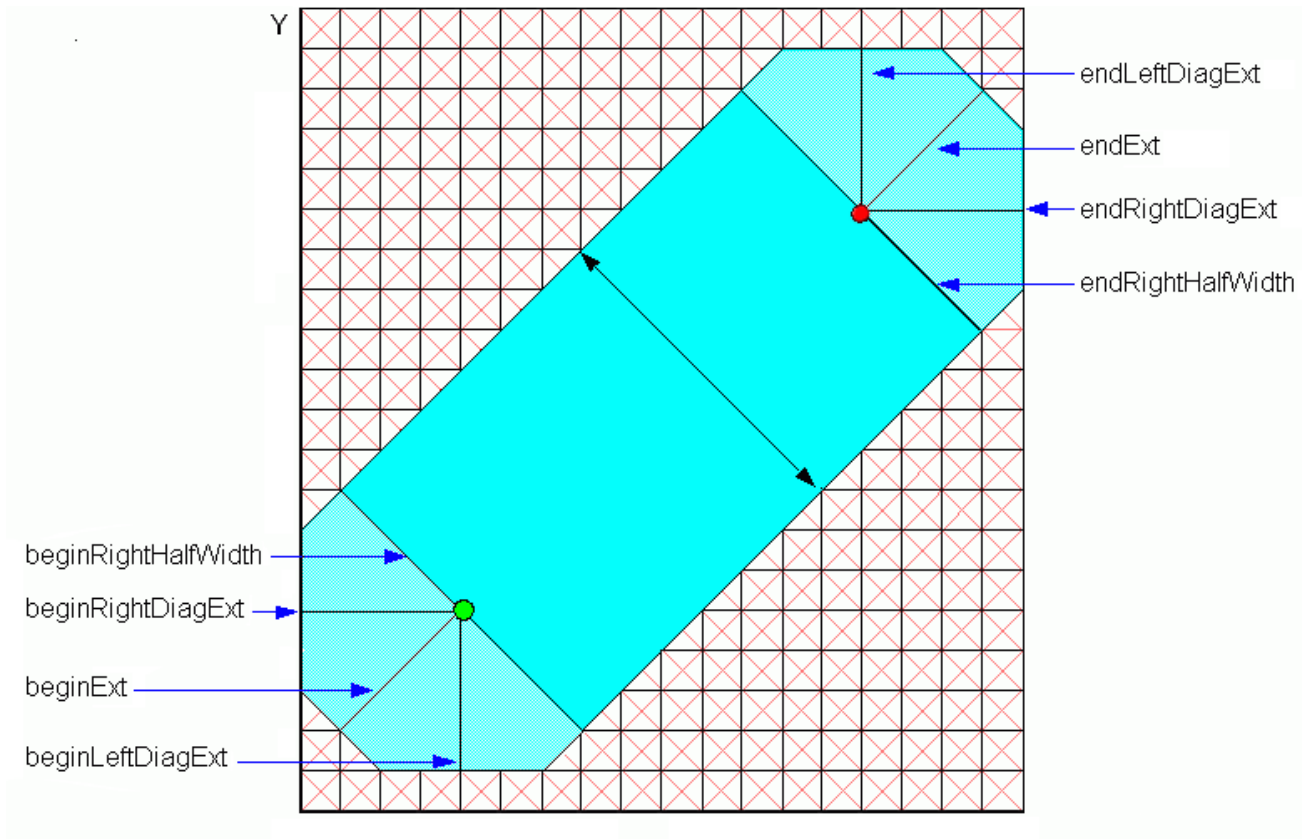
An orthogonal PathSeg (in which the left and right diagonals of the chamfer are obtained by multiplying the extension by 3/2):



Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

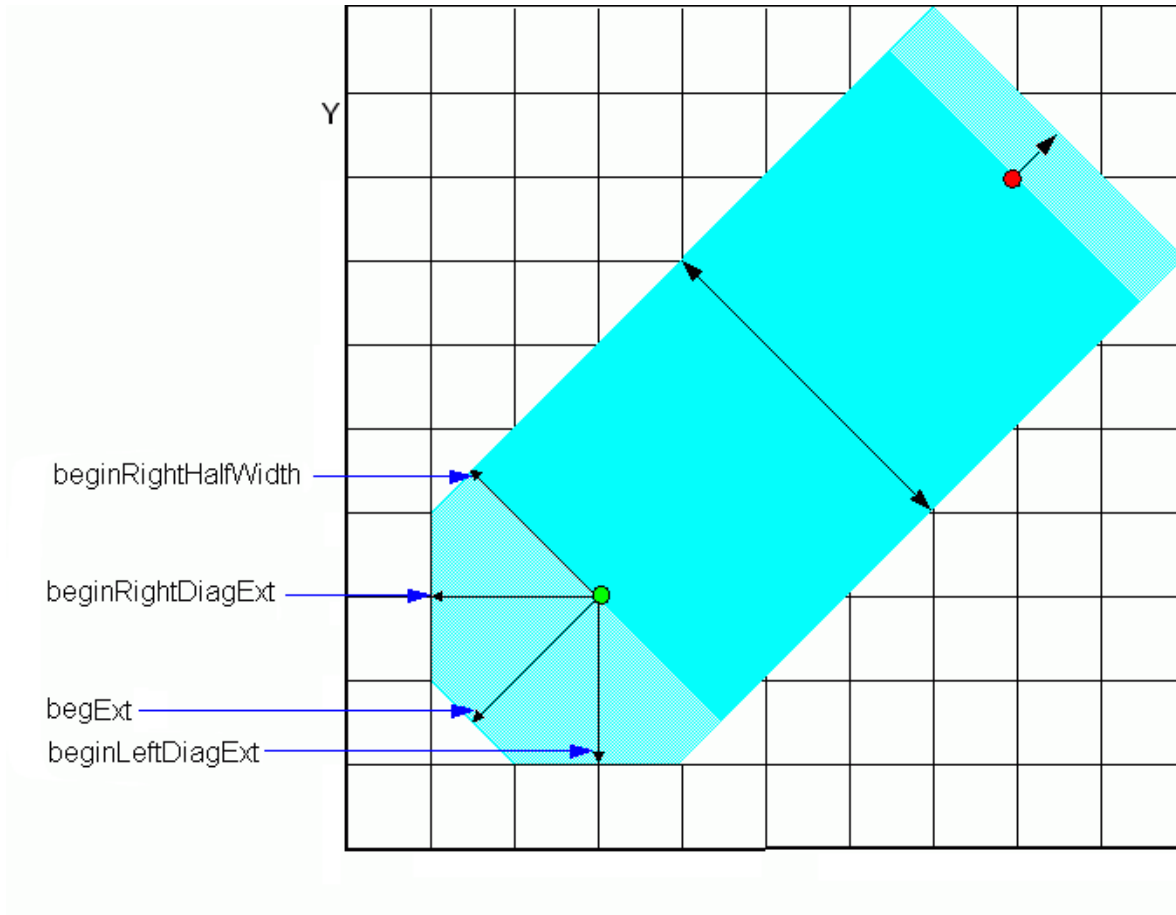
A diagonal PathSeg (in which the left and right diagonals of the chamfer are obtained by multiplying the extension by 2/3):



Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

A PathSeg with two different end styles:



Related Topics

[Database Access Function Argument Types](#)

[Interconnect Functions](#)

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.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview.
<i>d_netId</i>	Optional argument. Specifies the net in which to assign the route. Default is <code>nil</code> .
<i>g_isGlobal</i>	Optional argument. Specifies whether the net is <code>isGlobal</code> . Default is <code>nil</code> .
<i>t_routeStatus</i>	Optional argument. Specifies the placement status of the route. Valid values: <code>normal</code> , <code>fixed</code> , and <code>locked</code>

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<code>d_routeId</code>	When successful, returns the database ID of the created route.
<code>nil</code>	<code>nil</code> is returned if the cellview ID is invalid.

Examples

```
dbCreateRoute(cv netId t "normal")
```

Related Topics

[Routing Your Design](#)

[Interconnect Functions](#)

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.

This function cannot create a custom or standard via using parameters that will create inverted layer and implant enclosure shapes.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview.
<i>d_viaDefId</i>	Specifies the database ID of the viaDef associated with the via.
<i>l_point</i>	Specifies a list of points for the via.
<i>t_orient</i>	Specifies the orientation of the via.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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_implEnc, l_impl2Enc.
```

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.

Value Returned

<code>d_viaId</code>	When successful, returns the database ID of the via.
<code>nil</code>	<code>nil</code> is returned if the cellview ID is invalid.

Examples

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)
```

Related Topics

[Working with Vias](#)

[Database Access Function Argument Types](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Standard Via Attributes

Interconnect Functions

dbCreateCustomViaVariant

```
dbCreateCustomViaVariant(  
    d_cellViewId  
    t_viaVariantName  
    t_viaDefName  
    l_viaParams  
)  
=> d_viaVariantId / nil
```

Description

Creates a custom viaVariant in cellview.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.
<i>t_viaVariantName</i>	Specifies the name of the viaVariant.
<i>t_viaDefName</i>	Specifies the name of the viaDef associated with the viaVariant.
<i>l_viaParams</i>	The values and types of the parameters of the viaVariant, specified using a parameter list. <i>l_viaParams</i> , where <i>l_viaParams</i> is <code>list(t_propName t_propType g_value)</code>

Value Returned

<i>d_viaVariantId</i>	When successful, returns the database ID of the viaVariant.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Examples

```
dbCreateCustomViaVariant(cv "via1" list(list(p1 "float" 0.1)))
```

Related Topics

Working with Vias

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Custom viaVariant Attributes

Interconnect Functions

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.

This function cannot create standard via variant using parameters that will create inverted layer and implant enclosure shapes.

Arguments

<i>d_cellViewId</i>	Database ID of the cellview.
<i>t_viaVariantName</i>	Name of the standard viaVariant.
<i>t_viaDefName</i>	Name of the viaDef associated with the viaVariant.
<i>t_cutLayerName</i>	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
<i>t_cutLayerWidth</i>	Width of a cut, in user units. Valid values: Any positive number
<i>t_cutLayerHeight</i>	Height of a cut, in user units. Valid values: Any positive number
<i>n_cutRows</i>	Number of rows in the cut array. Valid values: Any positive integer

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

<i>n_cutCol</i>	Number of columns in the cut array. Valid values: Any positive integer
<i>f_cutSpaceX</i>	The horizontal edge-to-edge distance between cuts, in user units. Valid values: Any positive floating-point number
<i>f_cutSpaceY</i>	The vertical edge-to-edge distance between cuts, in user units. Valid values: Any positive floating-point number
<i>f_layer1EncX</i>	The horizontal distance from the cut edge to the edge of layer 1, in user units. Valid values: Any floating-point number
<i>f_layer1EncY</i>	The vertical distance, from the cut edge to the edge of layer 1, in user units. Valid values: Any floating-point number
<i>f_layer2EncX</i>	The horizontal distance from the cut edge to the edge of layer 2, in user units. Valid values: Any floating-point number
<i>f_layer2EncY</i>	The vertical distance from the cut edge to the edge of layer 2, in user units. Valid values: Any floating-point number
<i>f_layer1OffsetX</i>	The horizontal offset for layer 1 in user units. Valid values: Any floating-point number
<i>f_layer1OffsetY</i>	The vertical offset for layer 1 in user units. Valid values: Any floating-point number
<i>f_layer2OffsetX</i>	The horizontal offset for layer 2 in user units. Valid values: Any floating-point number
<i>f_layer2OffsetY</i>	The vertical offset for layer 2 in user units. Valid values: Any floating-point number
<i>f_origOffsetX</i>	The horizontal offset for the origin of the via in user units. Valid values: Any floating-point number

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

<i>f_origOffsetY</i>	The vertical offset for the origin of the via in user units. Valid values: Any floating-point number
<i>f_implant1EncX</i>	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
<i>f_implant1EncY</i>	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
<i>f_implant2EncX</i>	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
<i>f_implant2EncY</i>	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
<i>l_cutPattern</i>	Cut pattern for the <code>viaVariant</code> . 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 <code>viaVariant</code> . Whereas, 1 represents cut for the correspondent cut in the <code>viaVariant</code> .

Value Returned

<i>d_stdViaVariantId</i>	Database ID of the <code>viaVariant</code> that was created.
<i>nil</i>	The <code>stdViaVariant</code> could not be created.

Related Topics

[Working with Vias](#)

[Standard viaVariant Attributes](#)

[Interconnect Functions](#)

dbFindViaVariantByName

```
dbFindViaVariantByName(  
    d_cellViewId  
    t_viaVariantName  
)  
=> d_viaVariantId / nil
```

Description

Finds the viaVariant in a cellview.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.
<i>t_viaVariantName</i>	Specifies the name of the viaVariant.

Value Returned

<i>d_viaVariantId</i>	Database ID of the viaVariant that was created.
<i>nil</i>	The specified cellview is a regular cellview or the cellview ID is invalid.

Examples

```
dbFindViaVariantByName(cv "via1")
```

Related Topics

[Working with Vias](#)

[Interconnect Functions](#)

dbGetViaLayer

```
dbGetViaLayer(  
    d_viaId  
    t_viaLayerType  
)  
=> t_layerName / nil
```

Description

Returns the layer name of the specified layer of the given via.

Arguments

<i>d_viaId</i>	Specifies the via ID.
<i>t_viaLayerType</i>	Specifies the via layer type. Valid values: layer1, layer2, cutLayer

Value Returned

<i>t_layerName</i>	Returns the layer name for the specified layer of the via.
<i>nil</i>	Returns <i>nil</i> if an error occurs (for example, invalid via ID or layer type).

Examples

```
cv = dbOpenCellViewByType( "tech" "design" "layout" "maskLayout" "w" )  
viaDef_M2_M1 = techFindViaDefByName( tf "M2_M1" )  
vialt_m2_m1 = dbCreateVia( cv viaDef_M2_M1 10:10 "R0" )  
dbGetViaLayer( vialt_m2_m1 "layer1" )  
=> "metal2"  
dbGetViaLayer( vialt_m2_m1 "layer2" )  
=> "metal1"  
dbGetViaLayer( vialt_m2_m1 "cutLayer" )  
=> "via"
```

Related Topics

[Working with Vias](#)

[Interconnect Functions](#)

dbIsCellViewStdViaMaster

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

Description

Returns a boolean value indicating whether the specified cellview is a standard via master.

Arguments

<i>d_cellViewId</i>	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.

Related Topics

[Working with Vias](#)

[Standard Via Attributes](#)

[Interconnect Functions](#)

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

<i>d_pathSegId</i>	The database ID of the pathSeg object.
<i>l_beginPt</i>	The beginning point of the segment.
<i>l_endPt</i>	The ending point of the segment.

Value Returned

<i>t</i>	When successful, returns <i>t</i> .
<i>nil</i>	Returns <i>nil</i> , if the updates are not successful.

Related Topics

[Path Attributes](#)

[Interconnect Functions](#)

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

<i>d_viaId</i>	The database ID of the via.
<i>t_paramName</i>	The name of a via parameter for the specified via.

Value Returned

<i>d_viaId</i>	Database ID of the via.
<i>nil</i>	The cellview ID is invalid.

Examples

The via parameter "cutWidth" is set to use the default value for the specified standard via.

```
dbSetViaParamDefault(stdViaId "cutWidth")
```

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*.

Related Topics

[Working with Vias](#)

[Interconnect Functions](#)

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

<code>d_customViaId</code>	The database ID of the custom <code>cdsVia</code> for which the layer purpose parameters are to be set.
<code>tx_purpose</code>	The new shape purpose. The purpose can be specified either by the purpose name or the purpose number.

Value Returned

<code>t</code>	The layer purpose parameter values for the <code>cdsVia</code> were successfully set.
<code>nil</code>	<code>nil</code> is returned if the layer purpose parameters were not set.

Examples

```
dbSetViaShapePurpose(viaId "power")  
dbSetViaShapePurpose(viaId 256)
```

To reset layer purposes back to their original values, as specified in the `cdsViaDevice` definition, use:

```
dbSetViaShapePurpose(viaId " ")
```

The via parameter `"cutWidth"` is set to use the default value for the specified standard via.

dbGetRouteAuthor

```
dbGetRouteAuthor(  
    d_routeId  
    [ g_getSubAuthor ]  
)  
=> t_authorType / l_authorAndSubAuthorType
```

Description

Returns the author and, optionally, the sub-author, type of the specified route.

Arguments

<i>d_routeId</i>	Database ID of the route.
<i>g_getSubAuthor</i>	Boolean specifying whether the sub-author type of the route must also be retrieved.

Value Returned

<i>t_authorType</i>	Author type of the specified route.
<i>l_authorAndSubAuthorType</i>	List of author types and sub-author types.

Examples

Creates a new route.

```
rt1 = dbCreateRoute(MainCV mainNet t "normal")  
db:0x1f7b9c1a
```

Retrieves the author type of the route `rt1`.

```
dbGetRouteAuthor(rt1)  
=> "Manual"
```

Retrieves the author type of the route `rt1`.

```
dbGetRouteAuthor(rt1 nil)  
=> "Manual"
```

Retrieves the author and sub-author type of the route `rt1`.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

```
dbGetRouteAuthor(rt1 t)  
=> ("Manual" "None")
```

Related Topics

[dbEnableRouteObservers](#)

[Interconnect Functions](#)

dbSetRouteAuthor

```
dbSetRouteAuthor(  
    d_routeId  
    t_authorType  
    [ t_subAuthorType ]  
)  
=> t / nil
```

Description

Sets the author and, optionally, the sub-author type for the specified route.

Arguments

<i>d_routeId</i>	Database ID of a route.
<i>t_authorType</i>	Author type to be set for the route. Valid values: Auto, Manual Any other value will result in an error.
<i>t_subAuthorType</i>	Sub-author type to be set for the route. Valid values: None, Mesh, Spine, Shield, Tie Shield Any other value will result in an error.

Value Returned

<i>t</i>	Specified author type and sub-author type, are set for the given route.
<i>nil</i>	Specified author type is not set for the given route.

Examples

Create a new route.

```
rtl = dbCreateRoute(MainCV mainNet t "normal")  
=> db:0x1f7b9c1a
```

Set author type to Auto.

```
dbSetRouteAuthor(rtl "Auto")
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

```
=> t
```

Check the author type.

```
dbGetRouteAuthor(rt1)
```

```
=> "Auto"
```

Create a new route.

```
rt1 = dbCreateRoute(MainCV mainNet t "normal")
```

Set author type to `Manual` and sub-author type to `None`.

```
dbSetRouteAuthor(rt1 "Manual" "None")
```

```
=> t
```

Related Topics

[Interconnect Functions](#)

dbGetRouteSubAuthor

```
dbGetRouteSubAuthor(  
    d_routeId  
)  
=> t_subAuthorType
```

Description

Retrieves the sub-author attribute on the specified route.

Argument

<i>d_routeId</i>	Database ID of a route.

Value Returned

<i>t_subAuthorType</i>	Name of the sub-author type.

Examples

Create a new route `rt1` with sub-author type Mesh.

```
rt1 = dbCreateRoute(MainCV mainNet t "normal")  
dbSetRouteSubAuthor(rt1 "Mesh")  
=> t
```

Retrieve the sub-author type on the route `rt1`.

```
dbGetRouteSubAuthor(rt1)  
=> "Mesh"
```

Related Topics

[Interconnect Functions](#)

dbSetRouteSubAuthor

```
dbSetRouteSubAuthor(  
    d_routeId  
    t_subAuthorType  
)  
=> t / nil
```

Description

Sets the sub-author attribute on the specified route.

Arguments

<i>d_routeId</i>	Database ID of a route.
<i>t_subAuthorType</i>	Name of the sub-author type to be set. Valid values: None, Mesh, Spine, Shield, Tie Shield Any other value will result in an error.

Value Returned

<i>t</i>	The specified sub-author attribute was successfully set.
<i>nil</i>	The sub-author attribute could not be set.

Examples

Create a new route *rtl* and set sub-author type as *Mesh*.

```
rtl = dbCreateRoute(MainCV mainNet t "normal")  
dbSetRouteSubAuthor(rtl "Mesh")  
=> t
```

Retrieve the sub-author type.

```
dbGetRouteSubAuthor(rtl)  
=> "Mesh"
```

Related Topics

Interconnect Functions

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

<i>t_libName</i>	Specifies the library name.
<i>t_cellName</i>	Specifies the cell name.
<i>t_viewName</i>	Specifies the view name.

Value Returned

<i>t</i>	Via-like instances are detected and replaced with standard vias. This is possible only when correct library, cell, and view have been specified.
<i>nil</i>	Library, cell, and view specified do not exist, or there is an error.

Examples

```
xdvReplaceInstWithVias("designLib" "Chip_Top" "layout")
```

Detects and replaces via-like instances with standard vias across the hierarchy of the cellview designLib/Chip_Top/layout.

Related Topics

[Interconnect Functions](#)

Trim Functions

This topic provides a list of Cadence® SKILL functions associated with trims.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbGetAllTrimFillShapes](#)

[dbGetShapeTrimFillType](#)

dbGetAllTrimFillShapes

```
dbGetAllTrimFillShapes(  
    d_cellviewId  
)  
=> l_shapeID / nil
```

Description

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

<i>d_cellviewId</i>	Database ID of a cellview.

Value Returned

<i>l_shapeID</i>	List of database IDs of the shapes with a valid trim fill type. Valid trim fill types: <code>bridgeMetal</code> , <code>SADPFillPatch</code> , <code>gapFill</code>
<i>nil</i>	No shapes with valid trim fill type were found.

Examples

Returns the database IDs of the shapes with the trim fill type set as `bridgeMetal` or `SADPFillPatch` in the cellview `cv`.

```
cv = dbOpenCellViewByType("testLib" "cellview" "layout" "maskLayout" "w")  
dbGetAllTrimFillShapes(cv)  
=> (db:0x1d15cf1b db:0x1d15cf1a)
```

Related Topics

Trim Functions

dbGetShapeTrimFillType

```
dbGetShapeTrimFillType (
    d_shapeId
)
=> t_trimFillType / nil
```

Description

Returns the trim fill type for the specified trim shape.

Arguments

<i>d_shapeId</i>	Database ID of a shape.

Value Returned

<i>t_trimFillType</i>	Fill type for the trim shape. Valid values: <code>notTrim</code> , <code>bridgeMetal</code> , <code>SADPFillPatch</code> , <code>gapFill</code>
<i>nil</i>	Returned in case of failure.

Examples

Returns that the fill type for the trim shape `s1` is `notTrim`.

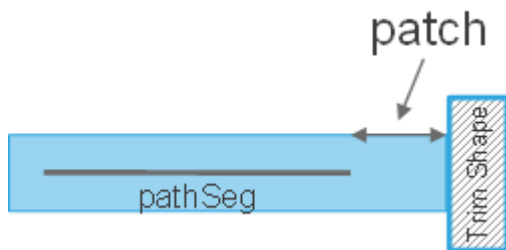
```
s1 = dbCreateRect(cv "M1" list(0:0 1:1))
dbGetShapeTrimFillType(s1)
=> "notTrim"
```

Related Topics

[Trim Functions](#)

Patch 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`.

This topic provides a list of Cadence® SKILL functions associated with patches.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

Supporting Patch Metal Shapes

[dbGetPatchType](#)

[dbSetBeginPatch](#)

[dbSetEndPatch](#)

[dbSetFullPatch](#)

[dbSetPatchType](#)

dbGetPatchType

```
dbGetPatchType(  
    d_pathSegId  
)  
=> t_patchType / nil
```

Description

Returns the patch type for the specified path segment.

Arguments

<i>d_pathSegId</i>	Specifies the path segment ID.

Value Returned

<i>t_patchType</i> .	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).

Examples

```
ps = dbCreatePathSeg(cv "metal1" list(-1 0) list(0 1) .05 "truncate" "truncate")  
t db:0x1f7b9c1a  
dbGetPatchType(ps)  
"notPatch"
```

Related Topics

[Patch Functions](#)

dbSetBeginPatch

```
dbSetBeginPatch(  
    d_pathSegId  
    g_value  
)  
=> t / nil
```

Description

Adds or removes the `beginStyle` patch type for the specified path segment depending on the argument `g_value`.

Arguments

<i>d_pathSegId</i>	The database ID of the path segment.
<i>g_value</i>	Value of the patch type. Set to <code>true</code> to set the <code>beginStyle</code> for the path segment. Else, set to <code>false</code> to remove the patch type..

Value Returned

<i>t</i>	The patch type for the path segment is successfully set to <code>beginStyle</code> .
<i>nil</i>	The patch type is not successfully set.

Examples

```
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
```

Related Topics

Patch Functions

dbSetEndPatch

```
dbSetEndPatch(  
    d_pathSegId  
    g_value  
)  
=> t / nil
```

Description

Adds or removes the `endStyle` patch type for the specified path segment depending on the argument `g_value`.

Arguments

<i>d_pathSegId</i>	The database ID of the path segment.
<i>g_value</i>	Value of the patch type. Set to <code>true</code> to set the <code>endStyle</code> for the path segment. Else, set to <code>false</code> to unset it.

Value Returned

<i>t</i>	The patch type for the path segment was successfully set to <code>endStyle</code> .
<i>nil</i>	<code>nil</code> is returned if the patch type was not successfully set.

Examples

```
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
```

Related Topics

Patch Functions

dbSetFullPatch

```
dbSetFullPatch(  
    d_pathSegId  
    g_value  
)  
=> t / nil
```

Description

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

<i>d_pathSegId</i>	The database ID of the path segment.
<i>g_value</i>	Value of the patch type. Set to <code>true</code> to set the full style for the path segment. Else, set to <code>false</code> to unset it.

Value Returned

<code>t</code>	The patch type for the path segment was successfully set to full style.
<code>nil</code>	<code>nil</code> is returned if the patch type was not successfully set.

Examples

```
ps = dbCreatePathSeg(cv "metall" 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
```

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Related Topics

Patch Functions

dbSetPatchType

```
dbSetPatchType(  
    d_pathSegId  
    t_patchType  
)  
=> t / nil
```

Description

Sets the patch type for the specified path segment.

Arguments

<i>d_pathSegId</i>	The database ID of the path segment.
<i>t_patchTypeurpose</i>	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

<i>t</i>	The patch type for the path segment was successfully set.
<i>nil</i>	<i>nil</i> is returned if the patch type was not set.

Examples

```
ps = dbCreatePathSeg(cv "metal1" list(-1 0) list(0 1) .05 "truncate" "truncate")  
db:0x1f7b9c1a  
dbSetPatchType(ps "endStyle")  
t
```

Related Topics

Patch Functions

Marker 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.

This topic provides a list of Cadence® SKILL functions associated with markers.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateMarker](#)

[dbAddObjectsToMarker](#)

[dbRemoveObjectsFromMarker](#)

[dbGetMarkerName](#)

[dbSetMarkerName](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_msg</i>	Message associated with the marker.
<i>t_tool</i>	Name value of the application associated with this marker.
<i>l_points</i>	Specifies the list of points for the marker.
<i>ld_objects</i>	Optional argument that specifies the list of database ID's of the objects associated with the marker. Default Value: nil
<i>g_isVisible</i>	Optional boolean argument that indicates if the marker is visible Valid values: t, nil Default Value: t

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

<i>g_isClosed</i>	Optional argument that specifies if the pointArray of the marker is closed. Valid values: t, nil Default Value: t
<i>t_severity</i>	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. Default Value: annotation
<i>t_shortMsg</i>	Optional argument that specifies the shortMsg associated with the marker. Default Value: " "
<i>t_deleteWhen</i>	Optional argument that specifies when markers are automatically deleted. Valid values: deleteNever, deleteFirst, deleteLast, deleteModify. Default Value: "deleteNever"

Value Returned

<i>d_markerId</i>	The database ID of the marker if the creation is successful.
nil	nil is returned if the cellview ID is invalid.

Related Topics

[Database Access Function Argument Types](#)

[Marker Attributes](#)

[Marker Functions](#)

dbAddObjectsToMarker

```
dbAddObjectsToMarker(  
    d_markerId  
    ld_objectId  
)  
=> t / nil
```

Description

Adds a list of objects to a specified marker.

Arguments

<i>d_markerId</i>	The database ID of the marker.
<i>ld_objectId</i>	The database ID's of the list of objects to be added to the marker.

Value Returned

<i>t</i>	The list is added successfully.
<i>nil</i>	<i>nil</i> is returned if the marker ID or object IDs are invalid.

Related Topics

[Marker Functions](#)

dbRemoveObjectsFromMarker

```
dbRemoveObjectsFromMarker(  
    d_markerId  
    ld_objectId  
)  
=> t / nil
```

Description

Removes a list of objects from a specified marker.

Arguments

<i>d_markerId</i>	The database ID of the marker.
<i>ld_objectId</i>	The database ID's of the list of objects to be removed from the specified marker.

Value Returned

<i>t</i>	The list is removed successfully.
<i>nil</i>	<i>nil</i> is returned if the marker ID or object IDs are invalid.

Related Topics

[Marker Functions](#)

dbGetMarkerName

```
dbGetMarkerName (
    d_markerId
)
=> t_markerName / nil
```

Description

Retrieves the name assigned to the marker associated with the given marker ID.

Arguments

<i>d_markerId</i>	Database ID of a marker.

Value Returned

<i>t_markerName</i>	Name assigned to the specified marker. Returns an empty string for unnamed markers.
<i>nil</i>	Marker name was not retrieved.

Examples

```
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.

Related Topics

Database Access Function Argument Types

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Marker Functions

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. The name can be an empty string. By default, the markers are first sorted by names and then by their box.

Arguments

<i>d_markerId</i>	Database ID associated with the marker to which you want to assign a name.
<i>t_markerName</i>	Name to be assigned to the marker ID.

Value Returned

t	Marker name was successfully assigned.
nil	Marker name was not assigned.

Examples

Creates a new marker with the specified list of points. You can assign a specific name to this marker.

```
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
```

Related Topics

Marker Functions

Query Functions

Query of objects is based on the open hierarchy level. Only cellviews that are currently open are searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

This topic provides a list of Cadence® SKILL functions associated with queries.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbBoundaryQuery](#)

[dbFigGroupQuery](#)

[dbGuideQuery](#)

[dbInstQuery](#)

[dbLayerBlockageQuery](#)

[dbMarkerQuery](#)

[dbPlacementBlockageQuery](#)

[dbRowQuery](#)

[dbPlacementBlockageQuery](#)

[dbRowQuery](#)

[dbShapeQuery](#)

[dbSteinerQuery](#)

[dbViaQuery](#)

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_boundaryId</i>	Returns a list identifying all the boundaries produced during the query.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

Query Functions

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*.

Arguments

<i>d_cellView</i>	The database ID of the cellview in which the design hierarchy will be traversed by the query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	Optional argument. The level of the design hierarchy in which to start the querying process. If the <i>startLevel</i> argument is not specified, the start level defaults to 0.
<i>x_stopLevel</i>	Optional argument. The level of the design hierarchy in which to stop the querying process. If the <i>stopLevel</i> argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	Optional argument. Specifies the figGroup filter size. The query produces only figGroups with a size larger than the <i>filterSize</i> . If no <i>filterSize</i> argument is specified, the blockage filter size defaults to 0.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

Related Topics

Query Functions

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_boundaryId</i>	Returns a list identifying all the guides produced during the query.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

Query Functions

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_instanceId</i>	Returns a list identifying all the instances produced during the query.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

Query Functions

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>tx_layer</i>	Specifies the layer name or number of the blockage.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	Optional argument. The level of the design hierarchy in which to start the querying process. If the <i>startLevel</i> argument is not specified, the start level defaults to 0.
<i>x_stopLevel</i>	Optional argument. The level of the design hierarchy in which to stop the querying process. If the <i>stopLevel</i> argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	Optional argument. Specifies the blockage filter size. The query produces only blockages with a size larger than the <i>filterSize</i> . If no <i>filterSize</i> argument is specified, the blockage filter size defaults to 0.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

Related Topics

[Query Functions](#)

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_markerId</i>	Returns a list identifying all the markers produced during the query.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

Query Functions

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

Related Topics

[Query Functions](#)

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_rowId</i>	Returns a list identifying all the rows produced during the query.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Related Topics

Query Functions

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_layerPurposePair</i>	Layer purpose pair. All shapes on this layer purpose pair will be queried and returned.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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_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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Examples

Returns a list containing shapes or lists representing occurrence shapes that are on M1 drawing, overlapping the (0:0 0:0) point.

```
shapes = dbShapeQuery( deGetCellView() list("M1" "drawing") list(0:0 0:0) )
```

Additional Information

The SKILL functions `dbOpenHier`, `dbShapeQuery`, and `dbGetOverlaps` perform different functions and have different implementations that may traverse the hierarchy in different ways. Although it is possible to use combinations of these and other functions to get similar results, code that is sensitive to the order of traversal may produce different results. For example, code that is traversing the hierarchy through `dbOpenHier` and `dbShapeQuery`, which needs to expand through Pcells, may produce a result that is different from `dbGetOverlaps`.

if there is a Pcell evaluation failure at some point of the traversal because these functions cannot guarantee order and it cannot be predicted whether or when a Pcell will fail to evaluate, some data that is produced by one set of SKILL functions may not be produced by a different set of SKILL functions.

Related Topics

[Query Functions](#)

[dbOpenHier](#)

[dbGetOverlaps](#)

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`.

Arguments

<code>d_cellviewId</code>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<code>tx_layer</code>	Specifies the layer name or number of the blockage.
<code>l_bBox</code>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<code>x_startLevel</code>	Optional argument. The level of the design hierarchy in which to start the querying process. If the <code>startLevel</code> argument is not specified, the start level defaults to 0.
<code>x_stopLevel</code>	Optional argument. The level of the design hierarchy in which to stop the querying process. If the <code>stopLevel</code> argument is not specified, the stop level defaults to maximum hierarchy depth. The query will stop prior to <code>x_stopLevel</code> if it encounters a cellview which is not currently loaded.
<code>n_filterSize</code>	Optional argument. Specifies the steiner filter size. The query produces only steiners with a size larger than the <code>filterSize</code> . If no <code>filterSize</code> argument is specified, the blockage filter size defaults to 0.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

Related Topics

[Query Functions](#)

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*.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which the design hierarchy will be traversed by the region query.
<i>l_bBox</i>	The bounding box that specifies the region in context of the top cellview to be used as the basis of the query.
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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.

Related Topics

Query Functions

Row 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).

To allow placement of different types of elements, different types of rows need to be created. Rows can be created overlapping each other.

This topic provides a list of Cadence® SKILL functions associated with rows.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateRow](#)

[dbFindRowByName](#)

[dbFindRowHeader](#)

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

<i>d_cellViewId</i>	The database ID of the cellview.
<i>d_siteDef</i>	The <code>siteDef</code> to be used for this row.
<i>t_name</i>	Specifies the name of the row.
<i>l_point</i>	The origin point for the row.
<i>x_numSites</i>	The number of site placements in the row.
<i>t_siteOrient</i>	Specifies the orientation of the site placements of the row. Default value: R0
<i>t_rowOrient</i>	Specifies the orientation of the row. Default value: R0

Value Returned

<i>d_rowId</i>	The database ID of the row if creation is successful.
<code>nil</code>	<code>nil</code> is returned if the cellview ID is invalid.

Related Topics

[Database Access Function Argument Types](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Row Functions

dbFindRowByName

```
dbFindRowByName (
    d_cellViewId
    t_name
)
=> d_rowId / nil
```

Description

Finds the specified row.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview.
<i>t_name</i>	Specifies the name of the row.

Value Returned

<i>d_rowId</i>	The database ID of the row when successful.
<i>nil</i>	The specified row does not exist in the given cellview.

Related Topics

[Row Functions](#)

dbFindRowHeader

```
dbFindRowHeader(  
    d_cellViewId  
    t_siteDefName  
)  
=> d_rowHeaderId / nil
```

Description

Finds the specified row header. A row header contains information about a particular `siteDef` including the rows using the `siteDef`.

Arguments

<i>d_cellViewId</i>	The database ID of the cellview in which to search for the row header.
<i>t_siteDefName</i>	Name of the <code>siteDef</code> associated with the row header.

Value Returned

<i>d_rowHeaderId</i>	The database ID of the row header when successful.
<code>nil</code>	The specified row header does not exist in the given cellview.

Related Topics

[Row Functions](#)

Steiner Functions

This topic provides a list of Cadence® SKILL functions associated with Steiner.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateSteiner](#)

[dbCreateTapSteiner](#)

[dbIsTapSteiner](#)

dbCreateSteiner

```
dbCreateSteiner(  
    d_cellViewId  
    l_bBox  
    [ tx_layer ]  
)  
=> d_steinerId / nil
```

Description

Creates a Steiner object in the given cellview.

Arguments

<i>d_cellviewId</i>	Database ID of the cellview.
<i>l_bBox</i>	List of bBox for the Steiner to be created.
<i>tx_layer</i>	(Optional) Layer associated with the created tap steiner.

Value Returned

<i>d_steinerId</i>	ID of the created Steiner.
<i>nil</i>	Returned if the Steiner cannot be created.

Examples

```
steiner1 = dbCreateSteiner(cv list( 4:4 8:8))
```

Related Topics

[Steiner Functions](#)

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

<i>d_cellviewId</i>	Database ID of a cellview.
<i>l_bBox</i>	List of bBox for the Steiner to be created.
<i>tx_layer</i>	Layer associated with the created tap steiner.

Value Returned

<i>d_steinerId</i>	ID of the created tap steiner.
<i>nil</i>	Returned if the tap steiner cannot be created.

Examples

```
dbCreateTapSteiner(cv list(-5:-5 0:0) "metal3")
```

Related Topics

[Steiner Functions](#)

dbIsTapSteiner

```
dbIsTapSteiner(  
    d_object  
)  
=> t / nil
```

Description

Checks whether the specified ID is a tap steiner ID.

Arguments

<i>d_object</i>	ID of an object.

Value Returned

t	The object is a tap steiner object.
nil	The object is not a tap steiner object.

Examples

```
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
```

Related Topics

Steiner Functions

Supply and Ground Sensitivity Functions

This topic provides a list of Cadence® SKILL functions associated with supply and ground sensitivity.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbGetTermGroundSensitivity](#)

[dbGetTermSupplySensitivity](#)

[dbSetTermGroundSensitivity](#)

[dbSetTermSupplySensitivity](#)

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

<i>d_termId</i>	Database ID of a terminal.

Value Returned

<i>d_groundTermId</i>	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.
<i>nil</i>	<i>nil</i> is returned if the property <code>groundSensitivity</code> does not exist on the terminal.

Related Topics

[Supply and Ground Sensitivity Functions](#)

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

<i>d_termId</i>	Database ID of a terminal.

Value Returned

<i>d_supplyTermId</i>	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.
<i>nil</i>	<i>nil</i> is returned if the property <code>supplySensitivity</code> does not exist on the terminal.

Related Topics

[Supply and Ground Sensitivity Functions](#)

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

<i>d_termId</i>	The database ID of the terminal figure.
<i>d_groundTermId</i>	The database ID of the ground terminal.

Value Returned

<i>t</i>	<i>t</i> is returned when successful.
<i>nil</i>	<i>nil</i> is returned if either of the terminal IDs are invalid.

Related Topics

[Supply and Ground Sensitivity Functions](#)

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

<i>d_termId</i>	The database ID of the terminal figure.
<i>d_groundTermId</i>	The database ID of the supply terminal.

Value Returned

<i>t</i>	<i>t</i> is returned when successful.
<i>nil</i>	<i>nil</i> is returned if either of the terminal IDs are invalid.

Related Topics

[Supply and Ground Sensitivity Functions](#)

Terminal Ordering Functions

This topic provides a list of Cadence® SKILL functions associated with terminal ordering.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateInstTermByPosition](#)

[dbFindTermByPosition](#)

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

<i>d_netId</i>	The database ID of the net to be used. <i>nil</i> can be specified.
<i>d_instId</i>	The instance ID of the master in which the terminal is to be connected.
<i>x_position</i>	The position of a terminal in the instance master.

Value Returned

<i>d_instTermId</i>	Returns <i>t</i> if the instTerm is successfully create based on the position.
<i>nil</i>	<i>nil</i> is be returned if the instance is using connected by name/reference.

Examples

Creates an instTerm for terminal at 0 position for the instance i1.

```
dbCreateInstTermByPosition(net i1 0)
```

Related Topics

[Terminal Ordering Functions](#)

dbFindTermByPosition

```
dbFindTermByPosition(  
    d_cellviewId  
    x_position  
)  
=> d_termId / nil
```

Description

Finds the terminal assigned to specific position.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview in which to search for the terminal.
<i>x_position</i>	The position of the terminal.

Value Returned

<i>d_termId</i>	Returns the terminal ID for the specified position.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid.

Examples

```
dbFindTermByPosition(cv 0)
```

Related Topics

[Terminal Ordering Functions](#)

Terminal Antenna Functions

This topic provides a list of Cadence® SKILL functions associated with terminal antennas.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbDeleteAllTerminalAntennaRules](#)

[dbDeleteTerminalAntennaRuleOnLayer](#)

[dbGetTerminalAntennaRules](#)

[dbGetTerminalAntennaRuleOnLayer](#)

[dbGetTerminalAntennaRules](#)

[dbSetTerminalAntennaRuleOnLayer](#)

[dbGetTerminalAntennaRules](#)

[dbSetTerminalAntennaRuleOnLayer](#)

dbDeleteAllTerminalAntennaRules

```
dbDeleteAllTerminalAntennaRules(  
    d_termId  
    t_modelName  
)  
=> t / nil
```

Description

Deletes all antenna rules from the specified terminal.

Arguments

<i>d_termId</i>	The database ID of the terminal.
<i>t_modelName</i>	The name of the antenna model. Valid values: "defaultAntennaModel", "secondAntennaModel", "thirdAntennaModel", "fourthAntennaModel", "antennaModel5" to "antennaModel32"

Value Returned

<i>t</i>	The figure was added successfully.
<i>nil</i>	An error occurred during execution.

Example

```
dbDeleteAllTerminalAntennaRules(termA "defaultAntennaModel")
```

```
dbDeleteAllTerminalAntennaRules(termA "secondAntennaModel")
```

Related Topics

[Terminal Antenna Functions](#)

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

<i>d_termId</i>	The database ID of the terminal.
<i>t_modelName</i>	Specifies the model name. Valid values: "defaultAntennaModel", "secondAntennaModel", "thirdAntennaModel", "fourthAntennaModel", "antennaModel5" to "antennaModel32"
<i>t_dataName</i>	Specifies the data name. Valid values: "ANTENNAGATEAREA " "ANTENNADIFFAREA " "ANTENNAPARTIALMETALAREA " "ANTENNAPARTIALMETALSIDEAREA " "ANTENNAPARTIALCUTAREA " "ANTENNAMAXAREACAR " "ANTENNAMAXSIDEAREACAR " "ANTENNAMAXCUTCAR "
<i>tx_layer</i>	The layer name.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

t	The antenna rule was removed successfully.
nil	An error occurred during execution.

Example

```
dbDeleteTerminalAntennaRuleOnLayer(  
    termA "defaultAntennaModel" "ANTENNAGATEAREA" "M4")
```

Related Topics

[Terminal Antenna Functions](#)

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

<i>d_termId</i>	The database ID of the terminal.
<i>t_modelName</i>	Specifies the model name. Valid values: "defaultAntennaModel", "secondAntennaModel", "thirdAntennaModel", "fourthAntennaModel", "antennaModel15" to "antennaModel32"
<i>t_dataName</i>	Specifies the data name. Valid values: "ANTENNAGATEAREA" "ANTENNADIFFAREA" "ANTENNAPARTIALMETALAREA" "ANTENNAPARTIALMETALSIDEAREA" "ANTENNAPARTIALCUTAREA" "ANTENNAMAXAREACAR" "ANTENNAMAXSIDEAREACAR" "ANTENNAMAXCUTCAR"
<i>l_antennaValueLayerPairs</i>	A list of the antenna value layer pairs. list(list(g_value tx_layer)...)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

t	The antenna rule was removed successfully.
nil	An error occurred during execution.

Example

```
dbDeleteTerminalAntennaRules(termA "defaultAntennaModel" "ANTENNAGATEAREA"  
'( '(22.2 "M2"))')
```

Related Topics

[Terminal Antenna Functions](#)

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

<i>d_termId</i>	The database ID of the terminal.
<i>t_modelName</i>	Specifies the model name. Valid values: "defaultAntennaModel", "secondAntennaModel", "thirdAntennaModel", "fourthAntennaModel", "antennaModel5" to "antennaModel32"
<i>t_dataName</i>	Specifies the data name. Valid values: "ANTENNAGATEAREA " "ANTENNADIFFAREA " "ANTENNAPARTIALMETALAREA " "ANTENNAPARTIALMETALSIDEAREA " "ANTENNAPARTIALCUTAREA " "ANTENNAMAXAREACAR " "ANTENNAMAXSIDEAREACAR " "ANTENNAMAXCUTCAR "
<i>tx_layer</i>	The layer name.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>g_value</i>	The value of the antenna rule is returned.
<i>nil</i>	An error occurred during execution.

Example

```
dbGetTerminalAntennaRuleOnLayer(termA "defaultAntennaModel" "ANTENNAGATEAREA"  
"Metal1")
```

Related Topics

[Terminal Antenna Functions](#)

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

<i>d_termId</i>	The database ID of the terminal.
<i>t_modelName</i>	Specifies the model name. Valid values: "defaultAntennaModel", "secondAntennaModel", "thirdAntennaModel", "fourthAntennaModel", "antennaModel5" to "antennaModel32"
<i>t_dataName</i>	Specifies the data name. Valid values: "ANTENNAGATEAREA" "ANTENNADIFFAREA" "ANTENNAPARTIALMETALAREA" "ANTENNAPARTIALMETALSIDEAREA" "ANTENNAPARTIALCUTAREA" "ANTENNAMAXAREACAR" "ANTENNAMAXSIDEAREACAR" "ANTENNAMAXCUTCAR"

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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")
```

```
=> ((12.0 "M2") 50.0)
```

Related Topics

[Terminal Antenna Functions](#)

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

<i>d_termId</i>	The database ID of the terminal.
<i>t_modelName</i>	Specifies the model name. Valid values: "defaultAntennaModel", "secondAntennaModel", "thirdAntennaModel", "fourthAntennaModel", "antennaModel5" to "antennaModel32"
<i>t_dataName</i>	Specifies the data name. Valid values: "ANTENNAGATEAREA" "ANTENNADIFFAREA" "ANTENNAPARTIALMETALAREA" "ANTENNAPARTIALMETALSIDEAREA" "ANTENNAPARTIALCUTAREA" "ANTENNAMAXAREACAR" "ANTENNAMAXSIDEAREACAR" "ANTENNAMAXCUTCAR"
<i>g_value</i>	The value of the antenna rule.
<i>tx_layer</i>	The layer name.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

t	The antenna rule was set successfully.
nil	An error occurred during execution.

Example

```
dbSetTerminalAntennaRuleOnLayer(termA "defaultAntennaModel" "ANTENNAGATEAREA" 30  
"M3")
```

Related Topics

[Terminal Antenna Functions](#)

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

<i>d_termId</i>	The database ID of the terminal.
<i>t_modelName</i>	Specifies the model name. Valid values: "defaultAntennaModel", "secondAntennaModel", "thirdAntennaModel", "fourthAntennaModel", "antennaModel15" to "antennaModel32"
<i>t_dataName</i>	Specifies the data name. Valid values: "ANTENNAGATEAREA" "ANTENNADIFFAREA" "ANTENNAPARTIALMETALAREA" "ANTENNAPARTIALMETALSIDEAREA" "ANTENNAPARTIALCUTAREA" "ANTENNAMAXAREACAR" "ANTENNAMAXSIDEAREACAR" "ANTENNAMAXCUTCAR"
<i>l_antennaValueLayerPairs</i>	A list of the antenna value layer pairs. list(list(g_value tx_layer)...)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

t	The antenna rule was set successfully.
nil	An error occurred during execution.

Example

```
dbSetTerminalAntennaRules(termA "defaultAntennaModel" "ANTENNAGATEAREA"  
' (' (22.2 "M2") ) )  
=> t
```

Related Topics

[Terminal Antenna Functions](#)

Track Pattern 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.

This topic provides a list of Cadence® SKILL functions associated with track pattern.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateTrackPattern](#)

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.

To color the track pattern, use [dbSetTrackPatternFirstTrackColor](#) or [dbSetTrackPatternColorAlternating](#).

Arguments

<i>d_cellviewId</i>	The database ID of the cellview.
<i>g_isHorizontal</i>	Specifies a horizontal or vertical track pattern. Valid values: t: track pattern is horizontal. nil: track pattern is vertical.
<i>n_startCoord</i>	Specifies the start coordinate of the track pattern.
<i>x_numTracks</i>	Specifies the number of track lines of the track pattern.
<i>n_trackSpace</i>	Specifies the tracking space between two track lines.
<i>tx_layer</i>	Optional argument that specifies routing layer of the track pattern.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Value Returned

<i>d_TrackPatternId</i>	The database ID of the track pattern if the creation is successful.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid or one of the given arguments is invalid.

Related Topics

[Track Pattern Functions](#)

GCell Pattern Function

This topic provides a list of Cadence® SKILL functions associated with GCell pattern.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[dbCreateGCellPattern](#)

[GCellPatterns Attributes](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>g_isHorizontal</i>	Specifies a horizontal or vertical GCell pattern. Valid values: t: GCell pattern is horizontal. nil: GCell pattern is vertical.
<i>n_startCoord</i>	Specifies the starting coordinate of the GCell pattern.
<i>x_count</i>	Specifies the number of edges for the pattern.
<i>n_spacing</i>	Specifies the spacing between edges.
<i>tx_layer</i>	Optional argument that specifies the layer of the GCell pattern.

Value Returned

<i>d_TrackPatternId</i>	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.

Related Topics

GCell Pattern Function

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

GCellPatterns Attributes

Via Functions

This topic provides a list of Cadence® SKILL functions associated with vias.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at anytime.

Related Topics

[Optical Data Functions](#)

[Via Header Functions](#)

[Fixed Via Function](#)

Via GenVia Functions

This section describes the SKILL functions that enable creating and editing of [cdsGenVia](#).

Related Topics

[dbCreateGenViaVariant](#)

[dbIsCellViewGenViaMaster](#)

[dbIsGenVia](#)

[dbIsGenViaHeader](#)

[dbIsGenViaVariant](#)

[dbSetGenViaParamDefault](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

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

<i>d_cellviewId</i>	ID of the cellview where the object is created.
<i>t_name</i>	Name for the genViaVariant object.
<i>t_viaDefName</i>	Name of the viaDef to which the genViaVariant object is associated.
<i>l_paramList</i>	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, version, and 'cutArraySpacing' .

Value Returned

<i>viaVariantId</i>	Database ID of the viaVariant that was created.
<i>nil</i>	An error occurred.

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Examples

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)))
```

Related Topics

Optical Data Functions

dbIsCellViewGenViaMaster

```
dbIsCellViewGenViaMaster(  
    d_cellviewId  
)  
=> t / nil
```

Description

Checks whether the specified cellview ID is of a genViaMaster object.

Arguments

<i>d_cellviewId</i>	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.

Examples

Checks if the cellview *cv* is a genViaMaster object.

```
dbIsCellViewGenViaMaster(cv)
```

Related Topics

[Optical Data Functions](#)

dbIsGenVia

```
dbIsGenVia(  
    d_objId  
)  
=> t / nil
```

Description

Checks whether the specified database object is a genVia object.

Arguments

<i>d_objId</i>	Database ID of the input object.

Value Returned

<i>t</i>	Specified object is a genVia object.
<i>nil</i>	Specified object is not a genVia object.

Examples

Checks if the object `objId` is a genVia object.

```
dbIsGenVia(objId)
```

Related Topics

[Optical Data Functions](#)

dbIsGenViaHeader

```
dbIsGenViaHeader(  
    d_objId  
)  
=> t / nil
```

Description

Checks whether the specified database object is a genViaHeader object.

Arguments

<i>d_objId</i>	ID of the input object

Value Returned

t	Specified object is a genViaHeader object.
nil	Specified object is not a genViaHeader object.

Examples

Checks if the object `objId` is a genViaHeader object.

```
dbIsGenViaHeader(objId)
```

Related Topics

[Optical Data Functions](#)

dbIsGenViaVariant

```
dbIsGenViaVariant(  
    d_objId  
)  
=> t / nil
```

Description

Checks whether the specified database object is a genViaVariant object.

Arguments

<i>d_viaId</i>	ID of the input object

Value Returned

t	Specified object is a genViaVariant object.
nil	Specified object is not a genViaVariant object.

Examples

Checks if the object `objId` is a genViaVariant object.

```
dbIsGenViaVariant(objId)
```

Related Topics

[Optical Data Functions](#)

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

<i>d_viaID</i>	ID of a genVia object whose parameter is to be reset.
<i>t_name</i>	Name of the parameter to be reset to the default value.

Value Returned

<i>t</i>	The specified parameter is reset to the default value.
<i>nil</i>	The specified parameter could not be reset to the default value.

Examples

Sets the parameter `cutWidth` to its default value for genVia object `viaID`.

```
dbSetGenViaParamDefault(viaID "cutWidth")
```

Related Topics

[Optical Data Functions](#)

Via Header Functions

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`. Custom `ViaHeaders` 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.

This topic provides a list of Cadence® SKILL functions related to via header.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[dbFindViaHeaderByName](#)

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_viaDefName</i>	A string identifying a viaDef name.

Value Returned

<i>d_viaHeaderId</i>	The database ID of the viaHeader.
<i>nil</i>	<i>nil</i> is returned if the cellview ID is invalid or there are no viaHeaders corresponding to the specified viaDef.

Examples

```
viaHeader = dbFindViaHeaderByName(cvId "viaArray5")
where cvId is a cellview dbId and "viaArray5" identifies a viaDef.
```

Related Topics

[Via Header Functions](#)

Fixed Via Function

This topic provides a list of Cadence® SKILL functions related to fixed via.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Function

[dbIsFixedVia](#)

dbIsFixedVia

```
dbIsFixedVia(  
    d_objId  
)  
=> t / nil
```

Description

Checks whether the specified database object is a fixedVia object.

Arguments

<i>d_objId</i>	Database ID of an object.

Value Returned

t	The specified object is a fixedVia object.
nil	The specified object is not a fixedVia object.

Examples

Checks if the object `obj` is fixedVia object.

```
dbIsFixedVia(obj)
```

Related Topics

[Fixed Via Function](#)

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.

This topic provides a list of Cadence® SKILL functions related to optical data.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time

Related Topics

[dbCellViewHasOpticalData](#)

[dbIsNetOptical](#)

[dbListTypes](#)

dbCellViewHasOpticalData

```
dbCellViewHasOpticalData(  
    d_cellviewId  
)  
=> t / nil
```

Description

Checks whether the specified cellview has optical data.

Arguments

<i>d_cellviewId</i>	The database ID of a cellview.

Value Returned

<i>t</i>	The cellview contains optical data.
<i>nil</i>	The cellview does not contain optical data

Example

```
dbCellViewHasOpticalData(cv_id) => t / nil
```

Related Topics

[Optical Data Functions](#)

dbIsNetOptical

```
dbIsNetOptical(  
    d_netId  
)  
=> t / nil
```

Description

Checks whether the specified net or signal has optical data.

Arguments

<i>d_netId</i>	The database ID of a net.

Value Returned

<i>t</i>	The object is a net or a signal that has optical data.
<i>nil</i>	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
```

Related Topics

[Optical Data Functions](#)

dbListTypes

```
dbListTypes(  
    t_typeName  
)  
=> l_enumValues / nil
```

Description

Lists the enumerated values for a specified database enumerated type.

Arguments

<i>t_typeName</i>	Specifies the name of the DB enumerated type.
	Note: Currently, only <code>dbSigType</code> is supported.

Value Returned

<i>l_enumValues</i>	List of the enum values.
<i>nil</i>	An invalid type name is specified for the DB enumerated type.

Example

Lists the enum values for the `dbSigType` enumerated type.

```
dbListTypes("dbSigType")  
(  
    "unknown" "signal" "ground" "supply" "clock" "testLatch" "analog" "tieOff" "tieHi"  
    "tieLo"    "scan"  "reset" "optical" "singleModeOptical" "multiModeOptical"  
)
```

Related Topics

[Optical Data Functions](#)

Remaster SKILL Functions

This topic provides a list of Cadence® SKILL functions related to remastering.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[dbRemasterAnyInst](#)

[dbRemasterInstHeader](#)

dbRemasterAnyInst

```
dbRemasterAnyInst(  
    d_instId  
    d_newMasterCvId  
    [ t_paramReset ]  
)  
=> t / nil
```

Description

Remasters the specified instance to the specified master cellview.

Arguments

<i>d_InstId</i>	Database ID of an instance.
<i>d_newMasterCvId</i>	Database ID of the new master cellview.
<i>t_paramReset</i>	Valid parameter reset value. The default is "unknown".

Value Returned

<i>t</i>	The specified instance is remastered to the specified cellview.
<i>nil</i>	The operation failed.

Example

Remasters the instance *inst* to the master cellview *master2*.

```
master1 = dbOpenCellViewByType("lib" "master1" "schematic")  
inst = dbCreateInst(cv master1 "inst" list(3 4) "R0" 1)  
master2 = dbOpenCellViewByType("lib" "master2" "schematic")  
dbRemasterAnyInst(inst master2)
```

Related Topics

[Remaster SKILL Functions](#)

[dbUpdateCellNameInInstNamePrefixDuringRemaster](#)

[resetOnRemaster](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

resetLibList

dbRemasterInstHeader

```
dbRemasterInstHeader(  
    d_instHeaderId  
    d_newMasterCvId  
    [ t_paramReset ]  
=> t / nil
```

Description

Remasters all instances under the specified instance header to the specified master cellview.

Arguments

<i>d_InstHeaderId</i>	Database ID of an instance header.
<i>d_newMasterCvId</i>	Database ID of the new master cellview.
<i>t_paramReset</i>	Valid parameter reset value. The default is "unknown".

Value Returned

<i>t</i>	All instances under the specified under the instance header are remastered to the specified cellview.
<i>nil</i>	The operation failed.

Example

Remasters all instances under the instance header `instHeader` to the master cellview `master2`.

```
master1 = dbOpenCellViewByType("lib" "master1" "schematic")  
inst = dbCreateInst(cv master1 "inst" list(3 4) "R0" 1)  
instHeaderId = inst~>instHeader  
master2 = dbOpenCellViewByType("lib" "master2" "schematic")  
dbRemasterInstHeader(instHeaderId master2)
```

Related Topics

[Remaster SKILL Functions](#)

[dbUpdateCellNameInInstNamePrefixDuringRemaster](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

resetOnRemaster

resetLibList

Alternate Foundry Constraint Group SKILL Functions

This topic provides a list of Cadence® SKILL functions related to specifying alternateFoundry constraint group in designs.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[dbGetPerDesignAFCGName](#)

[dbSetPerDesignAFCGName](#)

dbGetPerDesignAFCGName

```
dbGetPerDesignAFCGName(  
    d_cellViewid  
)  
=> t_cgName / nil
```

Description

Retrieves the alternateFoundry constraint group name for the specified cellview.

Arguments

<i>d_cellViewid</i>	Database ID of a cellview.

Value Returned

<i>t_cgName</i>	Name of the alternateFoundry constraint group name set on the specified cellview.
<i>nil</i>	The operation failed.

Examples

Retrieves `myCGx` as the alternateFoundry constraint group name set on the cellview `cv`.

```
dbGetPerDesignAFCGName(cv)  
=> "myCGx"
```

Related Topics

[usePerDesignAFCG](#)

[dbSetPerDesignAFCGName](#)

dbSetPerDesignAFCGName

```
dbSetPerDesignAFCGName (
    d_cellViewid
    t_cgName
)
=> t / nil
```

Description

Sets the specified alternateFoundry constraint group name on the specified cellview. The function does not change the global alternateFoundry constraint group name.

Arguments

<i>d_cellViewid</i>	Database ID of a cellview.
<i>t_cgName</i>	Name of the constraint group to be set on the specified cellview.

Value Returned

<i>t</i>	The alternateFoundry constraint group name is set on the specified cellview.
<i>nil</i>	The operation failed.

Examples

The alternateFoundry constraint group name `myCG` is specified on the cellview `cv`.

```
dbSetPerDesignAFCGName (cv "myCG")
=> t
```

Related Topics

[usePerDesignAFCG](#)

[dbGetPerDesignAFCGName](#)

Virtuoso Studio Design Environment SKILL Reference

Database Access Functions

Cellview Input/Output Database Access Functions

This topic provides a list of basic editing Cadence® SKILL functions associated with cellview input/output database.

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.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- [dbAllCellViews](#)
- [dbCellViewHasVirtHier](#)
- [dbCheckRecursion](#)
- [dbClose](#)
- [dbComputeBBox](#)
- [dbComputeBBoxNoNLP](#)
- [dbCopyCellView](#)
- [dbCreateLib](#)
- [dbDefineProc](#)
- [dbFeaturePrintInfo](#)
- [dbFindOpenCellView](#)
- [dbFlattenInst](#)

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

- [dbFlattenInst2](#)
- [dbFullLibPath](#)
- [dbFullPath](#)
- [dbGetAnyInstSwitchMaster](#)
- [dbGetCellViewDdId](#)
- [dbGetLibDataCompressionLevel](#)
- [dbGetOpenCellViews](#)
- [dbHasAutoSavedFile](#)
- [dbHasPanicFile](#)
- [dbIsNonSKILLPcell](#)
- [dbOpenAutoSavedCellView](#)
- [dbOpenCellViewByType](#)
- [dbOpenHier](#)
- [dbOpenPanicCellView](#)
- [dbOpenParamCellView](#)
- [dbProduceMemName](#)
- [dbPurge](#)
- [dbRefreshCellView](#)
- [dbRefreshLibDataCompressionLevel](#)
- [dbRegPostSaveAsTrigger](#)
- [dbRegPostSaveTrigger](#)
- [dbRegPurgeTrigger](#)
- [dbRegSaveTrigger](#)
- [dbReopen](#)
- [dbRestoreAndOpenAutoSavedFile](#)
- [dbRestoreAndOpenPanicFile](#)
- [dbSave](#)

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

- dbSearchCDF
- dbSetAutoSave
- dbSetLibDataCompressionLevel
- dbUndefineProc
- dbUnregPostSaveAsTrigger
- dbUnregPostSaveTrigger
- dbUnregPurgeTrigger
- dbUnregSaveTrigger
- dbUnsetLibDataCompressionLevel
- dbWriteSkill
- dbWriteSkillWithLib

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

<i>g_lib</i>	The <i>ddId</i> of the library to search for the cell.
<i>t_cellName</i>	The cell name.
<i>d_contextCellView</i>	A library <i>dbCellViewId</i> to search for the cell if <i>g_lib</i> is not specified.

Values Returned

<i>l_viewName</i>	A list of view names.
<i>nil</i>	The view names are not listed.

Examples

Lists all the view names of 74LS169 cell.

```
dbAllCellViews(libId "74LS169")
=> ("schematic" "symbol")
```


dbCellViewHasVirtHier

```
dbCellViewHasVirtHier(  
    cellviewID  
)  
=> t / nil
```

Description

Determines whether the specified cellview contains virtual hierarchy data.

Arguments

<i>cellviewID</i>	Database ID of a cellview.

Values Returned

t	Cellview contains virtual hierarchy data.
nil	Cellview does not contain virtual hierarchy data.

Examples

Checks if *cellView* contains virtual hierarchy data.

```
dbCellViewHasVirtHier(cellView)  
=>t
```

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 hierarchy.

So whether the hierarchy is A->B->C->A or A->B->C->B, `dbCheckRecursion(A)` catches the recursion.

Arguments

<code>d_cellViewId</code>	Top cellview to check if there is recursion.

Values Returned

<code>t</code>	There is no recursion.
<code>nil</code>	There is recursion along with warning messages.

Examples

Checks for occurrence of `topCellView` in the hierarchy of the `instHeader` master.

```
dbCheckRecursion( topCellView )  
=>t
```

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

<code>d_cellViewId</code>	The cellview to be closed.

Values Returned

<code>t</code>	The cellview is closed.
<code>nil</code>	The cellview is not closed.

Examples

Saves the changes and close the cellview.

```
dbSave(cellview)  
dbClose(cellview)
```

dbComputeBBox

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

Description

Recomputes the given cellview bounding box.

Arguments

<i>d_cellViewId</i>	The cellview ID.

Values Returned

t	The computation is successful
nil	The given cellview id is invalid

Examples

Recomputes the cellview bounding box myBBox.

```
cv = dbOpenCellViewByType( "test" "myBBox" "layout" )  
dbComputeBBox(cv) => t
```

dbComputeBBoxNoNLP

```
dbComputeBBoxNoNLP(  
    d_cellViewId  
)  
=> l_bBox / nil
```

Description

Computes the given cellview bounding box, but excludes all the bounding boxes of the `nlp` labels in the given cellview itself and its reference cellviews.

Arguments

<code>d_cellViewId</code>	The cellview ID.

Values Returned

<code>l_bBox</code>	A bBox.
<code>nil</code>	Errors were encountered.

Examples

```
cv = dbOpenCellViewByType ("test" "hasNLP" "layout" "maskLayout" "w")  
myLabel = dbCreateLabel (cv 67 5:-21 "label1" "lowerLeft" "RO" "stick" 0.75 )  
myLabel~>labelType = "NLPLabel"  
myBoxNoNLP = dbComputeBBoxNoNLP (cv)
```

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.

Arguments

<i>d_srcCvId</i>	Source cellview to be copied.
<i>t_libName</i>	Destination library to which the file is to be copied.
<i>t_cellName</i>	Destination cell name.
<i>t_viewName</i>	Destination view name.
<i>t_version</i>	This argument is preserved for compatibility with earlier versions of the software and should be <code>nil</code> if specified.
<i>g_contextId</i>	This argument is preserved for compatibility with earlier versions of the software and its contents is ignored.
<i>g_overwrite</i>	Specifies whether a cellview in the destination library is to be over written. If this argument is <code>nil</code> (the default value), no cellviews are overwritten. If this argument is <code>t</code> , the cellviews are overwritten.

Values Returned

<i>d_cellViewId</i>	The destination cellview ID.
<code>nil</code>	The cellview is not copied.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

Examples

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
```

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

<i>t_libName</i>	The library name to be created.
<i>t_libPath</i>	Library path to be appended to the library name.

Values Returned

<i>ddId_libId</i>	If the library is created successfully, the <i>ddId</i> of the library is returned.
<i>nil</i>	Otherwise <i>nil</i> is returned.

Examples

```
libId = dbCreateLib( "myLibrary" )  
libId = dbCreateLib( "myLibrary" "/tmp" )
```


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

<i>d_cellView</i>	The cellview ID.
<i>t_fileName</i>	The file which contains unencrypted SKILL code defining procedure pcGenCell.

Values 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.

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.

Values 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)
```

dbFindOpenCellView

```
dbFindOpenCellView(  
    g_libId  
    t_cellName  
    t_viewName  
    [ t_version ]  
)  
=> d_cellViewId / nil
```

Description

Finds an opened cellview.

Cadence recommends using the `dbFindOpenCellViewByName` for this task.

Arguments

<i>g_libId</i>	Specifies the <i>ddId</i> of the library that contains the specified cellview.
<i>t_cellName</i>	Cell name of the cellview.
<i>t_viewName</i>	View name of the cellview.
<i>t_version</i>	This argument is preserved for compatibility with earlier versions of the software and should be <code>nil</code> if specified.

Values Returned

<i>d_cellViewId</i>	The cellview ID of the cellview with the specified <i>libId</i> , cell, and view if the cellview has been opened.
<code>nil</code>	The cellview is not found open.

Examples

Finds the schematic cellview `abc` from the library `lib`.

```
cv = dbFindOpenCellView( lib "abc" "schematic" )
```

Related Topics

[dbFindOpenCellViewByName](#)

dbFindOpenCellViewByName

```
dbFindOpenCellViewByName (
    t_libName
    t_cellName
    t_viewName
)
=> d_cellViewId / nil
```

Description

Finds an opened cellview. Cadence recommends using this function instead of `dbFindOpenCellView` because you do not have to retrieve the database ID of the library.

Arguments

<i>t_libName</i>	Name of the library that contains the specified cellview.
<i>t_cellName</i>	Name of the cellview.
<i>t_viewName</i>	Name of the view.

Values Returned

<i>d_cellViewId</i>	The database ID of the cellview with the specified library, cell, and view if the cellview is open in the memory.
<i>nil</i>	The specified cellview is not open in the memory.

Examples

Finds the layout cellview `testlib/cell/layout` if it is open in the memory.

```
cv = dbFindOpenCellViewByName("testlib" "cell" "layout")
```

Related Topics

[dbFindOpenCellView](#)

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.

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

<i>d_instId</i>	Database ID of the instance to flatten. To flatten a mosaic, specify the <i>mosaicId</i> .
<i>x_levels</i>	Number of levels of hierarchy through which to flatten the instance. Valid Values: an integer between 0 and 32
<i>g_flattenPCells</i>	Specifies whether <i>pcells</i> should be flattened. <i>t</i> indicates that <i>pcells</i> should be flattened. Valid Values: <i>t</i> , <i>nil</i> Default: <i>nil</i>

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

<i>g_preservePins</i>	Specifies whether the pin information should be preserved when flattening. When set to <code>t</code>, <code>dbFlattenInst</code> 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: <code>t</code>, <code>nil</code> Default: <code>nil</code>
<i>g_preserveRODobjs</i>	Specifies whether ROD properties should be preserved. Valid Values: <code>t</code>, <code>nil</code> Default: <code>nil</code>
<i>g_delDetachedBlockages</i>	Specifies whether the detached blockages should be deleted. Valid Values: <code>t</code>, <code>nil</code> Default: <code>nil</code>
<i>g_preservePinFigs</i>	Specifies whether the pin figures should be preserved. If set to <code>t</code>, the function preserves the pin figures. Valid Values: <code>t</code>, <code>nil</code> Default: <code>t</code>
<i>g_preserveTermName</i>	Specifies whether the terminal name should be preserved. If set to <code>t</code>, the function preserves the terminal name. Valid Values: <code>t</code>, <code>nil</code> Default: <code>nil</code>
<i>g_flattenVias</i>	Specifies whether the vias should be flattened or not. If set to <code>t</code>, the function flattens the vias. Values: <code>t</code>, <code>nil</code> Default: <code>t</code>

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

Value Returned

<code>t</code>	The instance is flattened.
<code>nil</code>	The instance is not flattened.

Examples

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( instance1 2 nil )
```

Flattens the instance `instId` and any `pcells` contained in the instance, preserves pin information, pin figures, and terminal names.

```
dbFlattenInst(instId 1 t t nil nil t t)
```

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: 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

<code>d_instId</code>	Specifies the database ID of the instance to flatten. Valid value: A valid instance <code>db_Id</code> .
<code>x_levels</code>	Number of levels of hierarchy through which to flatten the instance. Valid value: An integer between 0 and 32.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

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`.
- **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`

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

- `skipSteiners`: When specified, skips Steiners while flattening an instance. Values: `t`, `nil`. Default: `nil`
- `skipSteiners`: When specified, skips Steiners while flattening an instance. Values: `t`, `nil`. Default: `nil`
- `skipFluidShapes`: When specified, skips fluid shapes 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`.
- `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`.
- `preserveNetsWithNoTerms`: Specifies whether nets that are not connected to any terminal are to be preserved. Valid Values: `t`, `nil`. Default: `nil`.
- `preservePinFigLabels`: Specifies whether pin figure labels are preserved during flattening. If set to `nil`, the labels are not flattened. The default is `t`. This option will have no effect if option `preservePins` is set to `nil`.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

Values Returned

<code>t</code>	Returns <code>t</code> if the instance is successfully flattened.
<code>nil</code>	Returns <code>nil</code> if the instance is not flattened.

Examples

Flattens the instance `instance1` through two levels of the hierarchy. It uses default values for `g_options` argument.

```
dbFlattenInst2( instance1 2)
```

Flattens the instance `instance2` through three levels of the hierarchy. It uses values as provided by the user for the `g_options` argument.

```
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 `instId` at two levels, and a Steiner, if any, is not copied to the parent cell.

```
options = list(nil)
options->skipSteiners = 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.

Vias are flattened since `flattenVias` option is true by default.

```
options = list(nil)
options->flattenPCells = t
options->useViaPurpose = t
```

```
dbFlattenInst2(instId 2 options)
```

dbFullLibPath

```
dbFullLibPath(  
    t_libName  
)  
=> t_fullLibPath / nil
```

Description

Returns the full library path of the specified library.

Arguments

<i>d_libName</i>	The name of the library.

Values Returned

<i>t_fullPath</i>	Full path for library.
<i>nil</i>	There is an error.

Examples

```
dbFullLibPath(myLibrary ) /myDir/myLibrary
```

dbFullPath

```
dbFullPath(  
    d_cellView  
)  
=> t_fullPath / nil
```

Description

Returns the full path of a cellview.

Arguments

<i>d_cellView</i>	The cellview.

Value Returned

<i>t_fullPath</i>	Full path for cellview <i>d_cellView</i> .
<i>nil</i>	There is an error.

Examples

Get the full path of a cellview through `dbFullPath(cellView)` and `cellview~>fileName`, which should yield identical results.

```
dbFullPath(cellview)  
cellview~>fileName
```

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

<i>d_anyInst</i>	Instance of a master from which a different view is desired.
<i>t_viewList</i>	A string listing view names to switch into.

Values Returned

<i>d_cellView</i>	A different view of a master cell instance if successful.
nil	No other view exists or an error occurs.

Examples

Switch into a schematic view from a symbol.

```
schCV = dbGetAnyInstSwitchMaster(inst "schematic")
```

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

<i>d_dbCellViewId</i>	A database cellview ID.

Values Returned

<i>ddId_cellViewId</i>	A <i>ddId</i> associated with the <i>cellViewId</i> .
<i>nil</i>	No cellview ID exists or if an error occurs.

Examples

```
dbGetCellViewDdId( d_cvId )  
=> ddId
```

dbGetLibDataCompressionLevel

```
dbGetLibDataCompressionLevel(  
    dd_libId  
)  
=> value/ nil
```

Description

Returns the data compression level set on the specified library ID.

Arguments

<i>dd_libId</i>	The ID of the library whose data compression level must be returned.

Values Returned

<i>value</i>	The value of data compression level.
<i>nil</i>	Error occurred during execution.

Examples

Returns the data comparison level set on the `libId`.

```
dbGetLibDataCompressionLevel( libId )  
=> 1
```


dbGetOpenCellViews

```
dbGetOpenCellViews(  
    )  
=> (d_cv d_cv d_cv ...) / nil
```

Description

Returns all opened cellviews in virtual memory.

Arguments

None

Values Returned

<i>d_cv d_cv ...</i>	All opened cellviews in virtual memory.
<i>nil</i>	No cellview is opened.

dbHasAutoSavedFile

```
dbHasAutoSavedFile(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> t / nil
```

Description

Determines whether the specified cellview has an auto-saved file.

Arguments

<i>t_libName</i>	Name of a library.
<i>t_cellName</i>	Name of a cellview.
<i>t_viewName</i>	Name of a view.

Values Returned

<i>t</i>	The auto-save file is present in the specified cellview.
<i>nil</i>	The auto-save file is not present in the specified cellview.

Examples

```
;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
```

dbHasPanicFile

```
dbHasPanicFile(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> t / nil
```

Description

Determines whether the specified cellview has a panic file.

Arguments

<i>t_libName</i>	Name of a library.
<i>t_cellName</i>	Name of a cellview.
<i>t_viewName</i>	Name of a view.

Values Returned

<i>t</i>	Panic file is present in the specified cellview.
<i>nil</i>	Panic file is not present in the specified cellview.

Examples

```
;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
```

dbIsNonSKILLPcell

```
dbIsNonSKILLPcell(  
    d_cellviewId  
)  
=> t / nil
```

Description

Checks whether a pcell is not a SKILL pcell.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

Values Returned

t	The pcell is not a SKILL pcell.
nil	The pcell is a SKILL pcell.

Examples

```
dbIsNonSKILLPcell(cvId)  
=>nil
```

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

<i>t_libName</i>	Library name.
<i>t_cellName</i>	Cell name.
<i>lt_viewName</i>	One or more view names in one of two forms: <i>lt_viewName</i> is a SKILL list of view names, such as <code>list("viewA" "viewB")</code> <i>t_viewName</i> is a string containing one or more view names separated by spaces, such as "<code>viewA viewB</code>". Cellviews of the cell <i>t_cellName</i> are searched for a name matching one from the list. If a match is found, that cellview is opened.
<i>t_version</i>	This argument is preserved for compatibility with earlier versions of the software and should be <code>nil</code> if specified.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

Values Returned

<i>d_cellView</i>	The <i>dbObject</i> of the open cellview.
<i>nil</i>	An error is detected.

Examples

Open the auto-saved file for the `layout` view of `cellA` from the `test` library.

```
cellview = dbOpenAutoSavedCellView("test" "cellA" "layout")
```

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

<i>gt_lib</i> nil	Specifies the <i>ddlId</i> of a library, the name of a library <i>t_lib</i> , or nil. If it is nil, then a search is made for the library of context cellview <i>d_contextCellView</i> . If the name of a library is given, the library must exist in the <i>libList</i> .

<i>t_cellName</i>	Specifies the cell name.

<i>lt_viewName</i>	Specifies one or more view names in one of the following two forms:

- In the *lt_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.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

<i>t_viewTypeName</i>	Specifies the view type. If it is omitted or entered as <code>nil</code> or <code>"*.cdb"</code>, the cellview must exist. This argument supports different types of cellviews, some of which are specific to certain flows. Some of the commonly used <code>cellViewTypes</code> are: <code>"maskLayout"</code>, <code>"schematic"</code>, <code>"schematicSymbol"</code>, and <code>"netlist"</code>. Note: Schematics use <code>cellViewType "schematic"</code> and schematic symbols use <code>cellViewType "schematicSymbol"</code>. You need to specify a <code>cellViewType</code> to create a new cellview. An error occurs if you try to create a cellview with an unsupported type.
<i>t_mode</i>	Specifies the access mode to the cellview. The mode can be one of the following values: ■ <code>r</code> Opens the cellview in read mode. The cellview must already exist. ■ <code>a</code> Opens the existing cellview in append mode. If the cellview does not exist, it is created. ■ <code>w</code> 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. ■ <code>s</code> Opens the cellview in scratch mode. If the cellview does not exist, this mode is the same as <code>"w"</code> mode, which means that the cellview will have the same state and content as a cellview opened in <code>'w'</code> mode. However, the cellview cannot be saved to disk.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

- `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.

- `w $\bar{d}$`
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.
- `a $\bar{d}$`
Same as "`a`" if the cellview already exists. If the cellview does not exist, this mode is the same as "`w $\bar{d}$` " mode.
- `sc`
Same as "`s`" if the cellview already exists. If the cellview does not exist, this mode is the same as "`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.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

- `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.

Values 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")
```

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

<i>d_cellView</i>	Cellview that represents level 0 (zero) of a design hierarchy.
<i>x_numLevels</i>	Number of design hierarchy levels to open.

Values Returned

<i>t</i>	Always returns <i>t</i> and reports errors as encountered.

Examples

Opens the design hierarchy up to and including level one, which means open and bind the master of instances contained in the given cellview.

`dbOpenCellViewByType` does not automatically open and bind the masters.

```
cellview = dbOpenCellViewByType("design_gpdK090" "ADDFX2" "layout")
```

`dbOpenHier()` does not switch views. It only opens views that are directly instantiated.

```
dbOpenHier( cellview 1 )
```

Additional Information

The SKILL functions `dbOpenHier`, `dbShapeQuery`, and `dbGetOverlaps` perform different functions and have different implementations that may traverse the hierarchy in different ways. Although it is possible to use combinations of these and other functions to get similar results, code that is sensitive to the order of traversal may produce different results. For example, code that is traversing the hierarchy through `dbOpenHier` and `dbShapeQuery`, which needs to expand through Pcells, may produce a result that is different from `dbGetOverlaps`.

if there is a Pcell evaluation failure at some point of the traversal because these functions cannot guarantee order and it cannot be predicted whether or when a Pcell will fail to evaluate, some data that is produced by one set of SKILL functions may not be produced by a different set of SKILL functions.

Related Topics

[dbShapeQuery](#)

[dbGetOverlaps](#)

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

<i>gt_lib</i>	Specifies the <i>ddlId</i> of a library, the name of a library <i>t_lib</i> . Specifies which library to search the cellview in. It must exist in the <code>cds.lib</code> file.
<i>t_cellName</i>	Specifies the cell name.
<i>t_viewName</i>	Specifies the view name.
<i>t_version</i>	This argument is preserved for compatibility with earlier versions of the software and should be <code>nil</code> if specified.
<i>t_mode</i>	This argument is preserved for compatibility with earlier versions of the software and should be <code>r</code> if specified. The cellview opened by this function is always in "read" mode.
<i>d_contextCellView</i>	This argument is preserved for compatibility with earlier versions of the software and its content is ignored.

Values Returned

<i>d_cellView</i>	The <code>dbObject</code> of the cellview opened.
<code>nil</code>	An error is detected.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

Examples

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)
```

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

<i>t_libName</i>	Name of a library.
<i>t_cellName</i>	Name of a cellview.
<i>t_viewName</i>	Name of a view.
<i>l_params</i>	List of <i>l_param</i> , where <i>l_param</i> is <i>list(t_propName t_propType g_value)</i> . If parameter values are not specified, the default values are used.

Values Returned

<i>d_cellView</i>	Database ID of the cellview that is opened.
<i>nil</i>	The operation fails.

Examples

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))
```

dbOpenScratchDesign

```
dbOpenScratchDesign(  
    t_libraryName  
    t_cellViewType  
)  
=> d_cellViewID
```

Description

Opens a scratch design of the specified cellview type scoped to the specified library.

The function allows you to create an in-memory design or a scratch-pad area that does not require any directories or files to exist on the disk, does not require locking, and is guaranteed to be unique. It can be used as a scratch-pad in which you can do perform what-if experiments without committing anything to disk.

You can use the scratch design as a regular design, but it has the following restrictions:

- It cannot be saved in any manner.
- It cannot be used as a custom via master.
- It can be used as an instance master only when instantiated in another scratch design.

Arguments

<i>t_libraryName</i>	Name of the library to which the scratch design is scoped.
<i>t_cellViewType</i>	Cellview type of the scratch design. For example, "maskLayout" or "schematic".

Value Returned

<i>d_cellViewID</i>	Cellview ID of a unique scratch design.

Examples

Opens a layout cellview and a scratch layout design and creates an instance of the layout cellview in the scratch layout design.

```
\i layoutDB = dbOpenCellViewByType("testlib" "cell0" "layout" "maskLayout" "r")  
\t db:0x2990e59b  
\i scratchDB = dbOpenScratchDesign("testlib" "maskLayout")
```


Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

```
\t db:0x2790e61a
\i dbCreateInst(scratchDB layoutDB "I0" 0:0 "R0")
\t db:0x2720de1
```

The scratch design does not exist in the library.

```
\i ddGetObj("testlib" scratchDB~>cellName)
\t nil
```

The scratch design cannot be saved.

```
\i dbSave(scratchDB)
\w *WARNING* (DB-270000): dbSave: Attempt to save a scratch-mode design.
\t nil
```

dbProduceMemName

```
dbProduceMemName (
    t_name
)
=> list( <memName> ... <memName> ) / nil
```

Description

Returns a list of member names for the given name.

Arguments

<i>t_name</i>	Name whose members needs to be generated.

Values Returned

<i>list</i>	Returns list of the member names.
<i><memName></i>	Member names.
<i>nil</i>	There are errors.

Examples

Returns a list of member names for "7".

```
dbProduceMemName ("7")
=> ("7")
```

Returns a list of member names for "(X)".

```
dbProduceMemName ("(X)")
=> ("X")
```

Returns a list of member names for "(7)".

```
dbProduceMemName ("(7)")
("7")
```

Returns a list of member names for "()".

```
dbProduceMemName ("()")
=> nil
```

Returns a list of member names for "x(a)".

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

```
dbProduceMemName ("x(a) ")
=> ("x#28a#29")
```

Returns a list of member names for "x(3)".

```
dbProduceMemName ("x(3) ")
=> ("x(3) ")
```

Returns a list of member names for "(a)x".

```
dbProduceMemName ("(a)x")
=> nil
```

Returns a list of member names for "(2)x".

```
dbProduceMemName ("(2)x")
=> nil
```

Returns a list of member names for "<*3>(0)".

```
dbProduceMemName ("<*3>(0) ")
=> ("0" "0" "0")
```

Returns a list of member names for "<*3>(A)".

```
dbProduceMemName ("<*3>(A) ")
("A" "A" "A")
```

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

<code>d_cellView</code>	Cellview to purge.

Values Returned

<code>t</code>	The cellview is closed.
<code>nil</code>	The cellview is not closed.

Examples

Purge this cellview.

```
dbPurge (cv)
```

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

<i>d_cellViewId</i>	The database ID of the cellview.

Values Returned

t	The cellview was refreshed.
nil	The cellview was not refreshed.

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.

A translator running in another session can change the compression level of a library that is in use in the Virtuoso Studio 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

<i>dd_libId</i>	The ID of the library whose data compression level must be refreshed.

Values Returned

<i>t</i>	Returns <i>t</i> if the data compression level of the specified library ID is successfully refreshed.
<i>nil</i>	Returns <i>nil</i> if the data compression level of the specified library ID could not be refreshed.

Examples

```
dbRefreshLibDataCompressionLevel( dd_libId )  
=> t
```

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

<i>S_triggerFunc</i>	User-defined trigger function.
<i>x_priority</i>	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.

Values Returned

<i>t</i>	The trigger function gets registered successfully.
<i>nil</i>	The registration fails.

Examples

```
dbRegPostSaveAsTrigger("myPostSaveAsTrigger1" 0)
```

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

<code>S_triggerFunc</code>	User-defined trigger function.
<code>x_priority</code>	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.

The trigger function can be defined after calling `dbRegPostSaveTrigger`, but, in that case, you will get a warning message.

Values Returned

<code>t</code>	The trigger function gets registered successfully.
<code>nil</code>	The registration fails.

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

<i>S_triggerFunc</i>	User-defined trigger function.
<i>x_priority</i>	Priority of the trigger functions registered. This argument is optional. The default is the lowest priority.

Values Returned

<i>t</i>	The trigger function gets registered successfully.
<i>nil</i>	The registration fails.

Examples

```
procedure( myTrigFunc(cv) printf("cellview - %d\n", cv) )  
dbRegPurgeTrigger('myTrigFunc 10)
```

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

<i>S_triggerFunc</i>	SKILL trigger function. The function must be defined as: <i>funcName(d_cellviewId)</i>
<i>x_priority</i>	Optional priority. Defaults to <code>INT_MAX</code> (lowest priority).

Values Returned

<i>t</i>	The trigger was successfully registered.
<i>nil</i>	An error occurred during execution.

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) - 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.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

Values Returned

t	The mode change is successful.
nil	The mode change is not successful.

Examples

Re-open the cellview for append mode.

```
cv = dbOpenCellViewByType("lib" "decoder" "schematic" "" "r")
dbReopen(cv "a")
cv~>mode
=> "a"
```

Re-open the cellview for scratch mode.

```
dbReopen(cv "s")
cv~>mode
=> "s"
```

dbRestoreAndOpenAutoSavedFile

```
dbRestoreAndOpenAutoSavedFile(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> d_cellviewID / nil
```

Description

Restores and opens the auto-saved file for the specified cellview.

Arguments

<i>t_libName</i>	Name of a library.
<i>t_cellName</i>	Name of a cellview.
<i>t_viewName</i>	Name of a view.

Values Returned

<i>d_cellviewID</i>	Database ID of the cellview that opens.
<i>nil</i>	Returned in case of failure.

Examples

Checks if cellview has an auto-save file

```
dbHasAutoSavedFile("designLib" "testing1" "layout1")  
=>t
```

Restores and open the auto-save file

```
CV2 = dbRestoreAndOpenAutoSavedFile("designLib" "testing1" "layout1")  
=>db:0x08a1ec92
```

dbRestoreAndOpenPanicFile

```
dbRestoreAndOpenPanicFile(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> d_cellviewID / nil
```

Description

Restores and opens the panic file for the specified cellview.

Arguments

<i>t_libName</i>	Name of a library.
<i>t_cellName</i>	Name of a cellview.
<i>t_viewName</i>	Name of a view.

Values Returned

<i>d_cellviewID</i>	Database ID of the cellview that opens.
<i>nil</i>	Returned in case of failure.

Examples

Checks if cellview has a panic file

```
dbHasPanicFile("designLib" "testing1" "layout1")  
=>t
```

Restores and open the panic file

```
panicCV2 = dbRestoreAndOpenPanicFile("designLib" "testing1" "layout1")  
=>db:0x08a1ec92
```

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.

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

<i>d_cellView</i>	The cellview to be saved.
<i>d_destCellView</i>	Specifies a cellview to save the data into. In the first form, this destination cellview is specified by a <i>dbObject</i> .
<i>t_libname</i>	The library name.
<i>t_cellName</i>	Specifies the cell name.
<i>t_viewName</i>	Specifies one or more view names in the <i>l_viewName</i> or <i>t_cellName</i> form.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

Values 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")
```


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)

Values Returned

`t`

The operation was successful.

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

<code>g_isAutoSaveSet</code>	Specifies whether or not to set the automatic save feature. The default value is <code>nil</code> , which means the feature is not set; <code>t</code> sets the save feature.

<code>u_saveInterval</code>	Specifies the time interval in seconds after which the cell is saved automatically. The default time interval is 300 seconds or five minutes.

This value is interpreted as a time interval in seconds.

Value Returned

<code>t</code>	The automatic save feature has been set successfully.
<code>nil</code>	The automatic save feature has not been set successfully.

Examples

Disable the automatic save option.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

```
dbSetAutoSave(nil)  
=> t
```

Set the automatic save option so that data is saved automatically after every 100 seconds.

```
dbSetAutoSave(t 100)  
=> t
```

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.

The user-defined function can be registered only once, usually in the start-up script before calling the `dbSetAutoSave` function.

Arguments

<i>t_callback</i>	Name of the callback function.

Values Returned

<i>t</i>	The user-defined callback function is set.
<i>nil</i>	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.

Examples

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)  
        as_var = as_var + 1  
        res = nil
```

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

```
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.

You should examine the design size and configure a delay in auto-save only for those designs that larger then several megabytes.

Related Topics

[dbSetAutoSave](#)

dbSetLibDataCompressionLevel

```
dbSetLibDataCompressionLevel(  
    dd_libId  
    [ n_compressionLevel ]  
)  
=> t/ nil
```

Description

Sets the data compression level for designs that are saved to the specified library ID.

Arguments

<i>dd_libId</i>	The ID of the library whose data compression level must be set.
<i>n_compressionLevel</i>	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.

Values Returned

<i>t</i>	Returns <i>t</i> if the data compression level is successfully set on the specified library ID.
<i>nil</i>	Returns <i>nil</i> if the data compression level could not be set.

Examples

```
dbSetLibDataCompressionLevel( dd_libId 1 )  
=> t
```

Additional Information

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.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

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.

This process uses the `zlib` compression algorithm. The compression level refers to the number of passes the algorithm makes as it tries to compress the data. More passes results in more compression, but is affected by the diminishing returns. Additionally, for some types of data more passes can increase the size of data instead of decreasing it. However, more passes do not change the format of the compressed data, therefore, there are no compatibility issues between the compression levels.

Compression is implemented at the OpenAccess database level and is fully supported by any application or tool that reads and writes the OpenAccess data. This is a transparent process and the applications are not aware that the data is compressed.

Data is decompressed into memory when it is read from disk and is re-compressed only when writing back to the disk. Decompression does not require a lot of time and is much faster than having to push as much data across the network through NFS to read the data. Compressed data performs faster than the non-compressed data when the network is involved. However, the performance benefits are highly dependent on the users network and how the data is organized. The performance of writing compressed data is significantly higher (and increases as the number of passes specified by the compression level increases).

Since data is read much more frequently than it is written, exchanging longer write times for shorter read times is usually beneficial unless a non-default compression level is used. Different compression levels can be used within the same library and the same design hierarchy. This means that the same cell can even have different views with different compression levels and the same cellview may instantiate different masters with different compression levels. The compression level of the library is only the compression level that is used when new data is written to that library. SKILL functions are not aware of this process and cannot even detect that they are operating on compressed data. This is why SKILL functions that operate across hierarchies in a mixed-compression- level environment are not affected in anyway.

dbUndefineProc

```
dbUndefineProc(  
    d_cellView  
)  
= t / nil
```

Description

Changes the parameterized cell specified by the *d_cellView* to be a regular cellview.

Arguments

<i>d_cellView</i>	Specifies the cellview ID.

Values Returned

t	The parameterized cell gets changed to a regular cell successfully.
nil	Errors are encountered.

dbUnregPostSaveAsTrigger

```
dbUnregPostSaveAsTrigger(  
    S_triggerFunc  
)  
=> t / nil
```

Description

Allows users to unregister the trigger function called after a postSaveAs operation

Arguments

<i>S_triggerFunc</i>	User-defined trigger function that you want to unregister

Values Returned

t	The trigger function is successfully unregistered
nil	The trigger function is not unregistered

Examples

```
dbUnregPostSaveAsTrigger("myPostSaveAsTrigger1")
```

dbUnregPostSaveTrigger

```
dbUnregPostSaveTrigger(  
    s_triggerFunc  
)  
=> t / nil
```

Description

Un-registers the post save trigger function from `dbSave`.

Arguments

<i>s_triggerFunc</i>	User-defined trigger function.

Values Returned

<i>t</i>	The function is un-registered.
<i>nil</i>	The function is not un-registered.

dbUnregPurgeTrigger

```
dbUnregPurgeTrigger(  
    s_triggerFunc  
)  
=> t / nil
```

Description

Unregisters a user-defined trigger function.

Arguments

<i>s_triggerFunc</i>	User-defined trigger function.

Values Returned

t	The trigger function gets unregistered successfully.
nil	The un-registration fails.

Examples

```
dbUnregPurgeTrigger('myTrigFunc)
```

dbUnregSaveTrigger

```
dbUnregSaveTrigger(  
    S_triggerFunc  
)  
=> t / nil
```

Description

This function unregisters the SKILL preSave trigger function specified by the given name.

Arguments

<i>S_triggerFunc</i>	SKILL trigger function. The function must be defined as: <i>funcName(d_cellviewId)</i>

Values Returned

t	The trigger was successfully unregistered.
nil	An error occurred during execution.

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

<i>dd_libId</i>	The ID of the library whose data compression level must be removed.

Values Returned

<i>t</i>	Returns <i>t</i> if the data compression level is successfully removed for the specified library ID.
<i>nil</i>	Returns <i>nil</i> if the data compression level could not be removed.

Examples

Removes data from *dd_libId*.

```
dbUnsetLibDataCompressionLevel( dd_libId )  
=> t
```

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.

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

<i>d_cellViewId</i>	ID of the source cellview.
<i>t_fileName</i>	Target SKILL filename.
<i>t_mode</i>	Modes. Valid Values: <i>w</i> (overwrite), <i>a</i> (append) Using <i>a</i> mode, you can dump a few designs into one file.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

<i>t_release</i>	The release in which the file will be loaded. The valid values are "4.3", "4.4", "5.0", and "6.1".
<i>g_conn</i>	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.
<i>g_ref</i>	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.
<i>g_lppString</i>	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.

Values Returned

t	The function was successful.
nil	The function failed.

Examples

In the 6.1 or later, do

```
dbWriteSkill(cvId "/tmp/bar.il" "w" "6.1")
```

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:

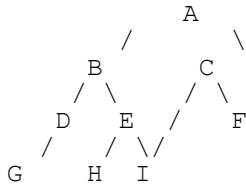
```
procedure( dbD_10056748Ref0() ... )
procedure( dbD_10055724Ref0() ... )
procedure( dbD_10055724Ref1() ... )
procedure( dbD_10055724Ref2() ... )
procedure( dbD_10055724Ref3() ... )
```

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.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

For example, create cellview A with the following reference cellviews:



In this example, the sequence of creating cellviews is G, D, H, I, E, B, F, C, and A.

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.

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.

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*.

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.

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

<i>d_topCellViewId</i>	The top design cellview ID.
<i>t_fileName</i>	Target SKILL filename.
<i>t_mode</i>	Modes. Valid Values: <code>w</code> (overwrite) or <code>a</code> (append) Using <code>a</code> mode, you can dump a few designs into one file.
<i>t_release</i>	The release in which the file will be loaded.
<i>g_conn</i>	The default is to dump out everything. If this argument is set to <code>TRUE</code> , then only the connectivity of the given cellview will be written out.

Virtuoso Studio Design Environment SKILL Reference

Cellview Input/Output Database Access Functions

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.

Values Returned

t The file was created.

nil The file was not created.

Examples

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`.

```
cvId2 = dbOpenCellViewByType("dummyLib" "cell2" "layout" nil "r")
```

```
dbWriteSkillWithLib( cvId2 "z4.il" "w" "6.1")
```

Related Topics

[dbWriteSkill](#)

[dbWriteSkillWithLib](#)

Instance Creation and Retrieval Database Access Functions

This topic provides a list of Cadence® SKILL functions associated with Instance Creation and Retrieval Database Access.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- [dbCreateInst](#)
- [dbCreateParamInst](#)
- [dbCreateExtParamInst](#)
- [dbCreateExtParamInstByMasterName](#)
- [dbCreateXformPCell](#)
- [dbDumpPcDefinePcell](#)
- [dbCheckParamCell](#)
- [dbEvalParamCell](#)
- [dbFixParamCell](#)
- [dbHasInstance](#)
- [dbIsCellViewModified](#)
- [dbReplaceParam](#)
- [dbReplaceInstParamList](#)
- [dbFindAnyInstByName](#)
- [dbFindMemInstByName](#)
- [dbGetHierPathTransform](#)

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

- [dbGetInstByName](#)
- [dbGetInstanceByName](#)
- [dbGetInstNameNumInst](#)
- [dbGetInstTransform](#)
- [dbIsAnyInstBaseNameUsed](#)
- [dbCreateInstByMasterName](#)
- [dbCreateParamInstByMasterName](#)
- [dbDumpPcell](#)
- [dbDumpPcellWithLib](#)
- [dbCreateViaShapeArray](#)
- [dbSetInstHeaderMasterName](#)
- [dbIsSameMaster](#)
- [dbIsObjectBound](#)
- [dbGetCellViewSymmetry](#)
- [dbSetCellViewSymmetry](#)
- [dbGetCellViewEEQMaster](#)
- [dbSetCellViewEEQMaster](#)
- [dbIsInstTransparent](#)
- [dbSetInstTransparent](#)

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*. This function creates a named object and has a ROD ID associated with this object.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

Arguments

<i>d_cellView</i>	Cellview where the instance is placed.
<i>d_master</i>	Master cellview of the instance to be created. This parameter must be a super master; use of a submaster can cause unexpected behavior.
<i>t_name</i>	If specified, the instance is named by <i>t_name</i> ; if <i>nil</i> the program assigns a unique name to the instance. <i>t_name</i> 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.
<i>l_point</i>	Origin of the new instance.
<i>t_orient</i>	Orientation of the new instance.
<i>x_stopIndex</i>	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 <i>stopIndex</i> +1. Default value: 'nil Note: If <i>t_name</i> is given, any value specified by <i>x_stopIndex</i> is overridden. (that is, the number of instances is derived from the name, instead of being taken from the <i>x_stopIndex</i> argument).
<i>g_physOnly</i>	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 <i>t</i> , the cellview is physical only. Default is <i>nil</i> .

Value Returned

<i>d_inst</i>	The <i>dbObject</i> of the instance.
<i>nil</i>	The instance is not created.

Examples

```
dbCreateInst(inCellView masterCV "inst1" list(0 0) "R0" 1)
dbCreateInst(inCellView masterDesign "A2"B2"Ins3" list(0 0) "R0")
```

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

Related Topics

[rodGetObj](#)

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. This function creates a named object and has a ROD ID associated with this object.

Arguments

<i>d_cellView</i>	Cellview the instance is placed in.
<i>d_master</i>	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 <code>dbCreateInst</code> except <i>l_params</i> .
<i>t_name</i>	If specified, the instance is named by <i>t_name</i> ; if <i>nil</i> the program assigns a unique name to the instance. <i>t_name</i> must obey the syntax of instance names as defined at the beginning of this chapter.
<i>l_point</i>	Origin of the new instance.
<i>t_orient</i>	Orientation of the new instance.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

<i>x_numInst</i>	Number of instances; if <i>t_name</i> is given, it overrides any value specified by <i>x_numInst</i> (that is, the number of instances is derived from the name instead of taken from the <i>x_numInst</i> argument). If not specified, the default is one.
<i>l_params</i>	List of <i>l_param</i> , where <i>l_param</i> is <i>list(t_propName t_propType g_value)</i> .
<i>g_physOnly</i>	Optional boolean argument that specifies whether the instance is only in physical domain when the argument is <i>t</i> ; or it is in both physical and module domain when the argument is <i>nil</i> . Default is <i>nil</i> .

Value Returned

<i>d_inst</i>	The <i>dbObject</i> of the instance.
<i>nil</i>	The instance is not created.

Examples

```
dbCreateParamInst(inCellView masterCV "inst1" list(0 0) "R0" 1 list(list("w"
"float" 2.0) list("l" "float" 5.0) t))
```

Additional Information

The following example explains the difference between `dbCreateParamInst` and `dbCreateExtParamInst`:

Suppose a super master has parameters A=1 and B= 2. You create a parameter instance with A=2 in a top design and do not specify B, it is inherited.

The effective parameters on the instance will be A=2, B=2.

This top design is saved and purged.

Suppose the super master is modified out of context, and now A=2 and B=1.

You open the top design again and check the parameter instance.

If the parameter instance was first created with `dbCreateParamInst`, the effective parameters will now be A=2, B=1.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

If the parameter instance was first created with `dbCreateExtParamInst`, the effective parameters will now be A=2, B=2.

This means that, `dbCreateExtParamInst` fixes the parameter values of unspecified parameters (here, B) so they do not change in the future if the super master is redefined.

Related Topics

[rodGetObj](#)

[dbCreateExtParamInst](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

Arguments

<i>d_cellView</i>	Cellview in which the instance is placed.
<i>d_master</i>	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.
<i>t_name</i>	The parametrized cell instance is named by <i>t_name</i> ; if <i>nil</i> the program assigns a unique name to the instance. <i>t_name</i> must obey the syntax of instance names as defined at the beginning of this chapter.
<i>l_origin</i>	Origin of the new instance.
<i>t_orient</i>	Orientation of the new instance.
<i>x_numinst</i>	(Optional) Number of instances; if <i>t_name</i> is given, it overrides any value specified by <i>x_numInst</i> (that is, the number of instances is derived from the name instead of taken from the <i>x_numInst</i> argument). If not specified, the default is one.
<i>l_params</i>	(Optional) Specifies a list of <i>l_param</i> , where <i>l_param</i> is <code>list(<i>t_propName</i> <i>t_propType</i> <i>g_value</i>)</code> . If <i>l_params</i> is not specified the default is <i>nil</i> .
<i>g_physonly</i>	(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 <i>t</i> , the cellview is physical only. Default is <i>nil</i> .

Value Returned

<i>d_inst</i>	Returns dbObject of the instance.
<i>nil</i>	Returns <i>nil</i> if the instance is not created.

Examples

```
i3 = dbCreateExtParamInst(cell master "I3" 0:0 "R0" 1 list(list("aInt" "int" 11))  
t)
```

Additional Information

The following example explains the difference between `dbCreateParamInst` and `dbCreateExtParamInst`:

Suppose a super master has parameters A=1 and B= 2. You create a parameter instance with A=2 in a top design and do not specify B, it is inherited.

The effective parameters on the instance will be A=2, B=2.

This top design is saved and purged.

Suppose the super master is modified out of context, and now A=2 and B=1.

You open the top design again and check the parameter instance.

If the parameter instance was first created with `dbCreateParamInst`, the effective parameters will now be A=2, B=1.

If the parameter instance was first created with `dbCreateExtParamInst`, the effective parameters will now be A=2, B=2.

This means that, `dbCreateExtParamInst` fixes the parameter values of unspecified parameters (here, B) so they do not change in the future if the super master is redefined.

Related Topics

[rodGetObj](#)

[dbCreateParamInst](#)

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

<i>d_cellView</i>	Cellview in which the instance is created.
<i>t_libName</i>	Library name of the master cellview.
<i>t_cellName</i>	Cell name of the master cellview.
<i>t_viewName</i>	View name of the master cellview.
<i>t_name</i>	Name of the new Pcell instance. If <i>nil</i> 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.
<i>l_origin</i>	Origin of the new instance.
<i>t_orient</i>	Orientation of the new instance.
<i>x_numinst</i>	(Optional) Number of instances; if <i>t_name</i> is given, it overrides any value specified by <i>x_numInst</i> (that is, the number of instances is derived from the name instead of taken from the <i>x_numInst</i> argument). If not specified, the default is one.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

<i>l_params</i>	(Optional) Specifies a list of <i>l_param</i> , where <i>l_param</i> is <code>list(t_propName t_propType g_value)</code> . If <i>l_params</i> is not specified the default is <code>nil</code> .
<i>g_physonly</i>	(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 <i>t</i> , the cellview is physical only. Default is <code>nil</code> .

Value Returned

<i>d_inst</i>	Database ID of the instance.
<i>nil</i>	The instance was not created.

Examples

```
dbCreateExtParamInstByMasterName(cvId "libMaster" "cellMaster" "viewMaster" "I1"
0:0 "R0" llist(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.

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

<i>d_cellViewId</i>	The database cellview ID.
<i>t_libName</i>	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

<i>d_cellViewId</i>	The Pcell super master cellview ID that was created.
<i>nil</i>	An error occurred.

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

<i>d_cellViewId</i>	The cellview ID. Must be a parameterized cell.
<i>t_pCellFile</i>	Name of the output file.

Value Returned

<i>t</i>	The output file is successfully created.
<i>nil</i>	With warning messages if any errors are encountered.

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

<i>d_cellViewId</i>	The database cellview ID.

Value Returned

<i>t</i>	The specified cellview is a valid parameterized cell.
<i>nil</i>	The specified cellview is not a valid parameterized cell.

Examples

```
dbCheckParamCell(pcell)
```

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

<i>d_cellViewId</i>	The database cellview ID.

Value Returned

t	The super master is successfully evaluated.
nil	The function was not successful.

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

<i>d_cellViewId</i>	The cellview for which the reading, binding, or evaluation has failed.
<i>t_fileName</i>	The source file that contains the SKILL procedure used in the re-definition.

Value Returned

<code>t</code>	The re-definition succeeded.
<code>nil</code>	The re-definition did not succeed or there was nothing to fix.

Examples

```
dbFixParamCell(pcell "newPcell.il")
```

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

<code>d_topCellViewId</code>	The top design cellview ID.
<code>t_regLibName</code>	The name of a new library that will be used to store the regular frozen design hierarchy. The specified library should not exist.
<code>t_pcellLibName</code>	The name of a new library, which is used to store the Pcell frozen design hierarchy
<code>g_cleanup</code>	Optional argument. When set to <code>t</code> , removes unused instances. Default is <code>nil</code> .

Value Returned

<code>d_figGroupId</code>	Returns <code>t</code> if freezes a Pcell design hierarchy successfully.
<code>nil</code>	If the specified library exists, an error is issued and <code>nil</code> is returned.

Examples

```
cvId = dbOpenCellView("testlib" "Atop" "layout")
when( !cvId
    info("Failed to open design(testlib Atop layout) in 'r' mode.\n")
    return(nil)
)
```

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

```
when( !dbFreeze(cvId "fzLib" "pcellLib")
info("Failed to freeze design(testlib Atop layout)
    hierarchically by
        dbFreeze().\n")
    return(nil)
)
```

Related Topics

[Freeze Pcell Instance Form](#)

dbHasInstance

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

Description

Checks whether the specified cellview contains any instances.

Arguments

<i>d_cellViewId</i>	The database cellview ID.

Value Returned

t	The cellview contains instances.
nil	The cellview does not contain instances.

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

<code>d_cellViewId</code>	The database cellview ID.

Value Returned

<code>t</code>	The cellview has been modified.
<code>nil</code>	The cellview has not been modified.

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

<i>d_cellViewId</i>	The cellview whose parameter is to be changed.
<i>t_name</i>	The name of the parameter.
<i>t_type</i>	The type of the parameter.
<i>g_value</i>	The value of the parameter.

Value Returned

<i>d_prop</i>	The property ID of the replaced parameter.
<i>nil</i>	The parameter was not successfully replaced.

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

<i>d_instId</i>	Specifies the ID of the Pcell instance whose parameters are to be changed.
<i>l_paramList</i>	Specifies the list of parameters to be replaced.

Value Returned

<i>t</i>	The instance parameter list was successfully replaced.
<i>nil</i>	The parameter was not successfully replaced.

Examples

```
dbReplaceInstParamList(pInst1 list(list("metal2Layer" "string" "Metal5")  
list("boolVal1" "boolean" nil) list("intLength" "integer" 10)))
```

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

<i>d_cellView</i>	Name of the cellview.
<i>t_name</i>	Name of the instance.

Value Returned

<i>d_inst</i>	The <i>dbObject</i> of the instance.
<i>nil</i>	The instance or mosaic is not retrieved.

Examples

```
dbFindAnyInstByName(inCellView "inst1")
dbFindAnyInstByName(inCellView "myInv<0:1>")
```

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

<i>d_cellView</i>	Name of cellview.
<i>t_name</i>	Name of the instance.

Value Returned

<i>d_inst x_index</i>	The <i>dbObject</i> and the member index of the instance as a list.
<i>nil</i>	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
```

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

```
dbFindMemInstByName(cvSch "m<2>") => nil  
dbFindMemInstByName(cvSch "m<3>") => (db:<dbId#> 0)
```

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

<i>l_dbInstId</i>	List of instance IDs of the hierPath.

Value Returned

<i>x_InstTransform</i>	Concatenated transform of the instance IDs passed.

Examples

```
objectsPaths = dbGetOverlaps(geGetEditCellView() geGetEditCellView()~>bBox t 32 t)  
foreach(objectPath objectsPaths println(dbGetHierPathTransform(objectPath)))
```

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

<i>d_cellView</i>	Name of the cellview.
<i>t_name</i>	Name of the instance.

Value Returned

<i>d_inst</i>	The database object of the instance.
<i>nil</i>	The instance or mosaic is not retrieved.

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

<i>d_cellView</i>	Name of the cellview.
<i>t_name</i>	Name of the instance.

Value Returned

<i>d_inst</i>	The database object of the instance.
<i>nil</i>	The instance or mosaic is not retrieved.

dbGetInstNameNumInst

```
dbGetInstNameNumInst(  
    t_name  
)  
=> x_numInst / nil
```

Description

Retrieves the number of instances of an instance or mosaic

Arguments

<i>t_name</i>	Name of the instance.

Value Returned

<i>x_numInst</i>	The number of instances.
<i>nil</i>	The number cannot be retrieved.

Examples

```
dbGetInstNameNumInst("i") 1  
dbGetInstNameNumInst("i<3>") 1  
dbGetInstNameNumInst("i<0:3>") 4
```

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

<i>l_dbInstId</i>	List of instance IDs of the <code>hierPath</code> .

Value Returned

<i>x_InstTransform</i>	Concatenated transform of all instance IDs passed.

Examples

```
objectsPaths = dbGetOverlaps(geGetEditCellView() geGetEditCellView()~>bBox t 32 t)  
foreach(objectPath objectsPaths println(dbGetInstTransform(objectPath)))
```

dbIsAnyInstBaseNameUsed

```
dbIsAnyInstBaseNameUsed(  
    d_cellView  
    t_name  
)  
=> t / nil
```

Description

Check if the name is used as instance basename in the cellview.

Arguments

<i>d_cellView</i>	Cellview within which to check the base name.
<i>t_name</i>	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.

Examples

```
dbIsAnyInstBaseNameUsed(cv "myInst")
```

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*.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

Arguments

<i>d_cellView</i>	Cellview where the instance is placed.
<i>t_libName</i>	Library name of the master cellview.
<i>t_cellName</i>	Cell name of the master cellview.
<i>t_viewName</i>	View name of the master cellview.
<i>t_instName</i>	If specified, the instance is named by <i>t_name</i> ; if <i>nil</i> the program assigns a unique name to the instance. <i>t_name</i> must obey the syntax of instance names as defined at the beginning of this chapter.
<i>l_origin</i>	Origin of the new instance.
<i>t_orient</i>	Orientation of the new instance.
<i>x_numInst</i>	Number of instances; if <i>t_name</i> is given, it overrides any value specified by <i>x_numInst</i> (that is, the number of instances is derived from the name instead of taken from the <i>x_numInst</i> argument). If not specified, the default is one.
<i>g_physOnly</i>	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 <i>t</i> , the cellview is physical only. Default is <i>nil</i> .

Value Returned

<i>d_inst</i>	The <i>dbObject</i> of the instance.
<i>nil</i>	The instance is not created.

Examples

```
dbCreateInstByMasterName(cv "basic" "nand2" "symbol" "inst1" list(0 0) "R0" 1)
```

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

<i>d_cellView</i>	Cellview in which the instance is created.
<i>t_libName</i>	Library name of the master cellview.
<i>t_cellName</i>	Cell name of the master cellview.
<i>t_viewName</i>	View name of the master cellview.
<i>t_instName</i>	Name of the new Pcell instance. If <code>nil</code> 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.
<i>l_origin</i>	Origin of the new instance.
<i>l_orient</i>	Orientation of the new instance.
<i>x_numInst</i>	Number of instances; if <i>t_name</i> is given, it overrides any value specified by <i>x_numInst</i> (that is, the number of instances is derived from the name instead of taken from the <i>x_numInst</i> argument). If not specified, the default is one.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

<i>l_params</i>	List of <i>l_param</i> , where <i>l_param</i> is <code>list(t_propName t_propType g_value)</code> .
<i>g_physOnly</i>	Optional boolean argument that specifies whether the instance is only in physical domain when the argument is <code>t</code> ; or it is in both physical and module domain when the argument is <code>nil</code> . Default is <code>nil</code> .

Value Returned

<i>d_inst</i>	Database ID of the instance.
<code>nil</code>	The instance was not created.

Examples

```
dbCreateParamInstByMasterName(inCellView "basic" "nand" "symbol" "inst1"
list(0 0) "R0" 1 list(list("w" "float" 2.0) list("l" "float" 5.0) t))
```

dbDumpPcell

```
dbDumpPcell(  
    d_cvId  
    t_outputFile  
    t_pCellFile  
)  
=> t / nil
```

Description

The `dbDumpPcell` function is used in the implementation of the *keepPcell* option, during XStream In and XStream Out translation. Pcells are preserved by writing the Pcells to SKILL files (during XStream Out translation) which are loaded back into Virtuoso Studio (during XStream In translation).

Dumping parameterized cell devices defined in a technology file is not supported in this function.

A supermaster cellview contains two parts. The first part is a cellview database that contains normal cellview components such as shapes, nets, and terminals. The second part is the Pcell script that contains the code to generate the submasters. The first part may be empty and the script may or may not depend on the contents of the first part but both parts must exist. The `dbDumpPcell` SKILL command dumps the cellview part into `t_outputFile` and it dumps the script part into `t_pCellFile`.

Arguments

<code>d_cvId</code>	Must be a valid cellview ID and Pcell.
<code>t_outputFile</code>	Output file which can be reloaded to recreate the Pcell.
<code>t_pCellFile</code>	Output file for the SKILL procedure attached to the Pcell, this file will be given to <code>dbDefineProc</code> when reloading <code>outputFile</code> .

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

Value Returned

<code>t</code>	The files are dumped successfully.
<code>nil</code>	The files are not dumped successfully.

Examples

The following example produces two files `nmos2v_nat.il` and `nmos2v_nat_script.il`.

```
cv=dbOpenCellViewByType("gpdk090" "nmos2v_nat" "layout" nil "r")
db:0x284be39a
dbDumpPcell(cv "nmos2v_nat.il" "nmos2v_nat_script.il")
t
```

- `nmos2v_nat.il` contains a script that can be used to recreate the shapes, such as rectangles, properties, nets, terminals, and pins of the Pcell supermaster. In this part of the dump, all the Pcell parameters appear as properties of the cellview.
- `nmos2v_nat_script.il` contains the Pcell script that is used to generate the submasters. In this section, the Pcell parameters appear as program variables.

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

<i>d_cvId</i>	Must be a valid cellview ID and Pcell.
<i>t_outputFile</i>	Output file which can be reloaded to recreate the Pcell.
<i>t_pCellFile</i>	Output file for the SKILL procedure attached to the Pcell, this file will be given to <code>dbFefineProc</code> when reloading <code>outputFile</code> .

Value Returned

<i>t</i>	The files are dumped successfully.
<i>nil</i>	The files are not dumped successfully.

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

<i>d_techCVID</i>	ID of the technology cellview.
<i>g_viaLayer</i>	Layer on which to create the via shape array.
<i>d_netId</i>	ID of the net to add the via shapes to.
<i>f_x</i>	X coordinate of the starting point.
<i>f_y</i>	Y coordinate of the starting point.
<i>f_xPitch</i>	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.
<i>f_yPitch</i>	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.
<i>f_w</i>	Width (along the X axis) of the rectangle.
<i>f_length</i>	Length (along the Y axis) of the rectangle.
<i>x_row</i>	Number of rows along the Y axis.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

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.

Examples

```
dbCreateViaShapeArray( cvId "contact" netId 10.2 10.1 5.5 6.7 3.1 4.9 2 3 )  
==>list(26.7 23.5)
```

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

<i>d_instHeader</i>	Instance header whose master library name, cell name, and view name are being replaced by the specified names.
<i>t_libName</i>	New master library name. If it is <code>nil</code> , use the one from the <code>instHeader</code> .
<i>t_cellName</i>	New master cell name. If it is <code>nil</code> , use the one from the <code>instHeader</code> .
<i>t_viewName</i>	New master view name. If it is <code>nil</code> , use the one from the <code>instHeader</code> .

Value Returned

<code>t</code>	The library name/cell name/view name of the master is set successfully.
<code>nil</code>	The library name/cell name/view name of the master is not set successfully.

Examples

```
dbSetInstHeaderMasterName(instH1 "basic" "nand2" "symbol")
```

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

<i>d_anyInst1</i>	First instance.
<i>d_anyInst2</i>	Second instance.

Value Returned

<code>t</code>	The two instances are in the same cellview and have the same master.
<code>nil</code>	The two instances are not in the same cellview or if they do not have the same master.

Examples

```
dbIsSameMaster(instId1 instId2)
```

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

<i>g_object</i>	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.

dbGetCellViewSymmetry

```
dbGetCellViewSymmetry(  
    d_cellView  
)  
=> t_symmetry / nil
```

Description

Returns the symmetry of the specified cellview.

Arguments

<i>d_cellviewId</i>	The database ID of the cellview.

Value Returned

<i>t</i>	Returns the symmetry value of the cellview.
<i>nil</i>	The cellview ID is invalid.

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

<i>d_cellviewID</i>	The database ID of the cellview.
<i>t_symmetry</i>	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

<i>t</i>	The symmetry value is set.
<i>nil</i>	The cellview ID is invalid.

Examples

```
cv = geGetWindowCellView(hiGetCurrentWindow())  
dbSetCellViewSymmetry(cv "X")  
dbGetCellViewSymmetry(cv)
```

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

<code>d_cellviewId</code>	The database ID of the cellview.

Value Returned

<code>t_EEQMaster</code>	The the value of the EEQmaster attribute is returned.
<code>nil</code>	The cellview ID is invalid. The EEQMaster attribute is not set or the string value is empty.

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

<i>d_cellviewId</i>	The database ID of the cellview.
<i>t_EEQMaster</i>	String value.

Value Returned

<i>t</i>	The value of the EEQmaster attribute is set.
<i>nil</i>	The cellview ID is invalid. An EEQMaster attribute is not set.

dbIsInstTransparent

```
dbIsInstTransparent(  
    d_instanceId  
)  
=> t / nil
```

Description

Checks if the specified instance ID is set to be transparent to the Virtuoso Layout Suite XL binder.

Arguments

<i>d_instanceId</i>	Database ID of the instance to be checked.

Value Returned

<i>t</i>	The specified instance is set as transparent to the Virtuoso Layout Suite XL binder.
<i>nil</i>	The instance is not transparent. The default value for any instance is <i>nil</i> .

Examples

```
isTransparent = dbIsInstTransparent(instanceId)
```

Returns *t* if the instance *instanceId* is already set as transparent. Otherwise, returns *nil*.

dbSetInstTransparent

```
dbSetInstTransparent
    d_instanceId
    g_transparent
)
=> t / nil
```

Description

Sets an instance as transparent to the Virtuoso Layout Suite XL binder.

Arguments

<i>d_instanceId</i>	Database ID of the instance to be set as transparent.
<i>g_transparent</i>	A Boolean that determines whether instance should be set as transparent. The default is <code>nil</code> .

Value Returned

<code>t</code>	The specified instance is set as transparent.
<code>nil</code>	The specified instance is not set as transparent.

Examples

```
dbSetInstTransparent(inst, t)
```

Makes instance `inst` transparent. When Virtuoso Layout Suite 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`. Virtuoso Layout Suite XL can no longer bind to the instances inside the master of this instance.

Virtuoso Studio Design Environment SKILL Reference

Instance Creation and Retrieval Database Access Functions

Connectivity Database Access Functions

This topic provides a list of Cadence® SKILL functions associated with connectivity database access.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- Connectivity Creation and Modification Functions
- Equivalent Connectivity Time Functions
- Connectivity Access Functions

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](#)

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

- [dbSetCellViewNetVoltageRange](#)
- [dbIsCellViewNetVoltageRangeSet](#)
- [dbIsNetVoltageRangeSet](#)
- [dbUnsetCellViewNetVoltageRange](#)
- [dbGetNetPowerDomain](#)
- [dbSetNetPowerDomain](#)
- [dbRenameNet](#)
- [dbIsCellViewPhysicalOnly](#)
- [dbUnsetCellViewPhysicalOnly](#)
- [dbGetAnyTermGroundSensitivity](#)
- [dbGetAnyTermSupplySensitivity](#)
- [dbIsAnyTermGroundSensitivityConsistent](#)
- [dbIsAnyTermGroundSensitivityConsistent](#)
- [dbIsAnyTermSupplySensitivityConsistent](#)
- [dbSetAnyTermGroundSensitivity](#)
- [dbSetAnyTermSupplySensitivity](#)

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

<code>d_cellView</code>	Cellview the net is created within.
<code>t_name</code>	Name of the net. For a child net, it can be an empty string or <code>nil</code> .
<code>d_parentNet</code>	To create a new top-level net, <code>d_parentNet</code> is either <code>nil</code> or unspecified. To create a child net, the <code>d_parentNet</code> object must not be <code>nil</code> .

Value Returned

<code>d_net</code>	A new net in the cellview, if one does not already exist.
<code>nil</code>	The net is not created or the net already exists.

Examples

Creates a top-level net.

```
parentNet = dbCreateNet(cellview "net")
```

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

```
hierNet = dbCreateNet(cellview "moduleA`moduleB`hierNet")
```

Creates a child net.

```
childNet = dbCreateNet(cellview "childNet" parentNet)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<code>d_parentNetId</code>	Database ID of the root net with which the subnet will be associated. The <code>d_parentNetId</code> also provides the cellview in which the new net is created.
<code>t_subNetName</code>	Name for the new subnet.

Value Returned

<code>t_subnetId</code>	Name of the subnet that is created with the parent net as its root net.
<code>nil</code>	The subnet could not be created or the subnet already exists.

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Examples

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")
```

Related Topics

[dbCreateNet](#)

[Connectivity Creation and Modification Functions](#)

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

<i>d_cellView</i>	Cellview the net is created within.
<i>t_name</i>	Name of the net. For a child net it can be an empty string or <i>nil</i> .
<i>d_parentNet</i>	To create a new top-level net, <i>d_parentNet</i> must be either <i>nil</i> or unspecified. To create a child net, the <i>d_parentNet</i> object must not be <i>nil</i> .

Value Returned

<i>d_net</i>	The new net or the existing top-level net <i>t_name</i> .
<i>nil</i>	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)
```

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Related Topics

Connectivity Creation and Modification Functions

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

<code>d_netId1</code>	The surviving net to which the merging net gets merged into.
<code>d_netId2</code>	The net that is going to get merged into the surviving net.

Value Returned

<code>dNetId1</code>	The merge is successful.
<code>nil</code>	Otherwise, <code>nil</code> is returned.

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>d_cellView</i>	Cellview within which the net is created.
<i>t_prefix</i>	Prefix used in auto-naming the net. nil or an empty string defaults to dbNet. Note: Here, <i>N_</i> cannot be used as prefix to create bus nets because it conflicts with the prefix of the auto-names.
<i>x_numBits</i>	Integer specifying the number of bits for the net. If <i>x_numBits</i> is 1, the name is <i>t_prefix</i> followed by a unique number. If <i>x_numBits</i> is greater than 1, <i>t_prefix</i> is followed by <i><0:numBits-1></i> . If <i>x_numBits</i> is less than -1, <i>t_prefix</i> is followed by <i><abs(numBits)-1:0></i> .

Value Returned

<i>d_net</i>	The net that is created.
nil	The net is not created.

Examples

Create a single bit net named Nx where x is a unique number.

```
net = dbCreateUniqueNamedNet(cellview "N" 1)=> N1
```

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Creates a new net named N<0:3>.

```
net = dbCreateUniqueNamedNet(cvId "N" 4)=> N<0:3>
```

Creates a bus named B<30:0>.

```
bus = dbCreateUniqueNamedNet(cellview "B" -31)=> B<30:0>
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>d_cellView</i>	Cellview within which the mode is set.
<i>g_mode</i>	Mode to be set. Either <code>t</code> or <code>nil</code> .

Value Returned

<code>t</code>	The mode is set successfully.
<code>nil</code>	The function fails.

Example

```
dbSetNetNameDescend(cellview t)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbIsNetNameDescend

```
dbIsNetNameDescend(  
    d_cellView  
)  
=> t / nil
```

Description

Checks if the net name descend mode is set or not.

Arguments

<i>d_cellView</i>	Cellview within which the mode is checked.

Value Returned

t	The mode is set.
nil	The mode is not set.

Example

```
dbIsNetNameDescend(cellview)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbNetIsSig

```
dbNetIsSig(  
    d_netId  
)  
=> t / nil
```

Description

Tests whether the specified net ID is also a signal.

Arguments

<i>d_netId</i>	A net ID.

Value Returned

t	The specified net ID is also a signal.
nil	The specified net ID is not also a signal.

Related Topics

[Connectivity Creation and Modification Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

dbDeleteAllNet

```
dbDeleteAllNet(  
    d_cellView  
)  
=> t / nil
```

Description

Deletes all nets from a cellview.

Arguments

<i>d_cellView</i>	Specifies the cellview.

Value Returned

t	The nets are deleted.
nil	The nets are not deleted.

Example

```
dbDeleteAllNet(cellview)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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 [Database 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.

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Arguments

<i>d_net</i>	Name of the net.
<i>t_name</i>	Name of the terminal to create. If <i>t_name</i> is <code>nil</code> or an empty string, the default is to use the same name as the net <i>d_net</i> .
<i>t_direction</i>	A string specifying the direction of the terminal. See <code>dbIsId</code> for valid terminal directions.
<i>x_unused</i>	This argument is for backward compatibility. An integer must be given.
<i>g_physOnly</i>	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 <code>t</code> , the cellview is physical only. Default is <code>nil</code> .

Value Returned

<i>d_term</i>	The new terminal.
<code>nil</code>	The terminal is not created.

Example

```
netId = dbCreateNet(getEditCellView() "CLK")
termId = dbCreateTerm(netId "myTerm" "input" 0 t)
```

Related Topics

[dbIsId](#)

[Connectivity Creation and Modification Functions](#)

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

<i>d_net</i>	Net to be connected to the instance terminal. If <i>nil</i> , the instance terminal created is left unconnected.
<i>d_inst</i>	Instance of a master cellview where <i>d_term</i> is defined.
<i>d_term</i>	Terminal defined in the master cellview of the instance <i>d_inst</i> .

Value Returned

<i>d_instTerm</i>	The new or existing instance terminal.
<i>nil</i>	The terminal is not created.

Examples

Creates a terminal in the master cellview.

```
net = dbCreateNet(master "A")  
term = dbCreateTerm(master "" "input")
```

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Creates an instance of the master.

```
inst = dbCreateInst(cellview master "" list(0 0) "R0" 1)
```

Creates 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)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbHasShortedTerminals

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

Description

Checks whether the specified cellview contains nets that are connected to multiple terminals.

Arguments

<i>d_cellViewId</i>	The database cellview ID.

Value Returned

<i>t</i>	The specified cellview has shorted terminals.
<i>nil</i>	The cellview ID is invalid or there is an error.

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>d_netId</i>	Net ID for the pin.
<i>d_figId</i>	Figure ID of the pin. The figure <i>purpose</i> attribute is left unchanged.
<i>t_pinName</i>	Pin name <code>nil</code> or an empty string produces an automatically generated name. Pin name <i>t_pinName</i> is optional. If specified, it must be unique among pins of the same terminal. A unique name is automatically generated if <i>t_pinName</i> is not given.
<i>d_termId</i>	Term ID for the pin. Term ID <i>d_termId</i> is optional.

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Value Returned

<i>d_pin</i>	The new pin.
<i>nil</i>	The pin is not created.

Example

```
pin = dbCreatePin(net fig)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>d_net</i>	Net to be connected to the instance terminal.
<i>d_inst</i>	Instance of the master cellview where terminal <i>d_term</i> is defined. <i>d_inst</i> cannot be a mosaic instance. Instance terms are not supported in mosaic instances.
<i>d_term</i>	Terminal defined in the master cellview of instance <i>d_inst</i> .

Value Returned

<i>d_instTerm</i>	The connected instance terminal.
<i>nil</i>	The instance is not connected.

Example

Creates an instance terminal of master terminal `term` on instance `inst` and connect it to `net`.

```
conn = dbCreateConn(net inst term)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<code>d_net</code>	Net to be connected to the instance terminal.
<code>d_inst</code>	Instance of the master cellview where terminal <code>d_term</code> is defined. <code>d_inst</code> cannot be a mosaic instance. Instance terms are not supported in mosaic instances.
<code>t_name</code>	The name of a terminal defined in the master cellview of instance <code>d_inst</code> .

Value Returned

<code>d_instTerm</code>	The connected instance terminal.
<code>nil</code>	The instance terminal is not connected.

Example

Creates an instance terminal of master terminal `A` on instance `inst` and connect it to the `net`.

```
conn = dbCreateConnByName(net inst "A")
```

If you specify `nil` as the `d_net` value, a warning message displays and the instance terminal is not created.

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Related Topics

Connectivity Creation and Modification Functions

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

<code>d_net</code>	Net to be connected to the instance terminal.
<code>d_inst</code>	Instance of the master cellview where terminal <code>t_name</code> is defined. <code>d_inst</code> cannot be a mosaic instance. Instance terms are not supported in mosaic instances.
<code>t_name</code>	Name of a new or existing terminal in the master cellview of instance <code>d_inst</code> .

Value Returned

<code>d_instTerm</code>	The connected instance terminal.
<code>nil</code>	The instance terminal is not created.

Example

Creates 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.

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Related Topics

Connectivity Creation and Modification Functions

dbSubFigFromNet

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

Description

Removes the given figure object from the net object with which it is already associated.

Arguments

<i>d_figId</i>	The given figure object.

Value Returned

t	The figure object was removed.
nil	The figure object was not removed.

Example

When the `dbSubFigFromNet` SKILL function is successful, you can verify the result using:

```
dbSubFibFromNet (figId)
```

Where `figId` is not a member of the returned list.

```
netId~>figs=>list (figs...)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>d_figId</i>	The given figure object.
<i>d_netId</i>	The specified net object.

Value Returned

<i>t</i>	The figure object was assigned.
<i>nil</i>	The figure object was not assigned.

Example

When the SKILL command is successful, you can verify if the figure is added using

```
dbAddFigToNet(figId netId)
```

Where *figId* is a member of the returned list.

```
netId~>figs=>list(figs...)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbGetNetVoltageRange

```
dbGetNetVoltageRange (
    d_netId
)
=> l_voltages / nil
```

Description

Returns the minimum and maximum voltages on a net.

Arguments

<i>d_netId</i>	Database ID of a net.

Value Returned

<i>l_voltages</i>	The list of floating numbers representing the minimum and maximum voltages.
<i>nil</i>	The specified net ID is incorrect.

Examples

The following example returns the minimum and maximum voltage values for the net IN.

```
cv=dbOpenCellViewByType (
    "vdr_demo"
    "inverter_chain"
    "layout_cv"
    ""
    "r"
)
;Opens the specified layout cellview in read-only mode.
cv~>nets~>name
;Prints names of nets available in the specified cellview.
in=dbFindNetByName (
    cv
    "IN"
)
;Retrieves the database ID of the net IN.
```

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

```
printf("dbGetNetVoltageRange for the net IN is %L \n" dbGetNetVoltageRange(in))  
;Returns the minimum and maximum voltages for the net IN.
```

```
=> db:0x2c6fe31a
```

```
=>
```

```
("VSS" "VDD1" "net15" "IN" "VDD2"  
  "net14" "VDD3" "OUT"
```

```
)
```

```
=> db:0x2c6fc59d
```

```
=> dbGetNetVoltageRange for the net IN is (1.021592e-06 1.499999) => t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbSetNetVoltageRange

```
dbSetNetVoltageRange(  
    d_netId  
    n_minVoltage  
    n_maxVoltage  
)  
=> t / nil
```

Description

Sets the minimum and maximum voltages on the specified net.

Arguments

<i>d_netId</i>	The net ID whose minimum and maximum voltages needs to be set.
<i>n_minVoltage</i>	The minimum voltage value to be set for the specified net ID.
<i>n_maxVoltage</i>	The maximum voltage value to be set for the specified net ID.

Value Returned

<i>t</i>	The specified minimum and maximum voltages was set on the specified net.
<i>nil</i>	The specified minimum and maximum voltages was not set on the specified net.

Example

```
dbSetNetVoltageRange(netId 0.5 2.5) => t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbUnsetNetVoltageRange

```
dbUnsetNetVoltageRange(  
    d_netId  
)  
=> t / nil
```

Description

Resets the minimum and maximum voltages on a net to their default values.

Arguments

<i>d_netId</i>	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
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbGetNetVoltageRangeSource

```
dbGetNetVoltageRangeSource(  
    d_netId  
)  
=> t_source / nil
```

Description

Returns the voltage range source of the specified net ID.

Arguments

<i>d_netId</i>	The net ID whose voltage range source must be returned.

Value Returned

<i>t_source</i>	The text string representing the voltage range source of the net ID.
<i>nil</i>	There is no voltage range source for the specified net ID.

Examples

```
dbGetNetVoltageRangeSource(netId) => "user"  
dbGetNetVoltageRangeSource(netId) => "simulation"
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbSetNetVoltageRangeSource

```
dbSetNetVoltageRangeSource(  
    d_netId  
    t_source  
)  
=> t / nil
```

Description

Sets the voltage range source on the specified net ID.

Arguments

<i>d_netId</i>	The net ID whose voltage range source must be set.
<i>t_source</i>	The voltage range source to be set on the specified net ID.

Value Returned

<i>t</i>	The voltage range source was set successfully on the specified net ID.
<i>nil</i>	The voltage range source was not set on the specified net ID.

Example

```
dbSetNetVoltageRangeSource(netId "simulation") => t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbUnsetNetVoltageRangeSource

```
dbUnsetNetVoltageRangeSource (
    d_netId
)
=> t / nil
```

Description

Resets the voltage range source of the specified net ID to its default value.

Arguments

<i>d_netId</i>	The net ID whose voltage range source must be set to its default value.

Value Returned

<i>t</i>	The default voltage range source was reset successfully on the specified net ID.
<i>nil</i>	The default voltage range source was not reset on the specified net ID.

Example

```
dbUnsetNetVoltageRangeSource(netId) => t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbGetCellViewNetVoltageRange

```
dbGetCellViewNetVoltageRange(  
    d_cvId  
)  
=> l_voltages / nil
```

Description

Returns the default minimum and maximum net voltages on the specified cellview ID.

Arguments

<i>d_cvId</i>	The cellview ID whose minimum and maximum default net voltages must be returned.

Value Returned

<i>l_voltages</i>	The list of two floating numbers representing the minimum and maximum default net voltages on the cellview ID.
<i>nil</i>	The specified cellview ID is incorrect.

Example

```
dbGetCellViewNetVoltageRange(cvId) => (1.9 2.2)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbSetCellViewNetVoltageRange

```
dbSetCellViewNetVoltageRange (
    d_cvId
    n_minVoltage
    n_maxVoltage
)
=> t / nil
```

Description

Sets the default minimum and maximum net voltages on a cellview ID.

Arguments

<i>d_cvId</i>	The cellview ID whose minimum and maximum default net voltages must be set.
<i>n_minVoltage</i>	The default minimum net voltage value to be set on the specified cellview ID.
<i>n_maxVoltage</i>	The default maximum net voltage value to be set on the specified cellview ID.

Value Returned

<i>t</i>	The default minimum and maximum net voltages were successfully set on the specified cellview ID.
<i>nil</i>	The default minimum and maximum net voltages were not set on the specified cellview ID.

Example

```
dbSetCellViewNetVoltageRange(cvId 1.5 3.5) => t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbIsCellViewNetVoltageRangeSet

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

Description

Ensures that the voltage range is set on a cellview.

Arguments

<i>d_cvId</i>	Database ID of a cellview.

Value Returned

<i>t</i>	The voltage range is set on the specified cellview.
<i>nil</i>	The voltage range is not on the specified cellview.

Example

```
dbIsCellViewNetVoltageRangeSet (cvId) => t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbIsNetVoltageRangeSet

```
dbIsNetVoltageRangeSet (
    d_netId
)
=> t / nil
```

Description

Ensures that the voltage range is set on a net.

Arguments

<i>d_netId</i>	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
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbUnsetCellViewNetVoltageRange

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

Description

Resets the voltage range on a cellview to its default value.

Arguments

<i>d_cvId</i>	Database ID of a cellview.

Value Returned

<i>t</i>	The default voltage range is reset on the specified cellview.
<i>nil</i>	The default voltage range is not rest on the specified cellview.

Example

```
dbUnsetCellViewNetVoltageRange (cvID) => t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>dd_netId</i>	ID of the net.

Value Returned

<i>l_list</i>	The power domain.
<i>nil</i>	No power domain is associated with the specified net.

Example

```
dbGetNetPowerDomain( netId )  
> ("powerDomain1" "powerDomain2")
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbSetNetPowerDomain

```
dbSetNetPowerDomain(  
    dd_netId  
    l_list  
)  
=> t / nil
```

Description

Sets the power domain for the specified net.

Arguments

<i>dd_netId</i>	ID of the net.
<i>l_list</i>	The power domain; you can specify a single string or a list of strings.

Value Returned

<i>t</i>	The power domain was set successfully for the specified net.
<i>nil</i>	The power domain was not set for the specified net.

Example

```
dbSetNetPowerDomain( netId "powerDomain" )  
> true  
  
dbSetNetPowerDomain( netId list( "powerDomain1" "powerDomain2" ))  
> true
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>d_cellViewId</i>	Specifies the cellview from which data needs to be retrieved.

Value Returned

<i>l_cluster</i>	("cluster1" "cluster2"... "clusterN") List containing multiple entries. Each entry is list of: ("clusterName" "clusterBoundaryName" "maxVoltage")
<i>nil</i>	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)
```

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

```
(( "powerDomain1" "powerDomain1" 3.2)  
  ("powerDomain2" "powerDomain2" 2.5)  
)
```

Related Topics

Connectivity Creation and Modification Functions

dbRenameNet

```
dbRenameNet (
    d_netId
    t_newNetName
)
=> t / nil
```

Description

This function changes the name of the net pointed by `netId` to the `newNetName`.

Arguments

<code>d_netId</code>	Specifies the ID of the net for which you want to change the name.
<code>t_newNetName</code>	Specifies the new name of the net.

Value Returned

<code>t</code>	Returns <code>t</code> if the new name is successfully changed.
<code>nil</code>	Returns <code>nil</code> if the new name cannot be changed.

Example

```
net = dbCreateNet(cv "net1")
dbRenameNet(net "newNet")
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbIsCellViewPhysicalOnly

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

Description

Checks whether the connectivity detail of a cellview is visible only to the block domain or to both the block domain and the OpenAccess module domain.

Arguments

<i>d_cvId</i>	Database ID of a cellview.

Value Returned

<i>t</i>	The cellview is visible to the block domain only.
<i>nil</i>	The cellview is visible to both block and module domains.

Example

Confirms that the cellview *cvId* is visible only to the block domain.

```
dbIsCellViewPhysicalOnly(cvId)  
nil
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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

<i>d_cvId</i>	Database ID of a cellview.

Value Returned

<i>t</i>	Physical-only attribute of the specified cellview is unset.
<i>nil</i>	The operation fails.

Example

Unsets the Physical-only attribute of the cellview *cvId*.

```
dbUnsetCellViewPhysicalOnly(cvId)  
t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbGetAnyTermGroundSensitivity

```
dbGetAnyTermGroundSensitivity(  
    d_termId  
    [ g_all ]  
)  
=> d_groundTermId | l_groundTermId / nil
```

Description

Returns ground sensitivity terminal information for the specified terminal. This terminal can be a multi-bit terminal.

Arguments

<i>d_termId</i>	Database ID of the specified terminal.
<i>g_all</i>	A variable indicating whether the database IDs of the ground sensitivity terminals of all the bits are returned. The default is <i>nil</i> .

Value Returned

<i>d_groundTermId</i> <i>l_groundTermId</i>	If <i>g_all</i> is <i>nil</i> or not specified, the return value is database ID of the ground sensitivity terminal on the majority of the bits of the specified terminal. If <i>g_all</i> is not <i>nil</i> , the return value is a list where each element of the list is the database ID of the ground sensitivity terminal for the corresponding bit in the specified terminal.
<i>nil</i>	Most or all the bits of a multi-bit terminal do not have a ground sensitivity terminal set.

Examples

Returns the database ID of the ground sensitivity terminal on the majority of the bits of the terminal *busTerm*.

```
dbGetAnyTermGroundSensitivity(busTerm)  
> db:0x256adf1c
```


Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Returns the list of database IDs of all ground sensitivity terminals for the corresponding bit in the terminal `busTerm`.

```
dbGetAnyTermGroundSensitivity(busTerm 'all)  
> (db:0x256adf1c db:0x256adf1c db:0x256adf1c db:0x256adf1e)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbGetAnyTermSupplySensitivity

```
dbGetAnyTermSupplySensitivity(  
    d_termId  
    [ g_all ]  
)  
=> d_supplyTermId | l_supplyTermId / nil
```

Description

Returns the supply sensitivity terminal information for the specified terminal. The terminal can be a multi-bit terminal.

Arguments

<i>d_termId</i>	Database ID of the specified terminal.
<i>g_all</i>	A variable indicating whether the database IDs of the supply sensitivity terminals of all the bits are returned. The default is <i>nil</i> .

Value Returned

<i>d_supplyTermId l_supplyTermId</i>	If <i>g_all</i> is <i>nil</i> or not specified, the return value is database ID of the supply sensitivity terminal on the majority of the bits of the specified terminal. If <i>g_all</i> is not <i>nil</i> , the return value is a list where each element of the list is the database ID of the supply sensitivity terminal for the corresponding bit in the specified terminal.
<i>nil</i>	All or most of the bits of a multi-bit terminal do not have a supply sensitivity term set.

Examples

Returns the database ID of the supply sensitivity terminal on the majority of the bits of the terminal *busTerm*.

```
dbGetAnyTermSupplySensitivity(busTerm)  
> db:0x256adf1c
```

Virtuoso Studio Design Environment SKILL Reference

Connectivity Database Access Functions

Returns the list of database IDs of all supply sensitivity terminals for the corresponding bit in the terminal `busTerm`.

```
dbGetAnyTermSupplySensitivity(busTerm 'all')  
> (db:0x256adf1c db:0x256adf1c db:0x256adf1c db:0x256adf1e)
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbIsAnyTermGroundSensitivityConsistent

```
dbIsAnyTermGroundSensitivityConsistent(  
    d_termId  
)  
=> t / nil
```

Description

Checks whether all the bits of the specified terminal have the same ground sensitivity terminal.

Argument

<i>d_termId</i>	Database ID of the specified terminal.

Value Returned

t	All the bits of the specified terminal have the same ground sensitivity terminal.
nil	The bits of the specified terminal do not all have the same ground sensitivity terminal.

Example

Checks whether all the bits of the terminal `term` have the same ground sensitivity terminal.

```
dbIsAnyTermGroundSensitivityConsistent(term)  
> t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbIsAnyTermSupplySensitivityConsistent

```
dbIsAnyTermSupplySensitivityConsistent(  
    d_termId  
)  
=> t / nil
```

Description

Checks whether all the bits of the specified terminal have the same supply sensitivity terminal.

Argument

<i>d_termId</i>	Database ID of the specified terminal.

Value Returned

<i>t</i>	All the bits of the specified terminal have the same supply sensitivity terminal.
<i>nil</i>	The bits of the specified terminal do not all have the same supply sensitivity terminal.

Example

Checks whether all the bits of the terminal `term` have the same supply sensitivity terminal.

```
dbIsAnyTermSupplySensitivityConsistent(term)  
> t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbSetAnyTermGroundSensitivity

```
dbSetAnyTermGroundSensitivity(  
    d_termId  
    d_groundTermId  
)  
=> t / nil
```

Description

Sets the ground sensitivity value for each bit of the specified terminal. If the terminal is a multi-bit terminal, the function only sets the ground sensitivity value on implicit bits. The ground sensitivity values on explicit bits are not affected.

Arguments

<i>d_termId</i>	Database ID of the specified terminal.
<i>d_groundTermId</i>	Database ID of the ground sensitivity terminal.

Value Returned

<i>t</i>	The ground sensitivity value is set for the specified terminal.
<i>nil</i>	The ground sensitivity value is could not be set for the specified terminal. This may be because the terminal IDs are invalid, or the ground terminal ID is not a single-bit terminal, or the ground sensitivity terminal and the specified terminal are not in the same cellview.

Example

Sets the ground sensitivity value *groundTerm* for each bit of the terminal *term*.

```
dbSetAnyTermGroundSensitivity(term groundTerm)  
> t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

dbSetAnyTermSupplySensitivity

```
dbSetAnyTermSupplySensitivity(  
    d_termId  
    d_suppplyTermId  
)  
=> t / nil
```

Description

Sets the supply sensitivity value for each bit of the specified terminal. If the terminal is a multi-bit terminal, the function only sets the supply sensitivity value on implicit bits. The supply sensitivity value on explicit bits are not affected.

Arguments

<i>d_termId</i>	Database ID of the specified terminal.
<i>d_suppplyTermId</i>	Database ID of the supply sensitivity terminal.

Value Returned

<i>t</i>	The supply sensitivity value is successfully set for the specified terminal.
<i>nil</i>	The supply sensitivity value is not successfully set for the specified terminal. This may be because terminal IDs are invalid, or the supply terminal ID is not a single-bit terminal, or the supply sensitivity terminal and the specified terminal are not in the same cellview.

Example

Sets the supply sensitivity value `supplyTerm` for each bit of the terminal `term`.

```
dbSetAnyTermSupplySensitivity(term supplyTerm)  
> t
```

Related Topics

[Connectivity Creation and Modification Functions](#)

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](#)

Related Topics

[Connectivity Creation and Modification Functions](#)

[Connectivity Access Functions](#)

dbCellViewHasEquivalentConnectivityTime

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

Description

Checks whether the equivalent connectivity time is set on the specified cellview.

Arguments

<i>d_cvId</i>	Database ID of a cellview.

Value Returned

<i>t</i>	A valid equivalent connectivity time is set on the specified cellview.
<i>nil</i>	Equivalent connectivity time is not set on the specified cellview.

Example

Confirms that the equivalent connectivity time is not set on the cellview *cvId*.

```
dbCellViewHasEquivalentConnectivityTime(cvId) => nil
```

Related Topics

[Equivalent Connectivity Time Functions](#)

dbGetCellViewEquivalentConnectivityTime

```
dbGetCellViewEquivalentConnectivityTime  
    d_cvId  
    )  
    t_timeString / nil
```

Description

Retrieves the equivalent connectivity time that is set on the specified cellview.

Arguments

<i>d_cvId</i>	Database ID of a cellview

Value Returned

<i>t_timeString</i>	Equivalent connectivity time that is set on the cellview.
<i>nil</i>	No equivalent connectivity time is set.

Example

Equivalent connectivity time Feb 17 12:01:27 2016 is returned for the cellview *cvId*.

```
dbGetCellViewEquivalentConnectivityTime(cvId) => "Feb 17 12:01:27 2016"
```

Related Topics

[Equivalent Connectivity Time Functions](#)

dbSetCellViewEquivalentConnectivityTime

```
dbSetCellViewEquivalentConnectivityTime
    d_cvId
    t_timestring
)
=> t / nil
```

Description

Sets the equivalent connectivity time on the specified cellview.

Arguments

<i>d_cvId</i>	Database ID of a cellview
<i>t_timeString</i>	Equivalent connectivity time to be set on the cellview.

Value Returned

<i>t</i>	Equivalent connectivity time is set on the specified cellview
<i>nil</i>	Failed to set the equivalent connectivity time.

Example

Equivalent connectivity time *timeVal* is set on the cellview *cvId*.

```
dbSetCellViewEquivalentConnectivityTime(cvId, timeVal) => t
```

Related Topics

[Equivalent Connectivity Time Functions](#)

dbUnsetCellViewEquivalentConnectivityTime

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

Description

Removes equivalent connectivity time information from the specified cellview.

Arguments

<i>d_cvId</i>	Database ID of a cellview

Value Returned

<i>t</i>	Equivalent connectivity time is removed from the specified cellview
<i>nil</i>	Failed to remove the equivalent connectivity time setting.

Example

Equivalent connectivity time information is removed from the cellview *cvId*.

```
dbUnsetCellViewEquivalentConnectivityTime(cvId) => t
```

Related Topics

[Equivalent Connectivity Time Functions](#)

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](#)

Related Topics

[Connectivity Creation and Modification Functions](#)

[Equivalent Connectivity Time Functions](#)

dbDeleteEmptyNet

```
dbDeleteEmptyNet (  
    d_netId  
)  
=> t / nil
```

Description

Deletes a net that is not associated with other objects.

Arguments

<i>d_netId</i>	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)
```

Related Topics

[Connectivity Access Functions](#)

dbFindNetByName

```
dbFindNetByName(  
    d_cellView  
    t_name  
)  
=> d_net / nil
```

Description

Retrieves a net in a cellview.

Arguments

<i>d_cellView</i>	Cellview within which the net is located.
<i>t_name</i>	Name of the net.

Value Returned

<i>d_net</i>	The net named <i>t_name</i> if found.
<i>nil</i>	The net is not retrieved.

Example

```
net = dbFindNetByName(cellview "FindMe")
```

Related Topics

[Connectivity Access Functions](#)

dbFindTermByName

```
dbFindTermByName(  
    d_cellView  
    t_name  
)  
=> d_term / nil
```

Description

Finds a terminal in a cellview.

Arguments

<i>d_cellView</i>	Name of the cellview.
<i>t_name</i>	Name of the terminal to search for.

Value Returned

<i>d_term</i>	The terminal.
<i>nil</i>	The terminal is not found.

Example

```
term = dbFindTermByName(cellview "FindMe")
```

Related Topics

[Connectivity Access Functions](#)

dbFindSigByName

```
dbFindSigByName(  
    d_cellView  
    t_name  
)  
=> d_sig / nil
```

Description

Finds a signal in a cellview.

Arguments

<i>d_cellView</i>	Name of the cellview.
<i>t_name</i>	Name of the signal.

Value Returned

<i>d_sig</i>	The signal found.
<i>nil</i>	The signal is not found.

Example

```
sig = dbFindSigByName(cellview "VDD")
```

Related Topics

[Connectivity Access Functions](#)

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

<i>d_cellView</i>	Cellview the signal is residing within.
<i>t_name</i>	The alias name for the signal.

Value Returned

<i>d_sig</i>	The signal found.
<i>nil</i>	The signal is not found.

Example

```
sig = dbFindSigByAlias(cellview "PWR")
```

Related Topics

[Connectivity Access Functions](#)

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

<i>d_net</i>	A net carrying the signal.
<i>x_index</i>	An integer representing a bit number in the net <i>d_net</i> .

Value Returned

<i>d_sig</i>	The signal carried by the specified member net.
<i>nil</i>	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)
```

Related Topics

[Connectivity Access Functions](#)

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

<i>d_net</i>	A net carrying the signal.
<i>x_index</i>	An integer representing a bit number in the net <i>d_net</i> .

Value Returned

<i>t_name</i>	The name of the signal carried by the specified member net.
<i>nil</i>	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)
```

Related Topics

[Connectivity Access Functions](#)

dbGetNetTerms

```
dbGetNetTerms (
    d_net
)
=> d_term / nil
```

Description

Returns the IDs of the terminals attached to the specified net.

Arguments

<i>d_net</i>	The name of the net.

Value Returned

<i>d_term</i>	The ID of any terminal attached to the net.
<i>nil</i>	No terminals are attached to the net.

Related Topics

[Connectivity Access Functions](#)

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

<i>d_cellView</i>	Cellview where signal <i>t_sigName</i> resides.
<i>t_sigName</i>	The name of the signal.

Value Returned

<i>t_netName x_index</i>	The name of the net and the index representing the net member containing the signal.
nil	signal <i>t_sigName</i> does not exist in cellview <i>d_cellView</i> .

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))
```

Related Topics

[Connectivity Access Functions](#)

dbIsSigNameGlobal

```
dbIsSigNameGlobal(  
    d_cellView  
    t_name  
)  
=> t / nil
```

Description

Tests if the signal *t_name* is a global signal.

Arguments

<i>d_cellView</i>	Cellview where signal <i>t_name</i> resides.
<i>t_name</i>	The name of the signal.

Value Returned

t	signal <i>t_name</i> is global.
nil	The signal is not global.

Example

```
isGlobal = dbIsSigNameGlobal(cellview "VDD!")
```

Related Topics

[Connectivity Access Functions](#)

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

<i>t_name</i>	Name string adhering to the net, terminal, or instance name syntax.
<i>x_index</i>	Integer representing member to retrieve.

Value Returned

<i>t_memName</i>	The name of the <i>x_index</i> member from <i>t_name</i> . In case of an error, the member name returned is composed of <i>t_name</i> suffixed by <i><x_index></i> .

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)
```

Related Topics

Connectivity Access Functions

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 format:

`baseName[<vector-expression>]`

Arguments

<i>t_bundleName</i>	Valid name expression.

Value Returned

<i>l_memNameExpr</i>	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>")
```

Related Topics

[Connectivity Access Functions](#)

dbGetMaxNumBit

```
dbGetMaxNumBit (  
    )  
=> x_numbit
```

Description

Returns the maximum width allowed for a net.

Arguments

None

Value Returned

<i>x_numbit</i>	The maximum net width, expressed as an integer .

Related Topics

[Connectivity Access Functions](#)

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

<i>t_name</i>	Name string adhering to the net, terminal, or instance name syntax.

Value Returned

<i>x_numBit</i>	Number of bits in <i>t_name</i> . 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>")
```

Related Topics

[Connectivity Access Functions](#)

dbMergeSignal

```
dbMergeSignal(  
    d_sig1Id  
    d_sig2Id  
)  
=> d_sig1Id / nil
```

Description

Merges two signals into one signal.

Arguments

<i>d_sig1Id</i>	The database ID of the first signal to be merged.
<i>d_sig2Id</i>	The database ID of the second signal to be merged.

Value Returned

<i>d_sig1Id</i>	Return the database ID of the single merged signal.
<i>nil</i>	An error occurred during execution.

Example

```
dbMergeSignal(sigId1 sigId2)
```

Related Topics

[Connectivity Access Functions](#)

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

<i>d_childId</i>	The database ID of the net you want to be the child.
<i>d_parentId</i>	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)
```

Related Topics

[Connectivity Access Functions](#)

dbMovePinToNet

```
dbMovePinToNet (
    d_pinId
    d_netId
)
=> t / nil
```

Description

Moves a pin to the specified net.

Arguments

<i>d_pinId</i>	The database ID of the pin you want move.
<i>d_netId</i>	The database ID of the net.

Value Returned

t	The pin was successfully moved.
nil	An error occurred during execution.

Example

```
dbMovePinToNet (pinId netId)
```

Related Topics

[Connectivity Access Functions](#)

Pin Database Access Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso Floorplanner.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Pin Connection Functions

dbCreateStrongGroup

dbExternallyConnectPins

dbGetPinConnection

dbGetStrongGroups

dbHasExternalConnections

dbHasMustConnections

dbMergeStrongGroups

dbMustConnectPins

dbStronglyConnectPins

dbTransformAccessDirection

dbWeaklyConnectPins

dbAddFigsToPin

dbAddFigToPin

dbGetPinFigAccessDirection

dbGetPinFigName

dbGetPinFigPlacementStatus

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

[dbSetPinFigAccessDirection](#)

[dbSetPinFigName](#)

[dbSetPinFigPlacementStatus](#)

[dbSetTermMustJoin](#)

[dbSubFigFromPin](#)

[dbSubFigsFromPin](#)

[dbUnsetTermMustJoin](#)

[Pin Group Guide Functions](#)

[dbCreateAreaPinGroupGuide](#)

[dbCreateEdgePinGroupGuide](#)

[dbFindPinGroupGuideByName](#)

[dbGetAreaBoundaryPinGroupGuide](#)

[dbGetAreaPinGroupGuideAreas](#)

[dbGetFigPinGroupGuide](#)

[dbGetPRBoundaryPinGroupGuides](#)

[dbGetPinGroupGuideEdgeBoundary](#)

[dbGetPinGroupGuideEdgeName](#)

[dbGetPinGroupGuideEdgeStartOffset](#)

[dbGetPinGroupGuideEdgeStopOffset](#)

[dbGetPinGroupGuides](#)

[dbGetPinPinGroupGuides](#)

[dbAddAreaToPinGroupGuide](#)

[dbAddFigToPinGroupGuide](#)

[dbAddPinToPinGroupGuide](#)

[dbSubAreaFromPinGroupGuide](#)

[dbSubFigFromPinGroupGuide](#)

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

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

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

dbSetPinGroupGuideMinPinSpacing

dbSetPinGroupGuideMinPinWidth

dbSetPinGroupGuideValidRoutingLayers

dbSetPinMinPinSpacing

dbSetPinMinPinWidth

dbUnsetAreaBoundaryLayerPriorityOrder

dbUnsetAreaBoundaryValidRoutingLayers

dbUnsetPinGroupGuideKeepoutSpacing

dbUnsetPinGroupGuideLayerPriorityOrder

dbUnsetPinGroupGuideMinPinSpacing

dbUnsetPinGroupGuideMinPinWidth

dbUnsetPinGroupGuideValidRoutingLayers

dbUnsetPinMinPinSpacing

dbUnsetPinMinPinWidth

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.



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.

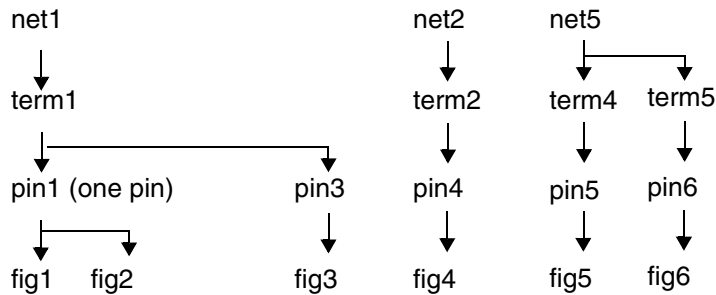
All of the `pinFigs` (shapes) on the same Pin are strongly connected to one another. These shapes represent a set of low resistance connection points to the same logical terminal. The router may chose to route the connection through the strongly connected `pinFigs` at the router's discretion. A typical 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`.

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

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

Use the SKILL functions described in this section to create, add, or remove figures from the pins in a pin connection.

[dbCreateStrongGroup](#)

[dbExternallyConnectPins](#)

[dbGetPinConnection](#)

[dbGetStrongGroups](#)

[dbHasExternalConnections](#)

[dbHasMustConnections](#)

[dbMergeStrongGroups](#)

[dbMustConnectPins](#)

[dbStronglyConnectPins](#)

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

dbTransformAccessDirection

dbWeaklyConnectPins

dbAddFigsToPin

dbAddFigToPin

dbGetPinFigAccessDirection

dbGetPinFigName

dbGetPinFigPlacementStatus

dbSetPinFigAccessDirection

dbSetPinFigName

dbSetPinFigPlacementStatus

dbSetTermMustJoin

dbSubFigFromPin

dbSubFigsFromPin

dbUnsetTermMustJoin

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

Arguments

l_pinList Specifies a list of strings and regular expressions (separated by space and enclosed in double quotes) of the pins on which the function must be run.

Example: "pin1 pin2 I1/p1 pa.* I1/p.*"

s_groupConnection Specifies the group connection for the geometry on the pin.

Value Returned

t The SKILL function was run successfully.

nil The SKILL function could not be run successfully.

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.

Arguments

<i>l_pinList</i>	Specifies a list of strings and regular expressions (separated by space and enclosed in double quotes) of the pins on which the function must be run.

Example: "pin1 pin2 I1/p1 pa.* I1/p.*"

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function could not be run successfully.

Related Topics

[dbSetTermMustJoin](#)

dbGetPinConnection

```
dbGetPinConnection(  
    d_pin1  
    d_pin2  
)  
=> dbcExternallyConnected / dbcWeaklyConnected /  
    dbcStronglyConnected / nil
```

Description

Returns a symbol representing the connectivity between two pins when two pins are given.

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.

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

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.

dbGetStrongGroups

```
dbGetStrongGroups(  
    d_object  
)  
=> ~>name
```

Description

This function is obsolete in this release. Strong connections are modeled by multiple pinFigs on the same pin.

Arguments

<i>d_object</i>	Database ID of an object in the current cellview.

Value Returned

None

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

<code>t</code>	The given net or terminal contains one or more externally connected pins.
<code>nil</code>	The given net or terminal does not contain one or more externally connected pins.

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

<i>anObject</i>	Can be a terminal, a net, or a pin.

Value Returned

t	The given object, <i>anObject</i> , has one or more pins that are must connected. <i>anObject</i> can be a terminal, a net, or a pin. If <i>anObject</i> is a net or a pin, then <i>anObject</i> 's terminal is checked for must connections.

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.

Arguments

<i>l_pinList</i>	Specifies a list of strings and regular expressions (separated by space and enclosed in double quotes) of the pins on which the function must be run.

Example: "pin1 pin2 I1/p1 pa.* I1/p.*"

Value Returned

<i>t</i>	The SKILL function was run successfully.
<i>nil</i>	The SKILL function could not be run successfully.

dbMustConnectPins

```
dbMustConnectPins(  
    l_pinList  
)  
=> t / nil
```

Description

This function is obsolete in this release. External Pin connections are modeled by collections of MustJoin Terms.

Arguments

<i>l_pinList</i>	Specifies a list of strings and regular expressions (separated by space and enclosed in double quotes) of the pins on which the function must be run.

Example: "pin1 pin2 I1/p1 pa.* I1/p.*"

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function could not be run successfully.

Related Topics

[dbSetTermMustJoin](#)

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.

Arguments

<i>l_pinList</i>	Specifies a list of strings and regular expressions (separated by space and enclosed in double quotes) of the pins on which the function must be run.

Example: "pin1 pin2 I1/p1 pa.* I1/p.*"

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function could not be run successfully.

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

<i>x_dirCode</i>	Access direction expressed as an integer. 4 (left), 8 (right), 1 (top), 2 (bottom), any summation of those values, and 0 (none). <i>nil</i> means unspecified and not 0.
<i>t_orient</i>	Specifies the orientation.

Value Returned

<i>x_dirCode</i>	Access direction expressed as an integer. 4 (left), 8 (right), 1 (top), 2 (bottom), any summation of those values, and 0 (none). <i>nil</i> means unspecified and not 0.
<i>nil</i>	Returns <i>nil</i> when the direction is unspecified.

Example

```
dbTransformAccessDirection(4 "R270")
```


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.

Arguments

l_pinList

Specifies a list of strings and regular expressions (separated by space and enclosed in double quotes) of the pins on which the function must be run.

Example: "pin1 pin2 I1/p1 pa.* I1/p.*"

Value Returned

t

The SKILL function was run successfully.

nil

The SKILL function could not be run successfully.

dbAddFigsToPin

```
dbAddFigsToPin(  
    d_pinId  
    ld_figId  
)  
=> t / nil
```

Description

Adds a list of figures to the given pin.

Arguments

<i>d_pinId</i>	The database ID of the pin.
<i>ld_figId</i>	Specifies a list of figure database IDs.

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

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

<i>d_pinId</i>	The database ID of the pin.
<i>d_figId</i>	The database ID of the figure.

Value Returned

<i>t</i>	The SKILL function was run successfully.
<i>nil</i>	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

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

<i>d_figId</i>	The database ID of the figure that is attached to the pin.

Value Returned

<i>t_pinFigAccessDir</i>	A string list indicating the allowed access direction of the pin figure.
<i>nil</i>	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

Example

```
foreach( fig setof(shape geGetEditCellView()~>shapes shape~>pin != nil)  
    printf("Pin fig access direction: %L\n" dbGetPinFigAccessDirection(fig)  
)
```

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

<i>d_figId</i>	The database ID of the figure that is attached to the pin.

Value Returned

<i>t_pinFigName</i>	A string indicating the name of the pin figure.
<i>nil</i>	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

Example

```
foreach( fig setof(shape geGetEditCellView()~>shapes shape~>pin != nil)
    printf("Pin fig name: %s\n" dbGetPinFigName(fig)
    )
)
```

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

<i>d_figId</i>	The database ID of the figure that is attached to the pin.

Value Returned

<i>t_pinFigPlacement</i>	A string list indicating the placement status of the pin figure. Valid values: placed, unplaced, firm, fixed
<i>nil</i>	The SKILL function was not run successfully if the pin ID 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)  
    )  
)
```

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

<i>d_figId</i>	The database ID of the figure that is attached to the pin.
<i>lt_accessDir</i>	A string list which indicates one or more access directions for the specified pin. Valid values: <code>list("left" "right" "bottom" "top")</code>

Value Returned

<i>t</i>	The SKILL function was run successfully.
<i>nil</i>	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

Example

```
dbSetPinFigAccessDirection(figId list("left" "right")) t
```

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

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

<i>d_pinId</i>	The database ID of the figure attached to the pin specified by <i>t_pinName</i> .
<i>t_pinName</i>	The database ID of the pin.

Value Returned

<i>t</i>	The SKILL function was run successfully.
<i>nil</i>	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

Example

```
dbSetPinFigName(figId "MyPin") t
```


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

<i>d_figId</i>	The database ID of the pin figure that is attached to the pin.
<i>g_placementStat</i>	A string list indicating the placement status for the pin figure. Valid values: placed, unplaced, firm, none The default is none.

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

Example

```
dbSetPinFigPlacementStatus(figId "placed") t
```

dbSetTermMustJoin

```
dbSetTermMustJoin(  
    d_termId  
    d_mustJoinTermId  
)  
=> t / nil
```

Description

Sets the connection status of the terminals specified by the terminal IDs to mustJoin.

Arguments

<i>d_termId</i>	The database ID of the terminal that you want to mustJoin to the terminal specified by <i>d_mustJoinTermId</i> .
<i>d_mustJoinTermId</i>	The database ID of the terminal that you want to mustJoin to the terminal specified by <i>d_termId</i> .

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function was not run successfully if either of the terminal IDs are invalid.

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

<i>d_pinId</i>	The database ID of the pin.
<i>d_figId</i>	The database ID of the figure to be removed from the pin.

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

dbSubFigsFromPin

```
dbSubFigsFromPin(  
    d_pinId  
    ld_figId  
)  
=> t / nil
```

Description

Removes the specified list of figures from the given pin.

Arguments

<i>d_pinId</i>	The database ID of the pin.
<i>ld_figId</i>	A list of figure to be removed from a pin.

Value Returned

t	The SKILL function was run successfully.
nil	The SKILL function was not run successfully if the pin ID or figure IDs are invalid.

dbUnsetTermMustJoin

```
dbUnsetTermMustJoin(  
    d_termId  
)  
=> t / nil
```

Description

Removes the mustJoin pin connection status from the given terminal.

Arguments

<i>d_termId</i>	The database ID of the terminal.

Value Returned

<i>t</i>	The SKILL function was run successfully.
<i>nil</i>	The SKILL function was not run successfully if either of the terminal IDs are invalid.

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

<i>d_termId</i>	Database ID of a single-bit terminal.
<i>g_value</i>	Boolean value specifying whether the <code>mustConnectAllPins</code> attribute should be set on the specified terminal. Valid values: ■ <code>t</code>, if the <code>mustConnectAllPins</code> attribute should be set. ■ <code>nil</code>, if the <code>mustConnectAllPins</code> attribute should be not set.

Value Returned

<code>t</code>	The <code>mustConnectAllPins</code> attribute on the specified terminal is set to the given value.
<code>nil</code>	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`.

dbTermHasMustConnectAllPins

```
dbTermHasMustConnectAllPins(  
    d_termId  
)  
=> t / nil
```

Description

Checks whether the `mustConnectAllPins` attribute value is set on the specified single-bit terminal.

Arguments

<code>d_termId</code>	The database ID of a terminal.

Value Returned

<code>t</code>	The <code>mustConnectAllPins</code> attribute value is set on the specified terminal.
<code>nil</code>	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`.

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.

Use the database SKILL functions described in this section to create pin groups and assign pin constraints to them.

[dbCreateAreaPinGroupGuide](#)

[dbCreateEdgePinGroupGuide](#)

[dbFindPinGroupGuideByName](#)

[dbGetAreaBoundaryPinGroupGuide](#)

[dbGetAreaPinGroupGuideAreas](#)

[dbGetFigPinGroupGuide](#)

[dbGetPRBoundaryPinGroupGuides](#)

[dbGetPinGroupGuideEdgeBoundary](#)

[dbGetPinGroupGuideEdgeName](#)

[dbGetPinGroupGuideEdgeStartOffset](#)

[dbGetPinGroupGuideEdgeStopOffset](#)

[dbGetPinGroupGuides](#)

[dbGetPinPinGroupGuides](#)

[dbAddAreaToPinGroupGuide](#)

[dbAddFigToPinGroupGuide](#)

[dbAddPinToPinGroupGuide](#)

[dbSubAreaFromPinGroupGuide](#)

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

dbSubFigFromPinGroupGuide

dbSubPinFromPinGroupGuide

dbRemovePinGroupGuideEdge

dbSetPinGroupGuideEdge

dbSetPinGroupGuideEdgeStartOffset

dbSetPinGroupGuideEdgeStopOffset

dbGetAreaBoundaryValidRoutingLayers

dbAreaBoundaryHasLayerPriorityOrder

dbAreaBoundaryHasValidRoutingLayers

dbGetAreaBoundaryLayerPriorityOrder

dbGetPinGroupGuideKeepoutSpacing

dbGetPinGroupGuideLayerPriorityOrder

dbGetPinGroupGuideMinPinSpacing

dbGetPinGroupGuideMinPinWidth

dbGetPinGroupGuideValidRoutingLayers

dbGetPinMinPinSpacing

dbGetPinMinPinWidth

dbPinGroupGuideHasKeepoutSpacing

dbPinGroupGuideHasLayerPriorityOrder

dbPinGroupGuideHasMinPinSpacing

dbPinGroupGuideHasMinPinWidth

dbPinGroupGuideHasValidRoutingLayers

dbPinHasMinPinSpacing

dbPinHasMinPinWidth

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

[dbSetAreaBoundaryLayerPriorityOrder](#)

[dbSetAreaBoundaryValidRoutingLayers](#)

[dbSetPinGroupGuideKeepoutSpacing](#)

[dbSetPinGroupGuideLayerPriorityOrder](#)

[dbSetPinGroupGuideMinPinSpacing](#)

[dbSetPinGroupGuideMinPinWidth](#)

[dbSetPinGroupGuideValidRoutingLayers](#)

[dbSetPinMinPinSpacing](#)

[dbSetPinMinPinWidth](#)

[dbUnsetAreaBoundaryLayerPriorityOrder](#)

[dbUnsetAreaBoundaryValidRoutingLayers](#)

[dbUnsetPinGroupGuideKeepoutSpacing](#)

[dbUnsetPinGroupGuideLayerPriorityOrder](#)

[dbUnsetPinGroupGuideMinPinSpacing](#)

[dbUnsetPinGroupGuideMinPinWidth](#)

[dbUnsetPinGroupGuideValidRoutingLayers](#)

[dbUnsetPinMinPinSpacing](#)

[dbUnsetPinMinPinWidth](#)

Related Topics

[Creating Pin Group Guides](#)

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

<i>d_cellViewId</i>	The cellview in which the area pin group guide is created.
<i>t_name</i>	The name of the area pin group guide.

Value Returned

<i>d_pinGroupGuideId</i>	The pin group guide ID is created with the specified name in the given cellview.
<i>nil</i>	The pin group guide ID is not created.

Example

```
dbCreateAreaPinGroupGuide(cv "pg1")  
=> db:0x13eaeb1b
```

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

<i>d_cellViewId</i>	The ID of the cellview in which the edge pin group guide is created.
<i>t_name</i>	The name of the edge pin group guide.

Value Returned

<i>d_edgePinGroupGuideId</i>	The edge pin group guide ID is created with the specified name in the given cellview.
<i>nil</i>	The SKILL function was not run successfully 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
```

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

<i>d_cellViewId</i>	The cellview in which the pin group guide is searched.
<i>t_name</i>	The name of the pin group guide.

Value Returned

<i>d_edgePinGroupGuideId</i>	The ID of the pin group guide that is found with the specified name in the given cellview.
<i>nil</i>	The pin group guide ID is not found.

Example

```
dbFindPinGroupGuideByName (cv "pgg1")
```

dbGetAreaBoundaryPinGroupGuide

```
dbGetAreaBoundaryPinGroupGuide (
    d_areaBoundaryId
)
=> d_pinGroupGuideId / nil
```

Description

Returns the pin group guide associated with the specified area boundary.

Arguments

<i>d_areaBoundaryId</i>	The ID of the specified area boundary whose associated pin group guide needs to be returned.

Value Returned

<i>d_pinGroupGuideId</i>	The pin group guide ID of the specified area boundary.
<i>nil</i>	The SKILL function was not run successfully if no pin group guide ID is associated with the specified area boundary.

Example

```
dbGetAreaBoundaryPinGroupGuide (areaBndyId)
=> db:0x1818de1b
```

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

<i>l_areaBoundaryIds</i>	The list of area boundaries associated with the specified area pin group guide.
<i>nil</i>	No area boundaries are associated with the specified area pin group guide.

Example

```
dbGetAreaPinGroupGuideAreas(pinGroupGuideId)  
=> list(db:0x1818daaa db:0x1818daae)
```

dbGetFigPinGroupGuide

```
dbGetFigPinGroupGuide(  
    d_figId  
)  
=> d_pinGroupGuideId / nil
```

Description

Returns the pin group guide associated with the specified figure.

Arguments

<i>d_figId</i>	The ID of the specified figure whose associated pin group guide needs to be returned.

Value Returned

<i>d_pinGroupGuideId</i>	The ID of a pin group guide associated with the specified figure is returned.
<i>nil</i>	The ID of a pin group guide is not returned, if no pin group guide is associated with the specified figure.

Example

```
dbGetFigPinGroupGuide(figId)  
=> db:0x1818de1b
```


dbGetPRBoundaryPinGroupGuides

```
dbGetPRBoundaryPinGroupGuides (
    d_prBoundaryId
)
=> l_pinGroupGuideIds / nil
```

Description

Returns the list of pin group guides associated with the specified prBoundary.

Arguments

<i>d_prBoundaryId</i>	The ID of the specified prBoundary whose associated pin group guides need to be returned.

Value Returned

<i>l_pinGroupGuideIds</i>	The list of the pin group guides associated with the specified prBoundary is returned.
<i>nil</i>	The SKILL function was not run successfully if no pin group guide is associated with the specified prBoundary.

Example

```
dbGetPRBoundaryPinGroupGuides (prBoundaryId)
=> list (db:0x1818df1b db:0x1818de8a)
```

dbGetPinGroupGuideEdgeBoundary

```
dbGetPinGroupGuideEdgeBoundary(  
    d_edgePinGroupGuide  
)  
=> d_prBoundary / nil
```

Description

Returns the prBoundary associated with the specified edge pin group guide.

Arguments

<i>d_edgePinGroupGuide</i>	The specified edge pin group guide whose associated prBoundary need to be returned.

Value Returned

<i>d_prBoundary</i>	The prBoundary associated with the specified edge pin group guide is returned.
nil	The SKILL function was not run successfully if no prBoundary is associated with the specified edge pin group guide.

Example

```
dbGetPinGroupGuideEdgeBoundary(epgId)  
=> db:0x13eaec1a
```

dbGetPinGroupGuideEdgeName

```
dbGetPinGroupGuideEdgeName(  
    d_edgePinGroupGuide  
)  
=> t_name / nil
```

Description

Returns the edge name of the specified edge pin group guide.

Arguments

<i>d_edgePinGroupGuide</i>	The specified edge pin group guide whose edge name needs to be returned.

Value Returned

<i>t_name</i>	The edge name of the specified edge pin group guide is returned.
<i>nil</i>	The SKILL function was not run successfully and returns an error. For example, when an invalid edge pin group guide is specified.

Example

```
dbGetPinGroupGuideEdgeName(epgId)  
=> "Edge1"
```

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

<i>n_offset</i>	The start offset of the edge for the specified edge pin group guide is returned.
<i>nil</i>	The SKILL function was not run successfully and returns an error. For example, when an invalid edge pin group guide is specified.

Example

```
dbGetPinGroupGuideEdgeStartOffset(epgId)  
=> 1
```

dbGetPinGroupGuideEdgeStopOffset

```
dbGetPinGroupGuideEdgeStopOffset(  
    d_edgePinGroupGuide  
)  
=> n_offset / nil
```

Description

Returns the stop offset of the edge for the specified edge pin group guide.

Arguments

<i>d_edgePinGroupGuide</i>	The specified edge pin group guide whose stop offset of the edge needs to be returned.

Value Returned

<i>n_offset</i>	The stop offset of the edge for the specified edge pin group guide is returned.
<i>nil</i>	The SKILL function was not run successfully and returns an error. For example, when an invalid edge pin group guide is specified.

Example

```
dbGetPinGroupGuideEdgeStopOffset(epgId)  
=> 2
```

dbGetPinGroupGuides

```
dbGetPinGroupGuides(  
    d_cellViewId  
)  
=> l_pinGroupGuides / nil
```

Description

Returns the list of pin group guides associated with the specified cellview.

Arguments

<i>d_cellViewId</i>	The ID of the cellview whose associated pin group guides need to be returned.

Value Returned

<i>l_pinGroupGuides</i>	The list of pin group guides associated with the specified cellview is returned.
<i>nil</i>	The SKILL function was not run successfully if no pin group guides are associated with the specified cellview.

Example

```
dbGetPinGroupGuides(cvId)  
=> list(db:0x1818df1b db:0x1818de8a)
```

dbGetPinPinGroupGuides

```
dbGetPinPinGroupGuides(  
    d_pinId  
)  
=> l_pinGroupGuideIds / nil
```

Description

Returns the list of pin group guides associated with the specified pin.

Arguments

<i>d_pinId</i>	The ID of the pin whose associated pin group guides need to be returned.

Value Returned

<i>l_pinGroupGuideIds</i>	The list of pin group guides associated with the specified pin is returned.
<i>nil</i>	The SKILL function was not run successfully and returns <i>nil</i> if no pin group guides are associated with the specified pin.

Example

```
dbGetPinPinGroupGuides(pinId)  
=> list(db:0x1818df1b db:0x1818de8a)
```

dbAddAreaToPinGroupGuide

```
dbAddAreaToPinGroupGuide(  
    d_areaPinGroupGuideId  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

Adds an area boundary to the specified area pin group guide.

Arguments

<i>d_areaPinGroupGuideId</i>	The specified area pin group guide to which the area boundary needs to be added.
<i>d_areaBoundaryId</i>	The ID of the specified area boundary.

Value Returned

<i>t</i>	The SKILL function was run successfully if an area boundary is added to the specified area pin group guide.
<i>nil</i>	The SKILL function was not run successfully and returns <i>nil</i> if an area boundary was not added to the specified area pin group guide.

Example

```
dbAddAreaToPinGroupGuide(apgId areaBndyId)  
=> t
```


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

<i>d_pinGroupGuideId</i>	The specified pin group guide.
<i>d_figId</i>	The via or non-text shape to be added.

Value Returned

<i>t</i>	The SKILL function runs successfully if the given figure is added to the pin group guide.
<i>nil</i>	The SKILL function was not run successfully and returns <i>nil</i> 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)
```

dbAddPinToPinGroupGuide

```
dbAddPinToPinGroupGuide(  
    d_pinGroupGuideId  
    d_pinId  
)  
=> t / nil
```

Description

This function adds a pin to the specified pin group guide.

Arguments

<i>d_pinGroupGuideId</i>	The database ID of the pin group guide.
<i>d_pinId</i>	The pin to be added.

Value Returned

<i>t</i>	The SKILL function was run successfully if the pin is added to the specified pin group guide.
<i>nil</i>	The SKILL function was not run successfully and returns <i>nil</i> on error.

Example

```
dbAddPinToPinGroupGuide(epg1 pin)
```

dbSubAreaFromPinGroupGuide

```
dbSubAreaFromPinGroupGuide(  
    d_areaPinGroupGuideId  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

Removes an area boundary from the specified area pin group guide.

Arguments

<i>d_areaPinGroupGuideId</i>	The ID of the area pin group guide.
<i>d_areaBoundaryId</i>	The ID of the area boundary to be removed.

Value Returned

<i>t</i>	The SKILL function was run successfully and returns <i>t</i> if the area boundary is removed from the specified area pin group guide.
<i>nil</i>	The SKILL function was not run successfully and returns <i>nil</i> on error.

Example

```
dbSubAreaFromPinGroupGuide(apgId areaBndyId) => t
```

dbSubFigFromPinGroupGuide

```
dbSubFigFromPinGroupGuide(  
    d_pinGroupGuideId  
    d_figId  
)  
=> t / nil
```

Description

Removes a via or a non-text shape from the specified pin group guide.

Arguments

<i>d_pinGroupGuideId</i>	The ID of the pin group guide.
<i>d_figId</i>	The ID of the via or non-text shape to be removed.

Value Returned

<i>t</i>	The SKILL function was run successfully if the via or the non-text shape is removed from the specified pin group guide.
<i>nil</i>	The SKILL function was not run successfully when the via or the non-text shape is not removed from the specified pin group guide.

Example

```
dbSubFigFromPinGroupGuide(epgl pinFig) => t
```

dbSubPinFromPinGroupGuide

```
dbSubPinFromPinGroupGuide(  
    d_pinGroupGuideId  
    d_pinId  
)  
=> t / nil
```

Description

Removes a pin from the specified pin group guide.

Arguments

<i>d_pinGroupGuideId</i>	The ID of the pin group guide.
<i>d_pinId</i>	The ID of the pin to be removed.

Value Returned

<i>t</i>	Returns <i>t</i> if the pin is removed from the specified pin group guide.
<i>nil</i>	Returns <i>nil</i> if the pin is not removed from the specified pin group guide.

Example

```
dbSubPinFromPinGroupGuide(epg1 pin)  
=> t
```

dbRemovePinGroupGuideEdge

```
dbRemovePinGroupGuideEdge (
    d_edgePinGroupGuide
)
=> t / nil
```

Description

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

<i>t</i>	Returns <i>t</i> if the prBoundary is removed from the given edge pin group guide.
<i>nil</i>	Returns <i>nil</i> in case of error, that is, when the prBoundary is not removed successfully.

Example

```
dbRemovePinGroupGuideEdge (epgId)
=> t
```

dbSetPinGroupGuideEdge

```
dbSetPinGroupGuideEdge(  
    d_edgePinGroupGuide  
    d_prBoundary  
    t_name  
)  
=> t / nil
```

Description

Specifies new or updates existing prBoundary edge information for the edge pin group guide.

Arguments

<i>d_edgePinGroupGuide</i>	Specifies the ID of the edge pin group guide name.
<i>d_prBoundary</i>	Specifies the ID of the prBoundary.
<i>t_name</i>	Specifies the edge name of the prBoundary that is being added. prBoundaries can have multiple edge names.

Value Returned

<i>t</i>	Returns <i>t</i> if the prBoundary edge specification for the given edge pin group guide is set or updated.
<i>nil</i>	Returns <i>nil</i> if the prBoundary edge specification for the given edge pin group guide is not set or updated.

Example

```
dbSetPinGroupGuideEdge(epgId prbId "Edge1") => t
```

dbSetPinGroupGuideEdgeStartOffset

```
dbSetPinGroupGuideEdgeStartOffset(  
    d_edgePinGroupGuide  
    n_offset  
)  
=> t / nil
```

Description

Sets the start offset of the edge on the edge pin group guide.

Arguments

<i>d_edgePinGroupGuide</i>	The ID of the edge pin group guide.
<i>n_offset</i>	The start offset of the edge. The default offset value is 0.

Value Returned

<i>t</i>	Returns <i>t</i> if the start offset of the edge on the edge pin group guide is set.
<i>nil</i>	Returns <i>nil</i> if the start offset of the edge on the edge pin group guide is not set.

Example

```
dbSetPinGroupGuideEdgeStartOffset(epgId 5)  
=> t
```


dbSetPinGroupGuideEdgeStopOffset

```
dbSetPinGroupGuideEdgeStopOffset(  
    d_edgePinGroupGuide  
    n_offset  
)  
=> t / nil
```

Description

Sets the stop offset of the edge on the edge pin group guide.

Arguments

<i>d_edgePinGroupGuide</i>	The ID of the edge pin group guide.
<i>n_offset</i>	The stop offset of the edge.

Value Returned

<i>t</i>	Returns <i>t</i> if the stop offset of the edge on the edge pin group guide is set.
<i>nil</i>	Returns <i>nil</i> if the stop offset of the edge on the edge pin group guide is not set.

Example

```
dbSetPinGroupGuideEdgeStopOffset(epgId 10)  
=> t
```

dbGetAreaBoundaryValidRoutingLayers

```
dbGetAreaBoundaryValidRoutingLayers(  
    d_areaBoundaryId  
)  
=> l_layers / nil
```

Description

Returns the value of the `validRoutingLayers` constraint on the given area boundary.

Arguments

<code>d_areaBoundaryId</code>	The ID of the area boundary.

Value Returned

<code>l_layers</code>	Returns the value of the <code>validRoutingLayers</code> constraint if it is set on the specified area boundary.
<code>nil</code>	Returns <code>nil</code> if no <code>validRoutingLayers</code> constraint is set on the specified area boundary.

Example

```
dbGetAreaBoundaryValidRoutingLayers(areaBndyId)  
=> ("M1" "M2" "VIA2" "VIA3" "M6")
```

dbAreaBoundaryHasLayerPriorityOrder

```
dbAreaBoundaryHasLayerPriorityOrder(  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

Returns a boolean value indicating that the `layerPriorityOrder` constraints set on the specified area boundary.

Arguments

<i>d_areaBoundaryId</i>	The ID of the area boundary.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>layerPriorityOrder</code> constraint is set on the specified area boundary.
<code>nil</code>	Returns <code>nil</code> if the <code>layerPriorityOrder</code> constraint is not set on the specified area boundary.

Example

```
dbAreaBoundaryHasLayerPriorityOrder(areaBndyId)  
=> t
```

dbAreaBoundaryHasValidRoutingLayers

```
dbAreaBoundaryHasValidRoutingLayers(  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

Returns a boolean value indicating that the `validRoutingLayers` constraint is set on the specified area boundary.

Arguments

<code>d_areaBoundaryId</code>	The ID of the area boundary.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>validRoutingLayers</code> constraint is set on the specified area boundary.
<code>nil</code>	Returns <code>nil</code> if the <code>validRoutingLayers</code> constraint is not set on the specified area boundary.

Example

```
dbAreaBoundaryHasValidRoutingLayers(areaBndyId)  
=> t
```

dbGetAreaBoundaryLayerPriorityOrder

```
dbGetAreaBoundaryLayerPriorityOrder(  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

Returns the value of the `layerPriorityOrder` constraint that is set on the specified area boundary.

Arguments

<code>d_areaBoundaryId</code>	The ID of the area boundary.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>layerPriorityOrder</code> constraint is set on the specified area boundary.
<code>nil</code>	Returns <code>nil</code> if the <code>layerPriorityOrder</code> constraint is not set on the specified area boundary.

Example

```
dbGetAreaBoundaryLayerPriorityOrder(areaBndyId)  
=> t
```

dbGetPinGroupGuideKeepoutSpacing

```
dbGetPinGroupGuideKeepoutSpacing(  
    d_pinGroupGuide  
    tx_layer  
)  
=> n_value / nil
```

Description

Returns the value of the `keepoutSpacing` constraint set on the specified pin group guide.

Arguments

<i>d_pinGroupGuide</i>	The ID for the pin group guide name.
<i>tx_layer</i>	The layer name or layer number.

Value Returned

<i>n_value</i>	Returns the value of the <code>keepoutSpacing</code> constraint if it is set on the specified pin group guide.
<i>nil</i>	Returns <code>nil</code> if the <code>keepoutSpacing</code> constraint is not set on the specified pin group guide.

Example

```
dbGetPinGroupGuideKeepoutSpacing(epg1 0)  
=> 6
```

dbGetPinGroupGuideLayerPriorityOrder

```
dbGetPinGroupGuideLayerPriorityOrder(  
    d_areaPinGroupGuide  
)  
=> t / nil
```

Description

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

<code>t</code>	Returns <code>t</code> if the <code>layerPriorityOrder</code> constraint is set on the specified area pin group guide.
<code>nil</code>	Returns <code>nil</code> if the <code>layerPriorityOrder</code> constraint is not set on the specified area pin group guide.

Example

```
dbGetPinGroupGuideLayerPriorityOrder(apgId) => t
```

dbGetPinGroupGuideMinPinSpacing

```
dbGetPinGroupGuideMinPinSpacing(  
    d_pinGroupGuide  
    tx_layer  
)  
=> n_value / nil
```

Description

Returns the value of the `minSpacing` constraint set on the specified pin group guide.

Arguments

<i>d_pinGroupGuide</i>	The ID of the pin group guide from which the <code>minSpacing</code> constraint value is to be retrieved.
<i>tx_layer</i>	The layer name or number in the pin group from which the value of the <code>minSpacing</code> constraint is to be retrieved.

Value Returned

<i>n_value</i>	Returns the value of the <code>minSpacing</code> constraint if it is set on the specified pin group guide.
<i>nil</i>	Returns <code>nil</code> if the <code>minSpacing</code> constraint is not set on the specified pin group guide.

Example

```
dbGetPinGroupGuideMinPinSpacing(epg 0) => 2
```


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

<i>d_pinGroupGuide</i>	The ID of the pin group guide from which the value of the <code>minWidth</code> constraint is to be retrieved.
<i>tx_layer</i>	The layer name or number in the pin group from which the value of the <code>minWidth</code> constraint is to be retrieved.

Value Returned

<i>n_value</i>	Returns the value of the <code>minWidth</code> constraint if it is set on the specified pin group guide.
<i>nil</i>	Returns <code>nil</code> if the <code>minWidth</code> constraint is not set on the specified pin group guide.

Example

```
dbGetPinGroupGuideMinPinWidth(epg 0) => 3
```

dbGetPinGroupGuideValidRoutingLayers

```
dbGetPinGroupGuideValidRoutingLayers(  
    d_areaPinGroupGuide  
)  
=> l_layers / nil
```

Description

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

<i>l_layers</i>	Returns the value of the <code>validRoutingLayers</code> constraint if it is set on the specified area pin group guide.
<i>nil</i>	Returns <code>nil</code> if the <code>validRoutingLayers</code> constraint is not set on the specified area pin group guide.

Example

```
dbGetPinGroupGuideValidRoutingLayers(apgId) => ("M1" "M2" "VIA2" "VIA3" "M6")
```

dbGetPinMinPinSpacing

```
dbGetPinMinPinSpacing(  
    d_pinId  
    tx_layer  
)  
=> n_value / nil
```

Description

Returns the value of the `minSpacing` constraint set on the specified pin.

Arguments

<code>d_pinId</code>	The ID of the pin.
<code>tx_layer</code>	The layer name or number in the pin group from which the value of the <code>minSpacing</code> constraint is to be retrieved.

Value Returned

<code>n_value</code>	Returns the value of the <code>minSpacing</code> constraint if it is set on the specified pin.
<code>nil</code>	Returns <code>nil</code> if the <code>minSpacing</code> constraint is not set on the specified pin.

Example

```
dbGetPinMinPinSpacing(pinId 0) => 6
```

dbGetPinMinPinWidth

```
dbGetPinMinPinWidth(  
    d_pinId  
    tx_layer  
)  
=> n_value / nil
```

Description

Returns the value of the `minWidth` constraint set on the specified pin.

Arguments

<code>d_pinId</code>	The ID of the pin.
<code>tx_layer</code>	The layer name or number in the pin group from which the value of the <code>minWidth</code> constraint is to be retrieved.

Value Returned

<code>n_value</code>	Returns the value of the <code>minWidth</code> constraint if it is set on the specified pin.
<code>nil</code>	Returns <code>nil</code> if the <code>minWidth</code> constraint is not set on the specified pin.

Example

```
dbGetPinMinPinWidth(pinId 0) => 8
```

dbPinGroupGuideHasKeepoutSpacing

```
dbPinGroupGuideHasKeepoutSpacing(  
    d_pinGroupGuide  
    tx_layer  
)  
=> t / nil
```

Description

Returns a boolean value indicating that the `keepoutSpacing` constraint is set on the specified pin group guide.

Arguments

<i>d_pinGroupGuide</i>	Specifies the ID of the pin group guide.
<i>tx_layer</i>	The layer name or number in the pin group from which the value of the <code>keepoutSpacing</code> constraint is to be retrieved.

Value Returned

<i>t</i>	Returns <i>t</i> if the <code>keepoutSpacing</code> constraint is set on the specified pin group guide.
<i>nil</i>	Returns <i>nil</i> if the <code>keepoutSpacing</code> constraint is not set on the specified pin group guide.

Example

```
dbPinGroupGuideHasKeepoutSpacing(epgl 0) => t
```

dbPinGroupGuideHasLayerPriorityOrder

```
dbPinGroupGuideHasLayerPriorityOrder(  
    d_areaPinGroupGuide  
)  
=> t / nil
```

Description

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

<code>t</code>	Returns <code>t</code> if the <code>layerPriorityOrder</code> constraint is set on the specified area pin group guide.
<code>nil</code>	Returns <code>nil</code> if the <code>layerPriorityOrder</code> constraint is not set on the specified area pin group guide.

Example

```
dbPinGroupGuideHasLayerPriorityOrder(apgId) => t
```

dbPinGroupGuideHasMinPinSpacing

```
dbPinGroupGuideHasMinPinSpacing(  
    d_PinGroupGuide  
    tx_layer  
)  
=> t / nil
```

Description

Returns a boolean value indicating whether the `minSpacing` constraint is set on the specified pin group guide.

Arguments

<i>d_PinGroupGuide</i>	The ID of the pin group guide.
<i>tx_layer</i>	The layer name or number in the pin group from which the value of the <code>minSpacing</code> constraint is to be retrieved.

Value Returned

<i>t</i>	Returns <i>t</i> if the <code>minSpacing</code> constraint is set on the specified pin group guide.
<i>nil</i>	Returns <i>nil</i> if the <code>minSpacing</code> constraint is not set on the specified pin group guide.

Example

```
dbPinGroupGuideHasMinPinSpacing(epg1 0) => t
```

dbPinGroupGuideHasMinPinWidth

```
dbPinGroupGuideHasMinPinWidth(  
    d_PinGroupGuide  
    tx_layer  
)  
=> t / nil
```

Description

Returns a boolean value indicating whether the `minWidth` constraint is set on the specified pin group guide.

Arguments

<code>d_PinGroupGuide</code>	The ID of the pin group guide.
<code>tx_layer</code>	The layer name or number in the pin group from which the value of the <code>minWidth</code> constraint is to be retrieved.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>minWidth</code> constraint is set on the specified pin group guide.
<code>nil</code>	Returns <code>nil</code> if the <code>minWidth</code> constraint is not set on the specified pin group guide.

Example

```
dbPinGroupGuideHasMinPinWidth(epg1 0) => t
```


dbPinGroupGuideHasValidRoutingLayers

```
dbPinGroupGuideHasValidRoutingLayers(  
    d_areaPinGroupGuide  
)  
=> t / nil
```

Description

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

<code>t</code>	Returns <code>t</code> if the <code>validRoutingLayers</code> constraint is set on the specified area pin group guide.
<code>nil</code>	Returns <code>nil</code> if the <code>validRoutingLayers</code> constraint is not set on the specified area pin group guide.

Example

```
dbPinGroupGuideHasValidRoutingLayers(apgId) => t
```

dbPinHasMinPinSpacing

```
dbPinHasMinPinSpacing(  
    d_PinId  
    tx_layer  
)  
=> t / nil
```

Description

Returns a boolean value indicating whether the `minSpacing` constraint is set on the specified pin.

Arguments

<code>d_PinId</code>	The ID of the pin.
<code>tx_layer</code>	The layer name or number in the pin group from which the value of the <code>minSpacing</code> constraint is to be retrieved.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>minSpacing</code> constraint is set on the specified pin.
<code>nil</code>	Returns <code>nil</code> if the <code>minSpacing</code> constraint is not set on the specified pin.

Example

```
dbPinHasMinPinSpacing(pinId 0) => t
```

dbPinHasMinPinWidth

```
dbPinHasMinPinWidth(  
    d_pinId  
    tx_layer  
)  
=> t / nil
```

Description

Returns a boolean value indicating whether the `minWidth` constraint has been applied to the specified pin.

Arguments

<code>d_pinId</code>	The database ID of the pin.
<code>tx_layer</code>	The layer name or the layer number of the specified pin.

Value Returned

<code>t</code>	Returns <code>t</code> when the <code>minWidth</code> constraint has been applied to the specified pin.
<code>nil</code>	Returns <code>nil</code> if the <code>minWidth</code> constraint has not been applied to the specified pin or if the pin ID is invalid.

Example

```
dbPinHasMinPinWidth(pinId 0) => t
```

dbSetAreaBoundaryLayerPriorityOrder

```
dbSetAreaBoundaryLayerPriorityOrder(  
    d_areaBoundaryId  
    g_value  
)  
=> t / nil
```

Description

Sets the value of the `layerPriorityOrder` constraint on the given area boundary.

Arguments

<code>d_areaBoundaryId</code>	The database ID of the area boundary.
<code>g_value</code>	The value of the <code>layerPriorityOrder</code> constraint that you want to set.

Value Returned

<code>t</code>	Returns <code>t</code> if the value of the <code>layerPriorityOrder</code> constraint is successfully set.
<code>nil</code>	Returns <code>nil</code> if the value of the <code>layerPriorityOrder</code> constraint cannot be set successfully.

Example

```
dbSetAreaBoundaryLayerPriorityOrder(areaBndyId t) => t
```

dbSetAreaBoundaryValidRoutingLayers

```
dbSetAreaBoundaryValidRoutingLayers(  
    d_areaBoundaryId  
    l_layers  
)  
=> t / nil
```

Description

Sets the value of the `validRoutingLayers` constraint on the given area boundary.

Arguments

<i>d_areaBoundaryId</i>	The ID of the area boundary.
<i>l_layers</i>	The list of valid routing layers.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>validRoutingLayers</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>validRoutingLayers</code> constraint cannot be set successfully.

Example

```
dbSetAreaBoundaryValidRoutingLayers(areaBndyId list("M1" "M2" "VIA2" "VIA3" "M6"))  
=> t
```

dbSetPinGroupGuideKeepoutSpacing

```
dbSetPinGroupGuideKeepoutSpacing(  
    d_pinGroupGuide  
    tx_layer  
    n_value  
)  
=> t / nil
```

Description

Sets the value of the `keepoutSpacing` constraint on the given pin group guide.

Arguments

<i>d_pinGroupGuide</i>	The ID of the pin group guide.
<i>tx_layer</i>	The layer number or layer name.
<i>n_value</i>	The value that you want to set for the constraint.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>keepoutSpacing</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>keepoutSpacing</code> constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideKeepoutSpacing(epg1 0 1)  
=> t
```

dbSetPinGroupGuideLayerPriorityOrder

```
dbSetPinGroupGuideLayerPriorityOrder(  
    d_areaPinGroupGuide  
    g_value  
)  
=> t / nil
```

Description

Sets the value of the `layerPriorityOrder` constraint on the given area pin group guide.

Arguments

<i>d_areaPinGroupGuide</i>	The ID of the area pin group guide.
<i>g_value</i>	The value that you want to set for the constraint.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>layerPriorityOrder</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>layerPriorityOrder</code> constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideLayerPriorityOrder(apgId t)  
=> t
```

dbSetPinGroupGuideMinPinSpacing

```
dbSetPinGroupGuideMinPinSpacing(  
    d_pinGroupGuide  
    tx_layer  
    n_value  
)  
=> t / nil
```

Description

Sets the value of the `minSpacing` constraint on the given pin group guide.

Arguments

<i>d_PinGroupGuide</i>	The ID of the pin group guide.
<i>tx_layer</i>	The layer name or layer number.
<i>n_value</i>	The integer value that you want to set for the <code>minSpacing</code> constraint.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>minSpacing</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>minSpacing</code> constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideMinPinSpacing(epg 0 2)  
=> t
```


dbSetPinGroupGuideMinPinWidth

```
dbSetPinGroupGuideMinPinSpacing(  
    d_pinGroupGuide  
    tx_layer  
    n_value  
)  
=> t / nil
```

Description

Sets the value of the `minWidth` constraint on the given pin group guide.

Arguments

<i>d_PinGroupGuide</i>	The ID of the pin group guide.
<i>tx_layer</i>	The layer name or layer number.
<i>n_value</i>	The integer value that you want to set for the <code>minWidth</code> constraint.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>minWidth</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>minWidth</code> constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideMinPinWidth(epg 0 3)  
=> t
```

dbSetPinGroupGuideValidRoutingLayers

```
dbSetPinGroupGuideValidRoutingLayers(  
    d_areapinGroupGuide  
    l_layers  
)  
=> t / nil
```

Description

Sets the value of the `validRoutingLayers` constraint on the given area pin group guide.

Arguments

<i>d_areaPinGroupGuide</i>	The ID of the area pin group guide.
<i>l_layers</i>	A list of valid routing layers.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>validRoutingLayers</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>validRoutingLayers</code> constraint cannot be set successfully.

Example

```
dbSetPinGroupGuideValidRoutingLayers(apgId list("M1" 2 3 "VIA3" "M6"))  
=> t
```

dbSetPinMinPinSpacing

```
dbSetPinMinPinSpacing(  
    d_pinID  
    tx_layer  
    n_value  
)  
=> t / nil
```

Description

Sets the value of the `minSpacing` constraint on the given pin.

Arguments

<i>d_pinID</i>	The ID of the pin.
<i>tx_layer</i>	The layer name or the layer number.
<i>n_value</i>	The integer value of the constraint that you want to set.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>minSpacing</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>minSpacing</code> constraint cannot be set successfully.

Example

```
dbSetPinMinPinSpacing(pinId 0 5)  
=> t
```

dbSetPinMinPinWidth

```
dbSetPinMinPinWidth(  
    d_pinID  
    tx_layer  
    n_value  
)  
=> t / nil
```

Description

Sets the value of the `minWidth` constraint on the given pin.

Arguments

<i>d_pinID</i>	The ID of the pin.
<i>tx_layer</i>	The layer name or the layer number.
<i>n_value</i>	The integer value of the constraint that you want to set.

Value Returned

<i>t</i>	Returns <i>t</i> if the value of the <code>minWidth</code> constraint is successfully set.
<i>nil</i>	Returns <i>nil</i> if the value of the <code>minWidth</code> constraint cannot be set successfully.

Example

```
dbSetPinMinPinWidth(pinId 0 4)  
=> t
```

dbUnsetAreaBoundaryLayerPriorityOrder

```
dbUnsetAreaBoundaryLayerPriorityOrder(  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

Deletes the `layerPriorityOrder` constraint on the given area boundary.

Arguments

`d_areaBoundaryId` The ID of the area boundary.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>layerPriorityOrder</code> constraint is successfully deleted.
<code>nil</code>	Returns <code>nil</code> if the <code>layerPriorityOrder</code> constraint is not deleted successfully.

Example

```
dbUnsetAreaBoundaryLayerPriorityOrder(areaBndyId)  
=> t
```

dbUnsetAreaBoundaryValidRoutingLayers

```
dbUnsetAreaBoundaryValidRoutingLayers(  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

Deletes the `validRoutingLayers` constraint on the given area boundary.

Arguments

d_areaBoundaryId The ID of the area boundary.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>validRoutingLayers</code> constraint is deleted successfully.
<code>nil</code>	Returns <code>nil</code> if the <code>validRoutingLayers</code> constraint cannot be deleted successfully.

Example

```
dbUnsetAreaBoundaryValidRoutingLayers(areaBndyId)  
=> t
```

dbUnsetPinGroupGuideKeepoutSpacing

```
dbUnsetPinGroupGuideKeepoutSpacing(  
    d_pinGroupGuide  
    tx_layer  
)  
=> t / nil
```

Description

Deletes the `keepoutSpacing` constraint on the given pin group guide.

Arguments

<i>d_pinGroupGuide</i>	The ID of the pin group guide.
<i>tx_layer</i>	The layer number or layer name.

Value Returned

<i>t</i>	Returns <i>t</i> if <code>keepoutSpacing</code> constraint is successfully deleted.
<i>nil</i>	Returns <i>nil</i> if the <code>keepoutSpacing</code> constraint cannot be deleted.

Example

```
dbSetPinGroupGuideKeepoutSpacing(epg1 0 1)  
=> t
```

dbUnsetPinGroupGuideLayerPriorityOrder

```
dbUnsetPinGroupGuideLayerPriorityOrder(  
    d_areaPinGroupGuide  
)  
=> t / nil
```

Description

Deletes `layerPriorityOrder` constraint on the given area pin group guide.

Arguments

d_areaPinGroupGuide The ID of the area pin group guide.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>layerPriorityOrder</code> constraint is successfully deleted.
<code>nil</code>	Returns <code>nil</code> if the <code>layerPriorityOrder</code> constraint cannot be deleted.

Example

```
dbUnsetPinGroupGuideLayerPriorityOrder(apgId)  
=> t
```


dbUnsetPinGroupGuideMinPinSpacing

```
dbUnsetPinGroupGuideMinPinSpacing(  
    d_pinGroupGuide  
    tx_layer  
)  
=> t / nil
```

Description

Deletes the `minSpacing` constraint on the given pin group guide.

Arguments

<i>d_PinGroupGuide</i>	The ID of the pin group guide.
<i>tx_layer</i>	The layer name or layer number.

Value Returned

<i>t</i>	Returns <i>t</i> if the <code>minSpacing</code> constraint is successfully deleted.
<i>nil</i>	Returns <i>nil</i> if the <code>minSpacing</code> constraint cannot be deleted successfully.

Example

```
dbUnsetPinGroupGuideMinPinSpacing(epg 0)  
=> t
```

dbUnsetPinGroupGuideMinPinWidth

```
dbUnsetPinGroupGuideMinPinSpacing(  
    d_pinGroupGuide  
    tx_layer  
)  
=> t / nil
```

Description

Deletes the `minWidth` constraint on the given pin group guide.

Arguments

<i>d_PinGroupGuide</i>	The ID of the pin group guide.
<i>tx_layer</i>	The layer name or layer number.

Value Returned

<i>t</i>	Returns <i>t</i> if the <code>minWidth</code> constraint is successfully deleted.
<i>nil</i>	Returns <i>nil</i> if the <code>minWidth</code> constraint cannot be deleted successfully.

Example

```
dbUnsetPinGroupGuideMinPinWidth(epg 0)  
=> t
```

dbUnsetPinGroupGuideValidRoutingLayers

```
dbUnsetPinGroupGuideValidRoutingLayers(  
    d_areapinGroupGuide  
)  
=> t / nil
```

Description

Deletes the `validRoutingLayers` constraint on the given area pin group guide.

Arguments

d_areaPinGroupGuide The ID of the area pin group guide.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>validRoutingLayers</code> constraint is successfully deleted.
<code>nil</code>	Returns <code>nil</code> if the <code>validRoutingLayers</code> constraint cannot be deleted successfully.

Example

```
dbUnsetPinGroupGuideValidRoutingLayers(apgId)  
=> t
```

dbUnsetPinMinPinSpacing

```
dbUnsetPinMinPinSpacing(  
    d_pinID  
    tx_layer  
)  
=> t / nil
```

Description

Deletes the `minSpacing` constraint on the given pin.

Arguments

<code>d_pinID</code>	The ID of the pin.
<code>tx_layer</code>	The layer name or the layer number.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>minSpacing</code> constraint is successfully deleted.
<code>nil</code>	Returns <code>nil</code> if the <code>minSpacing</code> constraint cannot be deleted successfully.

Example

```
dbUnsetPinMinPinSpacing(pinId 0)  
=> t
```

dbUnsetPinMinPinWidth

```
dbUnsetPinMinPinWidth(  
    d_pinID  
    tx_layer  
)  
=> t / nil
```

Description

Deletes the `minWidth` constraint on the given pin.

Arguments

<code>d_pinID</code>	The ID of the pin.
<code>tx_layer</code>	The layer name or the layer number.

Value Returned

<code>t</code>	Returns <code>t</code> if the <code>minWidth</code> constraint is successfully deleted.
<code>nil</code>	Returns <code>nil</code> if the <code>minWidth</code> constraint cannot be deleted successfully.

Example

```
dbUnsetPinMinPinWidth(pinId 0)  
=> t
```

Virtuoso Studio Design Environment SKILL Reference

Pin Database Access Functions

Placement Database Access Functions

This topic provides a list of Placement Database Access Cadence® SKILL functions.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

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](#)

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

- [dbGetRailDefAttr](#)
- [dbGetRailDefFigType](#)
- [dbSetRailDefFigType](#)

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](#)

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

- [dbAddRowRailDef](#)
- [dbGetRowRailDef](#)
- [dbRemoveRowCompTypeSetDef](#)
- [dbRemoveRowBackgroundDef](#)
- [dbRemoveRowRailDef](#)

Row Region Specification Functions

- [dbCreateRowRegionSpec](#)
- [dbFindRowRegionSpec](#)
- [dbGetCellViewRowRegionSpecs](#)

Row Region Functions

- [dbAttachRowRegionToPRBoundary](#)
- [dbCreateRowRegion](#)
- [dbDetachPRBoundaryFromRowRegion](#)
- [dbFindRowRegion](#)
- [dbFlattenRowRegion](#)
- [dbGetCellViewRowRegions](#)
- [dbGetRowRegionAttachedToPRBoundary](#)
- [dbGetRowRegionPlaceRows](#)
- [dbGetRowRegionPoints](#)
- [dbGetRowRegionRowRegionSpec](#)
- [dbGetRowRegionSpecRowRegions](#)
- [dbGetRowRegionUsesPartialRowSpec](#)
- [dbIsRowRegion](#)
- [dbIsRowRegionAttachedToPRBoundary](#)
- [dbRebuildRowRegion](#)
- [dbRowRegionMatchesSpec](#)

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

- [dbSetRowRegionPoints](#)
- [dbSetRowRegionRowRegionSpec](#)
- [dbSetRowRegionUsesPartialRowSpec](#)

Miscellaneous Functions

- [dbGetPlaceRowVersion](#)
- [dbGetRowRegionVersion](#)
- [dbIsImplicit](#)
- [dbIsInRowRegion](#)

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

<i>d_cvId</i>	Database ID of a cellview.
<i>t_name</i>	Name of a backgroundDef.
<i>l_lpp</i>	Layer-purpose pair.
<i>t_netName</i>	Name of a net.
<i>g_leftEnc</i>	Left enclosure of the background shape.
<i>g_rightEnc</i>	Right enclosure of the background shape.
<i>g_topEnc</i>	Top enclosure of the background shape.
<i>g_bottomEnc</i>	Bottom enclosure of the background shape.

Value Returned

<i>d_bgDefId</i>	Database ID of a backgroundDef.
<i>nil</i>	Returned in case of failure.

Examples

A background definition `bgDef1` is created in the cellview `cv` with the following attributes:

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

- Layer-purpose pair: FinFet region1
- Net: bgNet
- Left enclosure: 10
- Right enclosure: 20
- Top enclosure: 30
- Bottom enclosure: 40

```
bgDef1 = dbCreateBackgroundDef(cv "bgDef1" list("FinFet" "region1") "bgNet" 10 20  
30 40)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_cvId</i>	Database ID of a cellview.
<i>t_name</i>	Name of a backgroundDef.
<i>g_dpl</i>	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

<i>d_bgDefId</i>	Database ID of the backgroundDef that is created.
<i>nil</i>	Returned in case of failure.

Examples

Creates a SKILL DPL, and based on it, creates a background DPL.

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
; creating the SKILL DPL
    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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetBackgroundDefAttr

```
dbGetBackgroundDefAttr(  
    d_bgDefId  
)  
=> l_attr / nil
```

Description

Returns the attributes of the specified backgroundDef.

Arguments

<i>d_bgDefId</i>	Database ID of the backgroundDef.

Value Returned

<i>l_attr</i>	List of backgroundDef attributes.
<i>nil</i>	Returned in case of failure.

Examples

Creates a backgroundDef, and then retrieves its attributes.

```
;; 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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetCellViewBackgroundDef

```
dbGetCellViewBackgroundDef(  
    d_cellviewId  
)  
=> l_bgDefId / nil
```

Description

Returns list of backgroundDefs in the given cellview.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

Value Returned

<i>l_bgDefId</i>	List of backgroundDefs in the given cellview.
<i>nil</i>	Returned in case of failure.

Examples

Returns the list of background definitions in the cellview *cv*.

```
dbGetcellviewBackgroundDef(cv)
```

Related Topics

[Placement Database Access 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

<i>d_cellViewId</i>	Database ID of a cellview.
<i>t_compTypeSetName</i>	The name of the compTypeSetDef.
<i>t_rowReference</i>	Specifies row reference. Valid Values: bottomAlignment, centerAlignment, topAlignment.
<i>t_componentReference</i>	Specifies component reference. Valid Values: bottomAlignment, centerAlignment, topAlignment.
<i>l_allowedOrientations</i>	Allowed orientations that placer can use for cells in the component types.
<i>l_componentTypes</i>	List of component types that share placement attributes.
<i>g_componentOffset</i>	Optionally the offset from the reference points computed from row reference can be specified in user units.
<i>g_pitch</i>	Optionally the pitch or grid to use between placement locations can be specified.

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Value Returned

<i>d_compTypeSetDef</i>	Returns Database ID of a compTypeSetDef.
<i>nil</i>	An error occurred during execution.

Examples

Creates a container of component types that use the same placement attributes.

```
;; 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
    componentPitch) ; pitch or grid to use between placement locations
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_cvId</i>	Database ID of a cellview.
<i>t_name</i>	Name of a compTypeSetDef.
<i>l_attr_dpl</i>	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
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Value Returned

d_compTypeSetDef Database ID of the compTypeSetDef that is created.
nil Returned in case of failure.

Examples

Creates a compTypeSetDef in the `cellLib` cellview.

```
;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)
```

Related Topics

Placement Database Access Functions

dbGetCellViewCompTypeSetDef

```
dbGetCellViewCompTypeSetDef(  
    d_cellViewId  
)  
=> d_compTypeSetDef / nil
```

Description

This function returns a list of compTypeSetDef objects defined in the specified cellview.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>d_compTypeSetDef</i>	Returns a list of compTypeSetDef objects.
<i>nil</i>	If a compTypeSetDef objects does not exist, <i>nil</i> is returned.

Examples

Returns a list of compTypeSetDef objects defined in the current cellview.

```
cellViewId = geGetEditCellView()  
  
compTypeSetDefs = dbGetCellViewCompTypeSetDef(cellViewId)
```

Related Topics

[Placement Database Access Functions](#)

dbGetCompTypeSetDefAttr

```
dbGetCompTypeSetDefAttr(  
    d_compDefId  
)  
=> l_attr / nil
```

Description

Returns the attributes of the specified dbCompTypeSetDef.

Arguments

<i>d_compDefId</i>	Database ID of compTypeSetDef.

Value Returned

<i>l_attr</i>	Lists the attributes of the specified compTypeSetDef.
<i>nil</i>	Returned in case of failure.

Examples

Creates a compTypeSetDef and returns its attributes.

```
;; 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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetCompTypeSetDefCompFilters

```
dbGetCompTypeSetDefCompFilters(  
    d_compDefId  
)  
=> l_compFilters / nil
```

Description

Returns the component filters in the specified compTypeSetDef.

Arguments

<i>d_compDefId</i>	Database ID of a compTypeSetDef.

Value Returned

<i>l_compFilters</i>	List of component filters.
<i>nil</i>	Returns an error in case of failure.

Examples

Creates a compTypeSetDef and retrieves its compFilters.

```
; 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)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Related Topics

[Placement Database Access Functions](#)

dbSetCompTypeSetDefCompFilters

```
dbSetCompTypeSetDefCompFilters(  
    d_compDefId  
    l_compFilters  
)  
=> t/nil
```

Description

Sets the component filters in the specified compTypeSetDef.

Arguments

<i>d_compDefId</i>	Database ID of compTypeSetDef.
<i>l_compFilters</i>	List of component filters.

Value Returned

<i>t</i>	Updated component filters on compTypeSetDef.
<i>nil</i>	Returned in case of failure.

Examples

Creates a compTypeSetDef and sets its component filters.

```
;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")))
```

Related Topics

[Placement Database Access Functions](#)

dbUnsetCompTypeSetDefCompFilters

```
dbUnsetCompTypeSetDefCompFilters(  
    d_compDefId  
)  
=> t / nil
```

Description

Removes the component filters from the specified compTypeSetDef.

Arguments

<i>d_compDefId</i>	Database ID of a compTypeSetDef.

Value Returned

t	Removed component filters from the specified compTypeSetDef.
nil	Returned in case of failure.

Example

Creates a compTypeSetDef, sets its component filters, and removes the filters.

```
; 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)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Related Topics

[Placement Database Access Functions](#)

dbCreateRailDef

```
dbCreateRailDef(  
    d_cellViewId  
    t_railDefName  
    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 modified in the object, it is instantly reflected in all row/placeArea that reference the railDef.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>t_railDefName</i>	Name of the railDef.
<i>t_netName</i>	Name of the net.
<i>l_lpp</i>	Layer and purpose of the rail geometry.
<i>t_rowReference</i>	Specifies row reference. Valid Values: bottomAlignment, centerAlignment, topAlignment.
<i>t_railReference</i>	Specifies rail reference. Valid Values: bottomAlignment, centerAlignment, topAlignment.
<i>g_width</i>	Width of rail geometry.

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

<i>g_offset</i>	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.
<i>g_beginOffset</i>	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.
<i>g_endOffset</i>	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

<i>d_railDef</i>	Returns the database ID of the railDef.
<i>nil</i>	An error occurred during execution.

Examples

Creates railDef

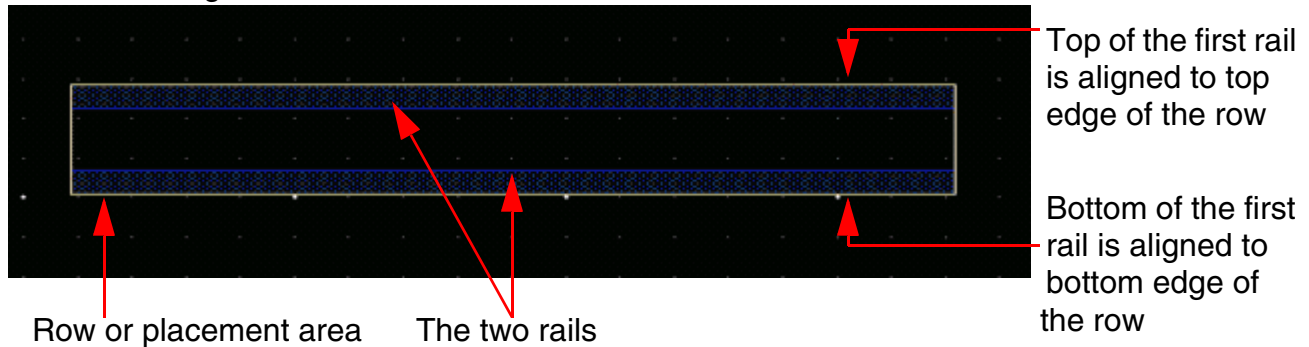
```
def = dbCreateRailDef(cellViewId
    "myRailDef"
    "netName"
    list("layer" "purpose")
    "bottomAlignment"
    "bottomAlignment"
    railWidth
    offsetFromRowReference)
```

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

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

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:



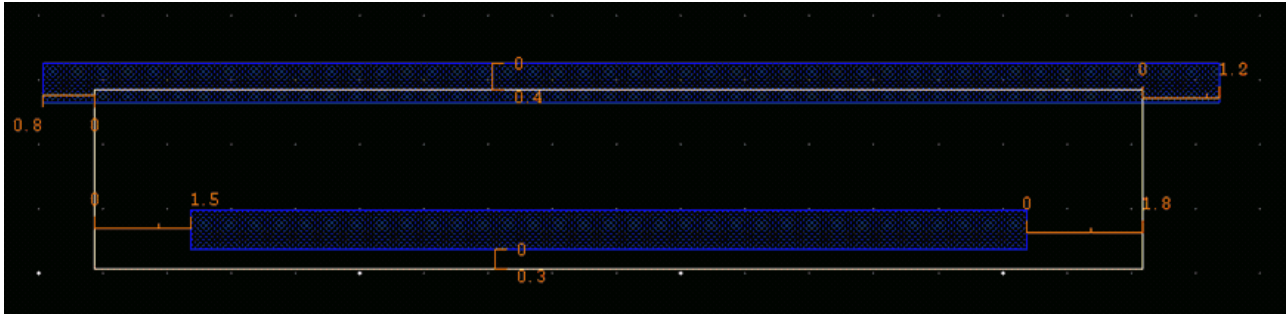
The following table specifies, in addition to the offset values, the begin and end offset values for the two rails.

Rail	Offset	Begin offset	End offset
Top rail	0.4	-0.8	+1.2
Bottom rail	0.3	+1.5	-1.8

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Note: A negative begin/end offset value sets the rail relative to the left side of the row and positive begin/end offset value sets the rail relative to the right side of the row. This is shown in the figure below:



Related Topics

[Placement Database Access Functions](#)

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

<i>d_cvId</i>	Database ID of a cellview.
<i>t_name</i>	Name of a railDef.
<i>l_attr_dpl</i>	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
```


Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Value Returned

<i>d_railDef</i>	Database ID of railDef that is created.
<i>nil</i>	Returned in case of failure.

Examples

Creates a SKILL DPL for railDef specification and creates railDef based on the SKILL DPL.

```
;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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetCellViewRailDef

```
dbGetCellViewRailDef(  
    d_cellViewId  
)  
=> l_railDefObjects / nil
```

Description

Returns a list of railDef objects defined in the specified cellview.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>l_railDefObjects</i>	Returns a list of railDef objects.
<i>nil</i>	If no railDef objects exist, <i>nil</i> is returned.

Examples

Returns a list of railDef objects in the current cellview.

```
cellViewId = geGetEditCellView()  
  
defs = dbGetCellViewRailDef(cellViewId)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRailDefAttr

```
dbGetRailDefAttr(  
    d_railDefId  
)  
=> l_attr / nil
```

Description

Returns the rail attributes of the specified railDef.

Arguments

<i>d_railDefId</i>	Database ID of the railDef database.

Value Returned

<i>l_railAttrs</i>	List of rail attributes of the specified railDef.
<i>nil</i>	Returned in case of failure.

Examples

Creates rail attributes and retrieves them.

```
;; 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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRailDefFigType

```
dbGetRailDefFigType(  
    d_railDefId  
)  
=> t_figType / nil
```

Description

Returns the figType specified in the given railDef.

Arguments

<i>d_railDefId</i>	Database ID of railDef.

Value Returned

<i>t_figType</i>	Returns figType of the specified railDef. Valid values are <code>rect</code> and <code>pathSeg</code> .
<code>nil</code>	Returned in case of failure.

Examples

Creates rail attributes and retrieves their figType.

```
; 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)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Related Topics

[Placement Database Access Functions](#)

dbSetRailDefFigType

```
dbSetRailDefFigType(  
    d_railDefId  
    t_railFigType  
)  
=> t / nil
```

Description

Sets the railFigType for the specified railDef.

Arguments

<i>d_railDef</i>	Database ID of a railDef.
<i>t_railFigType</i>	Value of figType. Valid values are default, rect and pathSeg. Default value is rect.

Value Returned

<i>t</i>	Specified figType is set on the given railDef.
<i>nil</i>	Returned in case of failure.

Examples

Sets figType pathseg on rail definition d_rowRegionId.

```
dbSetRailDefFigType(d_rowRegionId "pathSeg")
```

Related Topics

[Placement Database Access 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

<i>d_plcAreaId</i>	Database ID of placeArea.
<i>d_bgDefId</i>	Database ID of backgroundDef.

Value Returned

<i>t</i>	A reference to the specified placeArea object is added to the specified backgroundDef.
<i>nil</i>	Returned in case of failure.

Examples

Adds a reference to the placement area object *plcAreaId* in the given background definition *bgDefId*.

```
dbAddPlaceAreaBackgroundDef(plcAreaId bgDefId)
```

Related Topics

[Placement Database Access Functions](#)

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.

Examples

Creates a placement area, defines its component types, and adds a reference in the specified placeArea to the specified compTypeSetDefs.

```
;; 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
```


Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
componnentPitch)
```

```
;; Add reference to compTypeSetDef in placeArea  
dbAddPlaceAreaCompTypeSetDef(placeArea def)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_placeAreaId</i>	Database ID of a placeArea.
<i>d_railDefId</i>	Database ID of a railDef.

Value Returned

t	The reference was added successfully.
nil	An error occurred during execution.

Examples

Creates a placement area, defines rails, and adds a reference in the specified placeArea to the specified railDef.

```
;; 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
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
offsetFromRowReference)
```

```
;; Add reference to railDef in placeArea  
dbAddPlaceAreaRailDef(placeArea def)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_cellViewId</i>	Database ID of a cellview.
<i>t_name</i>	Name of a placeArea.
<i>g_width</i>	Width of the placeArea in user units.
<i>g_height</i>	Height of the placeArea in user units.
<i>l_origin</i>	Origin of the placeArea.
<i>t_orientation</i>	Orientation of the placeArea.
<i>f_utilization</i>	Specifies the utilization of the placeArea. The value is should be limited to [0.0,1.0], where 1.0 == 100% utilization.

Value Returned

<i>d_IdOfCustomPlacementArea</i>	Database ID of the placeArea is returned.
<i>nil</i>	An error occurred during execution.

Examples

Creates a custom placeArea.

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
placeArea = dbCreatePlaceArea (geGetEditCellView() "myPlaceArea" 100.0 10.0 "R0")
```

Related Topics

[Placement Database Access Functions](#)

dbGetCellViewPlaceArea

```
dbGetCellViewPlaceArea(  
    d_cellviewId  
)  
=> l_customPlacementAreas / nil
```

Description

Returns a list of custom placeAreas in the specified cellview.

Arguments

<i>d_cellviewIdy</i>	Database ID of the cellview.

Value Returned

<i>l_customPlacementAreas</i>	A list of custom placeAreas is returned.
<i>nil</i>	The cellview ID is invalid or no custom placeAreas exist.

Examples

Returns a list of custom placeAreas.

```
myCustomPlaceAreas = dbGetCellViewPlaceArea (geGetEditCellView())
```

Related Topics

[Placement Database Access Functions](#)

dbGetPlaceAreaBackgroundDef

```
dbGetPlaceAreaBackgroundDef(  
    d_plcAreaId  
)  
=> l_bgDefId / nil
```

Description

Returns a list of database IDs of the backgroundDefs in the specified placeArea.

Arguments

<i>d_placeAreaId</i>	Database ID of a placeArea.

Value Returned

<i>l_bgDefId</i>	List of database IDs of backgroundDefs.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetPlaceAreaBackgroundDef(placeAreaId)
```

Related Topics

[Placement Database Access Functions](#)

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.

Examples

Returns a list of database IDs of the background definitions in the placement area placeAreaId.

```
placeArea = car(dbGetCellViewPlaceArea(getEditCellView()))  
myCompTypeSetDefs = dbGetPlaceAreaCompTypeSetDef(placeArea)
```

Related Topics

[Placement Database Access Functions](#)

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

Examples

Retrieves the the orientation of the specified custom place area.

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView()))  
dbGetPlaceAreaOrient(placeArea)
```

Related Topics

[Placement Database Access Functions](#)

dbGetPlaceAreaRailDef

```
dbGetPlaceAreaRailDef(  
    d_customPlacementArea  
)  
=> l_dbRailDefs / nil
```

Description

Returns a list of dbRailDefs referenced in the specified custom placeArea.

Arguments

<i>d_customPlacementArea</i>	Database ID of a custom placeArea.

Value Returned

<i>l_dbRailDefs</i>	A list of dbRailDefs is returned.
<i>nil</i>	The custom placeArea ID is invalid

Examples

Returns a list of dbRailDefs referenced in the current cellview.

```
row = car(dbGetCellViewPlaceArea(getEditCellView()))  
myRailDefs = dbGetPlaceAreaRailDef(row)
```

Related Topics

[Placement Database Access Functions](#)

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

Examples

Returns the assigned placement utilization of the current cellview.

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView()))  
assignedUtilization = dbGetPlaceAreaUtilization(placeArea)
```

Related Topics

[Placement Database Access Functions](#)

dbGetPlacementGrid

```
dbGetPlacementGrid(  
    d_cellviewId  
)  
=> l_placementGridInfo / nil
```

Description

Returns the placement grid information for the specified cellview.

Arguments

<i>d_cellviewId</i>	Database ID of the cellview.

Value Returned

<i>l_placementGridInfo</i>	The origin of the placement grid, and the pitch X and Y values are returned.
<i>nil</i>	The cellview ID is invalid

Examples

The following returns the X origin, Y origin, X pitch, and Y pitch

```
dbGetPlacementGrid(getEditCellView())  
((0.0 0.0) 10.0 10.0)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_plcAreaId</i>	Database ID of a placeArea.
<i>d_bgDefId</i>	Database ID of a backgroundDef.

Value Returned

<i>t</i>	Association between the given placeArea and backgroundDef is successfully removed.
<i>nil</i>	Returned in case of failure.

Examples

Creates a placeArea and backgroundDef and removes the association between them.

```
; 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)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Related Topics

[Placement Database Access Functions](#)

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.

Examples

Creates a placeArea and backgroundDef and removes reference to the compTypeSetDef in the specified placeArea.

```
;; 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
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
componentPitch)
```

```
;; Add reference to compTypeSetDef in placeArea  
dbAddPlaceAreaCompTypeSetDef(placeArea def)
```

```
;; removes reference  
dbRemovePlaceAreaCompTypeSetDef(placeArea def)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_placeAreaId</i>	Database ID of a placeArea.
<i>d_railDefId</i>	Database ID of a railDef.

Value Returned

t	The reference was removed successfully.
nil	An error occurred during execution.

Examples

Creates a placeArea and railDef and adds reference to the railDef in the placeArea. dbRemovePlaceAreaRailDef is then used to remove the reference.

```
;; create placeArea  
placeArea = dbCreatePlaceArea(cellViewId  
    name  
    width  
    height  
    origin  
    orient  
    utilization)  
  
;; create railDef  
def = dbCreateRailDef(cellViewId  
    "myRailDef"  
    "netName"  
    list("layer" "purpose")  
    "bottomAlignment"  
    "bottomAlignment"  
    railWidth
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
offsetFromRowReference)
```

```
;; Add reference to railDef in placeArea  
dbAddPlaceAreaRailDef(placeArea def)
```

```
;; removes reference and rail geometry  
dbRemovePlaceAreaRailDef(placeArea def)
```

Related Topics

[Placement Database Access Functions](#)

dbSetPlaceAreaOrient

```
dbSetPlaceAreaOrient(  
    d_customPlacementArea  
    s_orientation  
)  
=> t / nil
```

Description

Sets the orientation of the specified custom placeArea.

Arguments

<i>d_customPlacementArea</i>	Database ID of a custom placeArea.
<i>s_orientation</i>	The orientation of the custom placeAreas.

Value Returned

<i>t</i>	The orientation is set successfully.
<i>nil</i>	The custom placeArea ID is invalid

Examples

Sets the orientation of the place area in the current cellview.

```
placeArea = car(dbGetCellViewPlaceArea(geGetEditCellView()))  
dbSetPlaceAreaOrient(placeArea "R90")
```

Related Topics

[Placement Database Access Functions](#)

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

Examples

Sets the assigned placement utilization for the place area in the current cellview.

```
placeArea = car(dbGetCellViewPlaceArea(getEditCellView()))  
dbSetPlaceAreaUtilization(placeArea .8)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_cellviewId</i>	Database ID of the cellview.
<i>l_origin</i>	The origin of the placement grid.
<i>f_pitchX</i>	The X pitch of the placement grid.
<i>f_pitchY</i>	The Y pitch of the placement grid.

Value Returned

t	The placement grid values are set successfully.
nil	The placement grid values are invalid

Examples

Sets the placement grid information for the place area in the current cellview.

```
dbSetPlacementGrid(geGetEditCellView() 0:0 10.0 10.0)
```

Related Topics

[Placement Database Access 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

<i>d_rowId</i>	Database ID of the row.
<i>d_bgDefId</i>	Database ID of the backgroundDef.

Value Returned

<i>t</i>	A reference to the specified row object is added to the specified backgroundDef.
<i>nil</i>	Returned in case of failure.

Examples

Adds a reference to the placement area object `rowId` in the given background definition `bgDefId`.

```
dbAddRowBackgroundDef(rowId bgDefId)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_row_Id</i>	Database ID of a row.
<i>d_compTypeSetDefId</i>	Database ID of a compTypeSetDef.

Value Returned

<i>t</i>	The reference was added successfully.
<i>nil</i>	An error occurred during execution.

Examples

Creates a row and compTypeSetDef. Adds reference to the compTypeSetDef in a row.

```
;; create row  
row = dbCreateRow(cellViewId  
    siteDef  
    name  
    origin  
    numSites)  
  
;; create compTypeSetDef  
def = dbCreateCompTypeSetDef(cellViewId  
    "myCompTypeSetDef"  
    "bottomAlignment"  
    "bottomAlignment"  
    list("R0" "MX")  
    list("Comptype1" "Comptype2")  
    offsetFromRowReference  
    componentPitch)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
;; Add reference to compTypeSetDef in row  
dbAddRowCompTypeSetDef(row def)
```

Related Topics

Placement Database Access Functions

dbGetRowBackgroundDef

```
dbGetRowBackgroundDef(  
    d_rowId  
)  
=> l_bgDefId / nil
```

Description

Returns a list of backgroundDefs in the specified custom row.

Arguments

<i>d_rowId</i>	Database ID of a custom row.

Value Returned

<i>l_bgDefId</i>	List of backgroundDefs in the specified custom row.
<i>nil</i>	Returned in case of failure.

Examples

Returns a list of backgroundDefs in the specified row.

```
; 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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowCompTypeSetDef

```
dbGetRowCompTypeSetDef(  
    d_rowId  
)  
=> l_compTypeSetDefs / nil
```

Description

Returns a list of dbCompTypeSetDefs referenced in the specified row.

Arguments

<i>d_rowId</i>	Database ID of the row.

Value Returned

<i>l_dbCompTypeSetDefs</i>	A list of dbCompTypeSetDefs is returned.
<i>nil</i>	The row ID is invalid

Examples

Returns a list of dbCompTypeSetDefs in the specified row.

```
row = car(geGetEditCellView()~>rows)  
myCompTypeSetDefs = dbGetRowCompTypeSetDef(row)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_rowId</i>	Database ID of a row.
<i>d_railDefId</i>	Database ID of a railDef.

Value Returned

<i>t</i>	The reference was added successfully.
<i>nil</i>	An error occurred during execution.

Examples

Ads a reference in the specified railDef to the specified row object.

```
;; create row  
row = dbCreateRow(cellViewId  
    siteDef  
    name  
    origin  
    numSites)  
  
;; create railDef  
def = dbCreateRailDef(cellViewId  
    "myRailDef"  
    "netName"  
    list("layer" "purpose")  
    "bottomAlignment"  
    "bottomAlignment"  
    railWidth  
    offsetFromRowReference)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
;; Add reference to railDef in row  
dbAddRowRailDef(row def)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRailDef

```
dbGetRowRailDef(  
    d_rowId  
)  
=> l_dbRailDefs / nil
```

Description

Returns a list of dbRailDefs referenced in the specified row.

Arguments

<i>d_rowId</i>	Database ID of the row.

Value Returned

<i>l_dbRailDefs</i>	A list of dbRailDefs is returned.
<i>nil</i>	The row ID is invalid

Examples

Returns a list of dbRailDefs referenced in the given row.

```
row = car(getEditCellView()~>rows)  
myRailDefs = dbGetRowRailDef(row)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_rowId</i>	Database ID of a row.
<i>d_compTypeSetDefId</i>	Database ID of a compTypeSetDef.

Value Returned

<i>t</i>	The reference was removed successfully.
<i>nil</i>	An error occurred during execution.

Examples

Removes a reference to the specified compTypeSetDef in the specified row.

```
;; create row  
row = dbCreateRow(cellViewId  
    siteDef  
    name  
    origin  
    numSites)  
;; create compTypeSetDef  
def = dbCreateCompTypeSetDef(cellViewId  
    "myCompTypeSetDef"  
    "bottomAlignment"  
    "bottomAlignment"  
    list("R0" "MX")  
    list("CompType1" "CompType2")  
    offsetFromRowReference  
    componentPitch)  
;; Add reference to compTypeSetDef in row  
dbAddRowCompTypeSetDef(row def)  
;; removes reference  
dbRemoveRowCompTypeSetDef(row def)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Related Topics

[Placement Database Access Functions](#)

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

<i>d_rowId</i>	Database ID of the custom row.
<i>d_bgDefId</i>	Database ID of the backgroundDef.

Value Returned

<i>t</i>	Returned when backgroundDef is successfully removed from the custom row.
<i>nil</i>	Returned in case of failure.

Examples

Removes the given backgroundDef from a custom row.

```
; 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)
```

Related Topics

[Placement Database Access Functions](#)

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

<i>d_rowId</i>	Database ID of a row.
<i>d_railDefId</i>	Database ID of a railDef area.

Value Returned

t	The reference was removed successfully.
nil	An error occurred during execution.

Examples

Removes a reference to the specified railDef in the specified row.

```
;; 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
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

`dbRemoveRowRailDef(row def)`

Related Topics

[Placement Database Access Functions](#)

dbCreateRowRegionSpec

```
dbCreateRowRegionSpec (
    d_cvId
    t_name
    { n_height | d_siteDef }
    l_rowAttrs
    [ l_refGridNames ]
    [ l_refGridOffsets ]
    [ t_relSnapPatternName ]
    [ t_type ]
)
=> d_specId / nil
```

Description

Creates rowRegionSpec in the specified cellview.

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.

You can create specifications to create two types of rows:

- dbRow - Needs siteDef from techDB
- dbPlaceArea - Does not need siteDef from techDB

Arguments

<i>d_cvId</i>	Database ID of a cellview.
<i>t_name</i>	Name of rowRegionSpec.

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

Value Returned

<i>d_specID</i>	Database ID of the rowRegionSpec that is created.
<i>nil</i>	Returned in case of failure.

Examples

There are two ways in which you can specify this function. These are illustrated in the examples below.

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

;; Create Row Attributes
row = list(nil)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
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))
```

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
        rightEnc     0.8
        topEnc       0.9
        botEnc       1.1
      )
    )
  )
)
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
    )
compAttrs (
    (nil
    refOnArea      "centerAlignment"
    refOnComp      "anyAlignment"
    allowedTypes  ("compType1" "compType2")
    orients        ("R0")
    compFilters    (("tn010" "nf*" "layout") ("tn010" "ng*" "layout"))
    )
    )
)
)
)
```

Related Topics

[Placement Database Access Functions](#)

dbFindRowRegionSpec

```
dbFindRowRegionSpec (
    d_cvId
    t_name
)
=> d_specId / nil
```

Description

Searches for a rowRegionSpec with the specified name in the given cellview.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.
<i>t_name</i>	Name of a rowRegionSpec.

Value Returned

<i>d_rowRegionSpecId</i>	Database ID of the searched rowRegionSpec, if found.
<i>nil</i>	Returned in case of failure.

Examples

Searches for the rowRegionSpec `rowRegion` in the cellview `cv`.

```
dbFindRowRegionSpec(cv "rowRegion")
```

Related Topics

[Placement Database Access Functions](#)

dbGetCellViewRowRegionSpecs

```
dbGetCellViewRowRegionSpecs(  
    d_cellviewId  
)  
=> l_rowRegionSpecs / nil
```

Description

Returns a list of rowRegionSpecs in the specified cellview.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

Value Returned

<i>l_rowRegionSpecs</i>	List of database IDs of the rowRegionSpecs in the given cellview.
<i>nil</i>	Returned in case of failure.

Example

Returns a list of RowRegionSpecs in the cellview *cv*.

```
dbGetCellViewRowRegionSpecs(cv)
```

Related Topics

[Placement Database Access Functions](#)

dbAttachRowRegionToPRBoundary

```
dbAttachRowRegionToPRBoundary(  
    d_rowRegionId  
)  
=> t / nil
```

Description

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

<i>d_rowRegionId</i>	Database ID of a rowRegion.
<i>d_bgDefId</i>	Database ID of a backgroundDef.

Value Returned

<i>t</i>	The given rowRegion is attached to the specified PRBoundary.
<i>nil</i>	Returned in case of failure.

Examples

Associates rowRegion with a custom PR boundary in the same block.

```
;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)
```

Related Topics

[Placement Database Access Functions](#)

dbCreateRowRegion

```
dbCreateRowRegion(  
    d_cvId  
    t_name  
    t_specName  
    l_points  
)  
=> d_rowRegionId/nil
```

Description

Creates a rowRegion in the specified cellview. The contents of the rowRegion are created based on the template defined in the rowRegionSpec. This function supports only non-concave rectilinear polygons.

A rowRegion is a figGroup that is created and edited on the canvas. It references a rowRegionSpec. A rowRegion should match its corresponding rowRegionSpec. If a 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.

Arguments

<i>d_cellviewID</i>	Database ID of a cellview.
<i>t_name</i>	Name of a rowRegion.
<i>t_specName</i>	Name of a rowRegionSpec.
<i>l_points</i>	List of the points that define the areaBoundary enclosing the rowRegion.

Value Returned

<i>d_rowRegionId</i>	Database ID of rowRegion that is created.
<i>nil</i>	Returned in case of failure.

Examples

Creates a row region named `rowRegion` in the cellview `cvId` based on the information in `rowRegionSpec`.

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
dbCreateRowRegion(cvId "rowRegion" "rowRegionSpec" list(list(0 0) list(2000 0)  
list(2000 5000) list(0 5000)))
```

Related Topics

[Placement Database Access Functions](#)

dbDetachPRBoundaryFromRowRegion

```
dbDetachPRBoundaryFromRowRegion(  
    d_PRBoundaryId  
)  
=> t / nil
```

Description

Detaches the PRBoundary from any associated rowRegion.

Arguments

d_PRBoundaryId Database ID of PRBoundary.

Value Returned

t PRBoundary is detached from the rowRegion.
nil Returned in case of failure.

Example

Creates a row region and PR boundary and attaches them.

`dbDetachPRBoundaryFromRowRegion` is then used to detach them.

```
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)
```

Related Topics

[Placement Database Access Functions](#)

dbFindRowRegion

```
dbFindRowRegion(  
    d_cvId  
    t_name  
)  
=> d_rowRegionId / nil
```

Description

Searches for a rowRegion by the specified name in the given cellview.

Arguments

<i>d_cvId</i>	Database ID of a cellview.
<i>t_name</i>	Name of a rowRegion to search.

Value Returned

<i>d_rowRegionId</i>	Database ID of the searched rowRegion, if found.
<i>nil</i>	Returned in case of failure.

Examples

Searches for rowRegion in the cellview cv.

```
;Create rowRegion  
    rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point)  
  
;; Find rowRegion by name.  
dbFindRowRegion(cv "rowRegion")
```

Related Topics

[Placement Database Access Functions](#)

dbFlattenRowRegion

```
dbFlattenRowRegion(  
    d_rowRegionId  
)  
=> t / nil
```

Description

Flattens the rowRegion by removing information related to the associated figGroup and areaBoundary, and retaining information related to rows and other shapes.

Arguments

<i>d_rowRegionId</i>	Database ID of the rowRegion to be flattened.

Value Returned

t	The rowRegion is successfully flattened.
nil	Returned in case of failure.

Examples

The specified row region rowRegion is successfully flattened.

```
dbFlattenRowRegion(rowRegion)  
=> t
```

Related Topics

[Placement Database Access Functions](#)

dbGetCellViewRowRegions

```
dbGetCellViewRowRegions(  
    d_cellviewId  
)  
=> l_rowRegionSpecs / nil
```

Description

Returns a list of rowRegions in the specified cellview.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

Value Returned

<i>l_rowRegionSpecs</i>	List of database IDs of rowRegion in the given cellview.
<i>nil</i>	Returned in case of failure.

Examples

Returns a list of rowRegions in the cellview *cv*.

```
dbGetcellviewRowRegion(cv)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRegionAttachedToPRBoundary

```
dbGetRowRegionAttachedToPRBoundary(  
    d_prBoundaryId  
)  
=> d_rowRegionId / nil
```

Description

Returns the database ID of the rowRegion attached to the given PRBoundary.

Arguments

<i>d_prBoundaryId</i>	Database ID of a PRBoundary.

Value Returned

<i>d_rowRegionId</i>	Database ID of the rowRegion attached to the specified PRBoundary.
<i>nil</i>	Returned when no rowRegion is attached to the PRBoundary.

Examples

Returns the database ID of the rowRegion attached to PR boundary prb.

```
;; 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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRegionPlaceRows

```
dbGetRowRegionPlaceRows(  
    d_rowRegionId  
=> l_placeRowId / nil
```

Description

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

<i>d_rowRegionId</i>	Database ID of a rowRegion.

Value Returned

<i>placeRowId</i>	List of database IDs for standard rows or custom placeArea objects.
<i>nil</i>	Returned in case of failure.

Examples

Returns a list of placement rows scoped to `rowRegionId`.

```
dbGetRowRegionPlaceRows(rowRegionId)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRegionPoints

```
dbGetRowRegionPoints(  
    d_rowRegionId  
)  
=> l_points / nil
```

Description

Returns a list of rectilinear points that define the rowRegion boundary.

Arguments

<i>d_rowRegionId</i>	Database ID of a rowRegion.

Value Returned

<i>l_points</i>	List of the rectilinear points.
<i>nil</i>	Returned in case of failure.

Example

Returns the points of the rowRegion boundary.

```
; 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)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRegionRowRegionSpec

```
dbGetRowRegionRowRegionSpec (
    d_rowRegionId
)
=> d_rowRegionSpecId / nil
```

Description

Returns the rowRegionSpec from which the specified rowRegion was generated.

Arguments

<i>d_rowRegionId</i>	Database ID of a rowRegion.

Value Returned

<i>d_rowRegionSpecId</i>	Database ID of the rowRegionSpec from which the specified rowRegion was generated.
<i>nil</i>	Returned in the case of failure.

Examples

Returns the row region specification from which the row region `rowRegionId` was generated.

```
dbGetRowRegionRowRegionSpec (rowRegionId)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRegionSpecRowRegions

```
dbGetRowRegionSpecRowRegions(  
    d_rowRegionSpecId  
)  
=> l_rowRegions / nil
```

Description

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.

Examples

Returns a list of rowRegions built on the basis of the rowRegionSpec *rowRegionSpecId*.

```
dbGetRowRegionSpecRowRegions(rowRegionSpecId)
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRegionUsesPartialRowSpec

```
dbGetRowRegionUsesPartialRowSpec(  
    d_rowRegionID  
)  
=> t / nil
```

Description

Confirms whether the fill space of rowRegion should be populated with partial specialization of rowRegionSpec.

Arguments

<i>d_rowRegionId</i>	Database ID of a rowRegion.

Value Returned

<i>t</i>	Returned if the rowRegion uses partial specialization of rowRegionSpec.
<i>nil</i>	Returned if the rowRegion does not use partial specialization of rowRegionSpec.

Examples

Checks if a custom rowRegion uses partial rowRegionSpec.

```
; Create rowRegion.  
rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point t "R90")  
; Query if the rowRegion uses partial rowRegionSpec.  
    dbGetRowRegionUsesPartialRowSpec(rowRegion)
```

Related Topics

[Placement Database Access Functions](#)

dbIsRowRegion

```
dbIsRowRegion(  
    d_Id  
)  
=> t / nil
```

Description

Checks if the specified database object is a rowRegion.

Arguments

<i>d_Id</i>	Database ID of an object.

Value Returned

<i>t</i>	Returned if the object is a rowRegion.
<i>nil</i>	Returned if the object is not a rowRegion.

Examples

Returns *t* because *rowRegion* is a row region.

```
rowRegion=dbCreateRowRegion(...)  
dbIsRowRegion(rowRegion)
```

Related Topics

[Placement Database Access Functions](#)

dbIsRowRegionAttachedToPRBoundary

```
dbIsRowRegionAttachedToPRBoundary(  
    d_rowRegionId  
)  
=> t / nil
```

Description

Checks whether the specified rowRegion is associated with the PRBoundary of the block that contains the rowRegion.

Arguments

<i>d_rowRegionId</i>	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.

Examples

Returns t because the row region is attached to the PRBoundary.

```
;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)
```

Related Topics

[Placement Database Access Functions](#)

dbRowRegionPointsCutOut

```
dbRowRegionPointsCutOut(  
    d_rowRegionId  
    l_box  
)  
=> t / nil
```

Description

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

<i>d_rowRegionId</i>	Database ID of a rowRegion.
<i>l_box</i>	The box to cut out from the current rowRegion.

Value Returned

<i>t</i>	The rowRegion specification was modified successfully.
<i>nil</i>	An error occurred during execution.

Examples

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)
```

Related Topics

Placement Database Access Functions

dbRebuildRowRegion

```
dbRebuildRowRegion(  
    d_rowRegionId  
)  
=> t / nil
```

Description

Removes contents of the specified rowRegion and then regenerates it based on the rowRegionSpec associated with it.

Arguments

<i>d_rowRegionId</i>	Database ID of a rowRegion.

Value Returned

<i>t</i>	Returned in case of success.
<i>nil</i>	Returned in case of failure.

Examples

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 attributes of its corresponding rowRegionSpec.

```
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.
```

Related Topics

[Placement Database Access Functions](#)

dbRowRegionMatchesSpec

```
dbRowRegionMatchesSpec(  
    d_rowRegionId  
)  
=> t / nil
```

Description

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

<i>d_rowRegionId</i>	Database ID of a rowRegion.

Value Returned

<i>t</i>	Specified rowRegion matches its rowRegionSpec.
<i>nil</i>	Specified rowRegion does not match its rowRegionSpec.

Examples

Checks whether the rowRegion matches its rowRegionSpec.

```
;; 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
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
dbRowRegionMatchesSpec(rowRegion)
\t nil
;; 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
```

Related Topics

[Placement Database Access Functions](#)

dbSetRowRegionPoints

```
dbSetRowRegionPoints(  
    d_rowRegionId  
    l_points  
)  
=> t / nil
```

Description

Changes the rectilinear specification of the given rowRegion.

Arguments

<i>d_rowRegionId</i>	Database ID of a rowRegion.
<i>l_points</i>	List of rectilinear points specifying the area covered by the rowRegion.

Value Returned

<i>t</i>	Modified rowRegion points.
<i>nil</i>	Returned in case of failure.

Examples

Defines a set of rectilinear points and sets the rowRegion points.

```
point = list(list(0 0) list(3000 0) list(3000 5000) list(0 5000))  
;; Set rowRegion points  
dbSetRowRegionPoints(rowRegion point)
```

Related Topics

[Placement Database Access Functions](#)

dbSetRowRegionRowRegionSpec

```
dbSetRowRegionRowRegionSpec (
    d_rowRegionId
    d_rowRegionSpecId
)
=> t / nil
```

Description

Sets the given rowRegionSpec as the template for the specified rowRegion.

Arguments

<i>d_rowRegionId</i>	Database ID of a rowRegion.
<i>d_rowRegionSpecId</i>	Database ID of a rowRegionSpec.

Value Returned

<i>t</i>	Sets the given rowRegionSpec as the row template for the specified rowRegion.
<i>nil</i>	Returned in case of failure.

Examples

Sets a row region specification as the row region template.

```
; 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")
```

Virtuoso Studio Design Environment SKILL Reference

Placement Database Access Functions

```
compAttr->orients = list("R0")
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)
```

Related Topics

Placement Database Access Functions

dbSetRowRegionUsesPartialRowSpec

```
dbSetRowRegionUsesPartialRowSpec(  
    d_rowRegionId  
    g_value  
)  
=> t / nil
```

Description

A Boolean indicating whether the population of rowRegion is set to fill the space with partial specialization of rowRegionSpec.

Arguments

<i>d_rowRegionId</i>	Database ID of a rowRegion.
<i>g_value</i>	A Boolean that determines whether rowRegion should be set to use partial specialization of rowRegionSpec.

Value Returned

<i>t</i>	Set or unset partial specialization of rowRegionSpec.
<i>nil</i>	Returned in case of failure.

Examples

Checks whether rowRegion is set to use partial specialization of rowRegionSpec.

```
; Create rowRegion  
rowRegion = dbCreateRowRegion(cv "rowRegion" "newRowRegionSpec" point)  
  
;; Set RowRegion to use partial rowSpec.  
dbSetRowRegionUsesPartialRowSpec(rowRegion t)
```

Related Topics

[Placement Database Access Functions](#)

dbGetPlaceRowVersion

```
dbGetPlaceRowVersion(  
    [d_cvId]  
)  
=> n_version / nil
```

Description

Returns the version of the placement row infrastructure in the given design, or returns the software version if the cellview is not specified.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

Value Returned

<i>n_version</i>	Version of the placement infrastructure used in the cellview or version of the software, when no cellview is specified.
<i>nil</i>	Returned in case of failure.

Examples

Returns the version of the placement row infrastructure in the cellview *cv*.

```
dbGetPlaceRowVersion(cv)
```

Returns the version of the software version.

```
dbGetPlaceRowVersion()
```

Related Topics

[Placement Database Access Functions](#)

dbGetRowRegionVersion

```
dbGetRowRegionVersion(  
    [d_cvId]  
)  
=> n_version / nil
```

Description

Returns the version of the rowRegion infrastructure used in the given design, or returns the software version if the cellview is not specified.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.

Value Returned

<i>n_version</i>	Version of the rowRegion infrastructure used in the cellview or version of the software, when no cellview is provided.
<i>nil</i>	Returned in case of failure.

Examples

Returns the version of the software version.

```
dbGetRowRegionVersion()
```

Returns the version of the rowRegion infrastructure in the cellview *cv*.

```
dbGetRowRegionVersion(cvId)
```

Related Topics

[Placement Database Access Functions](#)

dbIsImplicit

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

Description

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

<i>d_objectId</i>	Database ID of the object to be queried.

Value Returned

t	The object is implicit.
nil	The object is not implicit.

Examples

The object backgroundDefId is implicit.

```
dbIsImplicit(backgroundDefId)  
=> t
```

Related Topics

[Placement Database Access Functions](#)

dbIsInRowRegion

```
dbIsInRowRegion(  
    d_FigId  
)  
=> t / nil
```

Description

Checks if a figure with the given figure database ID belongs to a rowRegion.

Arguments

<i>d_figId</i>	Database ID of a figure.

Value Returned

<i>t</i>	Figure exists in or belongs to a rowRegion.
<i>nil</i>	Figure does not belong to a rowRegion.

Examples

Returns *t* because the figure belongs to the row region.

```
; 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)
```

Related Topics

[Placement Database Access Functions](#)

Topology Pattern Database Access Functions

This topic 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.

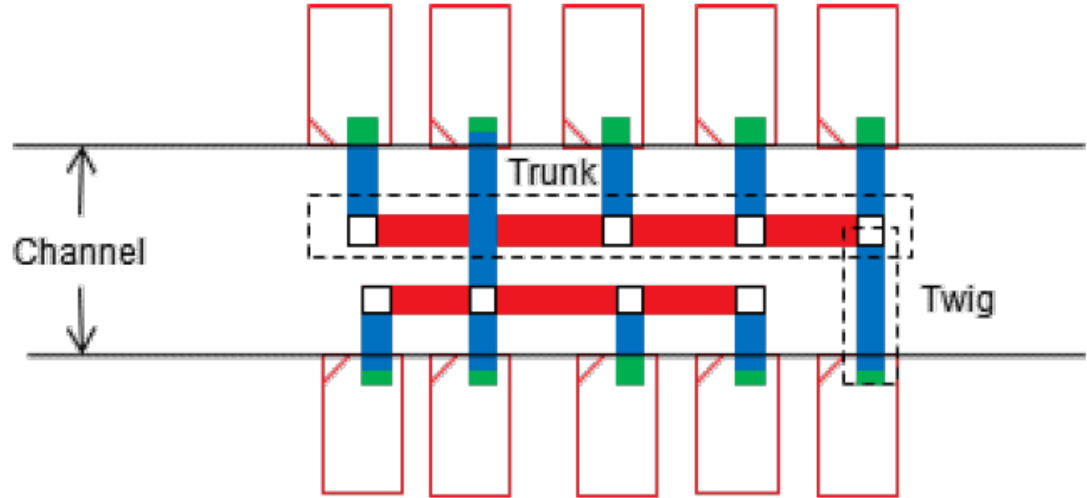
Topology Pattern	The set of rules for a channel-based router so that it automatically performs trunk-based routing. The different topology patterns are specified as follows: ■ Twig ■ Trunk ■ Trunk Connection ■ Strap
Trunk	An interconnect connection between instTerms that runs parallel to a channel.

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

Twig

An interconnect connection between instTerms that runs perpendicular to a channel and connects trunks to a different instTerm.

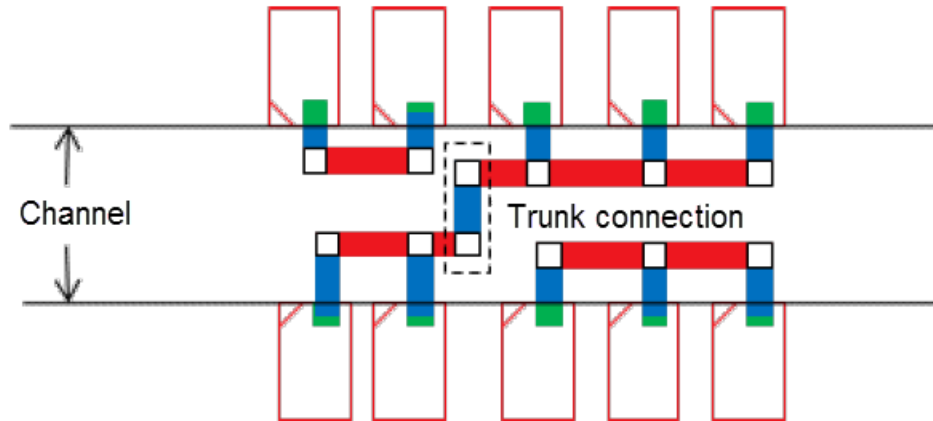


Virtuoso Studio Design Environment SKILL Reference

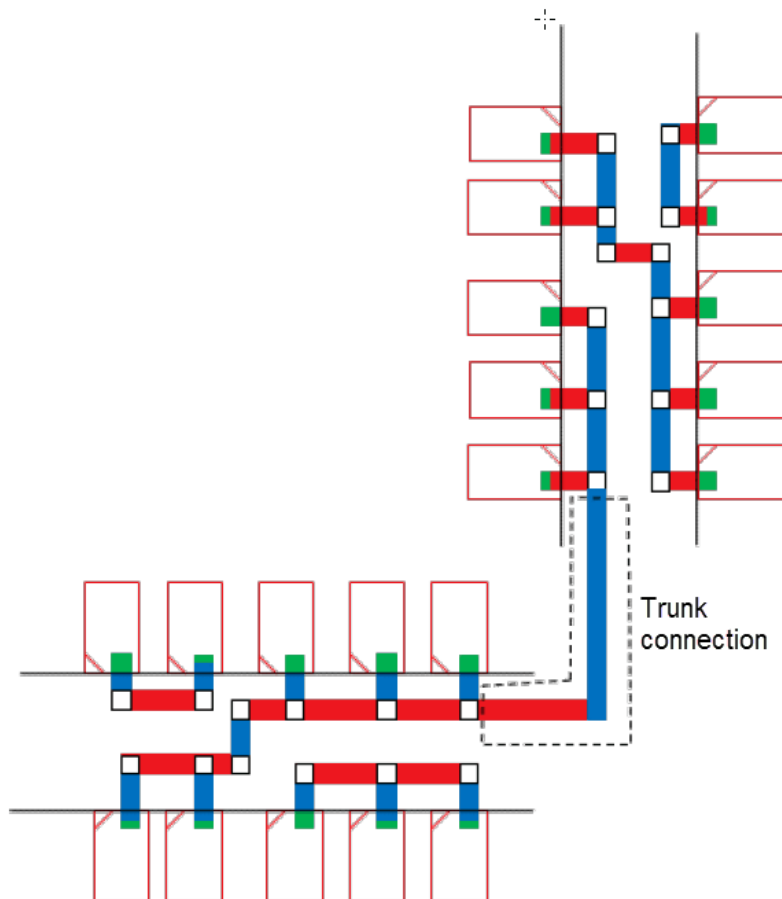
Topology Pattern Database Access Functions

Trunk
Connection

The connection between more than one trunks, also called a dogleg.



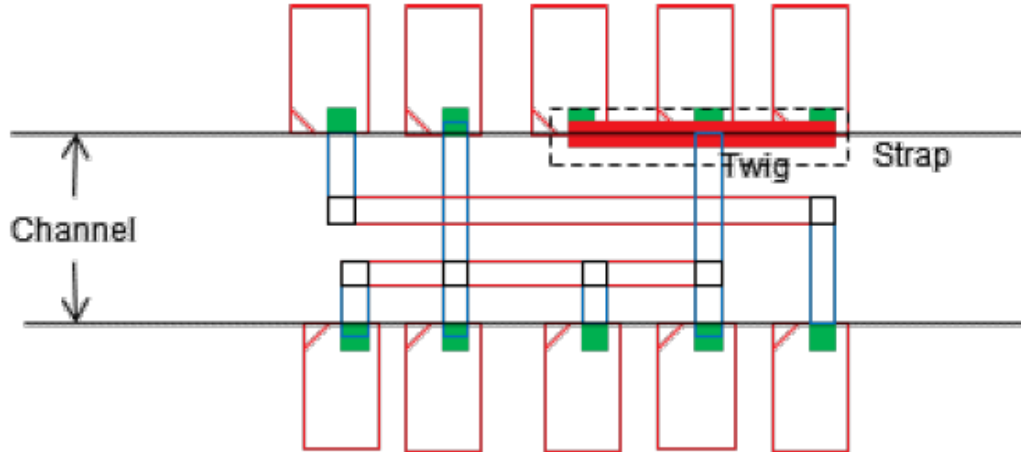
A vertical and a horizontal trunk are connected together using a trunk connection.



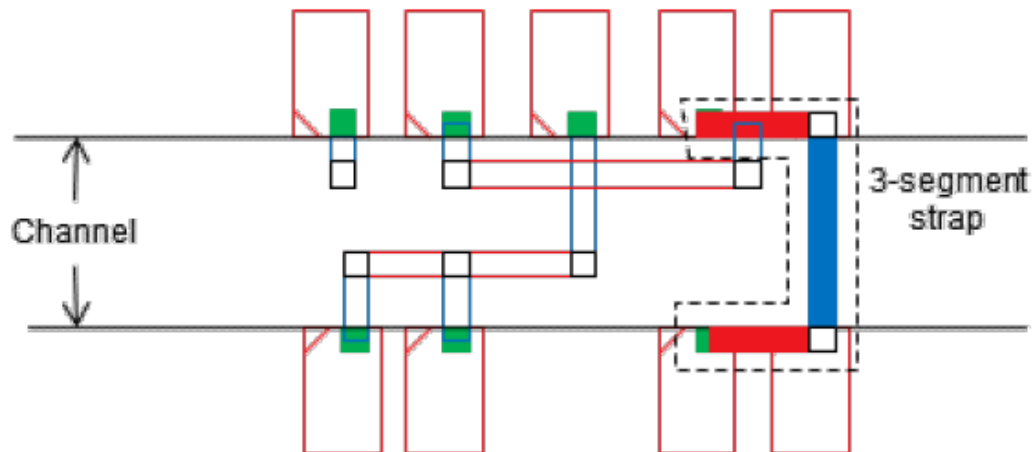
Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

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.



Following is a list of topology pattern database SKILL functions:

<u>dbCreateTopology</u>	<u>dbDeleteTopology</u>	<u>dbGetCellViewTopologies</u>
<u>dbGetNetTopologies</u>	<u>dbGetTopologyName</u>	<u>dbGetTopologyNet</u>
<u>dbGetTopologyObjects</u>	<u>dbGetTopologyPatternType</u>	<u>dbSetTopologyName</u>
<u>dbSetTopologyNet</u>	<u>dbCreateTrunk</u>	<u>dbDeleteTrunk</u>
<u>dbGetTrunkAnchor</u>	<u>dbGetTrunkDirection</u>	<u>dbGetTrunkLongOffset1</u>
<u>dbGetTrunkLongOffset2</u>	<u>dbGetTrunkLongOffsetSource</u>	<u>dbGetTrunkLongOffsetSource1</u>

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

<u>dbGetTrunkLongOffsetSource2</u>	<u>dbGetTrunkName</u>	<u>dbGetTrunkOrthoOffset</u>
<u>dbGetTrunkSide</u>	<u>dbGetTrunkTopology</u>	<u>dbGetTrunkTwigs</u>
<u>dbSetTrunkAnchor</u>	<u>dbSetTrunkLongOffset1</u>	<u>dbSetTrunkLongOffset2</u>
<u>dbSetTrunkLongOffsetSource</u>	<u>dbSetTrunkLongOffsetSource1</u>	<u>dbSetTrunkLongOffsetSource2</u>
<u>dbSetTrunkName</u>	<u>dbSetTrunkDirection</u>	<u>dbSetTrunkOrthoOffset</u>
<u>dbSetTrunkSide</u>	<u>dbCreateTwig</u>	<u>dbDeleteTwig</u>
<u>dbGetTwigName</u>	<u>dbGetTwigObject</u>	<u>dbGetTwigPinName</u>
<u>dbGetShapeTwigs</u>	<u>dbGetInstTermTwigs</u>	<u>dbGetViaTwigs</u>
<u>dbGetTwigTrunk</u>	<u>dbSetTwigName</u>	<u>dbSetTwigObject</u>
<u>dbSetTwigPinName</u>	<u>dbCreateTrunkConnect</u>	<u>dbDeleteTrunkConnect</u>
<u>dbGetTrunkConnectName</u>	<u>dbGetTrunkConnectTopology</u>	<u>dbGetTrunkConnectTrunks</u>
<u>dbSetTrunkConnectName</u>	<u>dbSetTrunkConnectTrunks</u>	<u>dbCreateStrap</u>
<u>dbDeleteStrap</u>	<u>dbGetStrapName</u>	<u>dbGetStrapObjects</u>
<u>dbGetStrapTopology</u>	<u>dbGetStrapTwig</u>	<u>dbGetInstTermStraps</u>
<u>dbGetViaStraps</u>	<u>dbGetShapeStraps</u>	<u>dbAddStrapObject</u>
<u>dbRemoveStrapObject</u>	<u>dbSetStrapName</u>	<u>dbSetStrapTwig</u>
<u>dbIsTopologyPatternId</u>	<u>dbNetHasTopology</u>	<u>dbSetStrapTwig</u>

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

<i>t_name</i>	The name of the new topology.
<i>d_netId</i>	The single-bit net ID to associate with the new topology.

Value Returned

<i>d_topologyId</i>	The ID of the new topology.
<i>nil</i>	The topology creation failed.

Examples

```
top = dbCreateTopology("top" netA)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbDeleteTopology](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

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

<i>d_topologyId</i>	The topology ID to be deleted.

Value Returned

t	The topology was deleted.
nil	The topology deletion failed.

Examples

```
dbDeleteTopology(top)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbCreateTopology](#)

dbGetCellViewTopologies

```
dbGetCellViewTopologies(  
    d_cellViewId  
)  
=> l_topologyIds / nil
```

Description

Returns a list of topologies in the specified cellview.

Arguments

<i>d_cellViewId</i>	The cellview ID whose list of topologies must be returned.

Value Returned

<i>l_topologyIds</i>	The list of topologies associated with the specified cellview.
<i>nil</i>	The list of topologies could not be returned.

Examples

```
dbGetCellViewTopologies(cv)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetNetTopologies](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

dbGetNetTopologies

```
dbGetNetTopologies(  
    d_netId  
)  
=> l_topologyIds / nil
```

Description

Returns a list of topologies associated with the specified net.

Arguments

<i>d_netId</i>	The net ID whose list of topologies must be returned.

Value Returned

<i>l_topologyIds</i>	The list of topologies associated with the specified net.
<i>nil</i>	The net does not have any associated topology pattern.

Examples

```
dbGetNetTopologies(netA)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetCellViewTopologies](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

dbGetTopologyName

```
dbGetTopologyName(  
    d_topologyId  
)  
=> t_name / nil
```

Description

Returns the name of the topology associated with the specified topology ID.

Arguments

<i>d_topologyId</i>	The topology ID whose name must be returned.

Value Returned

<i>t_name</i>	The name of the specified topology ID.
<i>nil</i>	The specified topology ID is incorrect.

Examples

```
dbGetTopologyName(top)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTopologyName](#)

dbGetTopologyNet

```
dbGetTopologyNet(  
    d_topologyId  
)  
=> t_name / nil
```

Description

Returns the net associated with the specified topology ID.

Arguments

<i>d_topologyId</i>	The topology ID whose net must be returned.

Value Returned

<i>t_name</i>	The name of the net associated with the specified topology ID.
<i>nil</i>	The specified topology ID is incorrect.

Examples

```
dbGetTopologyNet(top)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTopologyNet](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

dbGetTopologyObjects

```
dbGetTopologyObjects(  
    d_topologyId  
)  
=> l_patterns / nil
```

Description

Returns the list of trunks, trunk connects, and straps owned by the specified topology.

Arguments

<i>d_topologyId</i>	The topology ID whose list of trunks, trunk connects, and straps must be returned.

Value Returned

<i>l_patterns</i>	The list of trunks, trunk connects, and straps associated with the specified topology ID.
<i>nil</i>	The specified topology ID is incorrect.

Examples

```
dbGetTopologyObjects(top)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTopologyNet](#)

[dbGetTopologyName](#)

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

<i>d_patternId</i>	The topology pattern ID whose group type must be returned.

Value Returned

<i>t_tpGroupType</i>	The type of the topology pattern associated with the specified topology pattern ID.
<i>nil</i>	The specified topology pattern ID is incorrect.

Examples

```
dbGetTopologyPatternType(top)  
=>"topology"  
dbGetTopologyPatternType(trunkConn)  
=>"trunkConnect"  
dbGetTopologyPatternType(trunk)  
=>"trunk"  
dbGetTopologyPatternType(twig)  
=>"twig"  
dbGetTopologyPatternType(strap)  
=>"strap"
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTopologyObjects](#)

[dbGetTopologyName](#)

dbSetTopologyName

```
dbSetTopologyName(  
    d_topologyId  
    t_name  
)  
=> t / nil
```

Description

Sets the name of the specified topology ID.

Arguments

<i>d_topologyId</i>	The topology ID whose name must be changed.
<i>t_name</i>	The new name of the specified topology ID.

Value Returned

<i>t</i>	The specified new name was set for the specified topology ID.
<i>nil</i>	The specified new name was not set for the specified topology ID.

Examples

```
dbSetTopologyName(top, "newTopName")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTopologyName](#)

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

<i>d_topologyId</i>	The topology ID whose net association must be changed.
<i>d_netId</i>	The net ID to be associated with the specified topology ID.

Value Returned

<i>t</i>	The net association of the topology was changed with the specified net ID.
<i>nil</i>	The net association of the topology was not changed with the specified net ID.

Examples

```
dbSetTopologyNet(top, netB)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTopologyNet](#)

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

<i>t_name</i>	The name of the new trunk pattern.
<i>d_topologyId</i>	The topology ID on which the trunk pattern is created.
<i>t_direction</i>	The direction of the new trunk pattern. Valid direction values are <code>horizontal</code> and <code>vertical</code> .

Value Returned

<i>d_trunkId</i>	The ID of the new trunk pattern.
<code>nil</code>	The trunk pattern creation failed.

Examples

```
trunk1 = dbCreateTrunk("trunk1" top1 "horizontal")  
trunk2 = dbCreateTrunk("trunk2" top1 "vertical")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbDeleteTrunk](#)

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

<i>d_trunkId</i>	The trunk pattern ID to be deleted.

Value Returned

t	The trunk pattern was deleted.
nil	The trunk pattern deletion failed.

Examples

```
dbDeleteTrunk(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbCreateTrunk](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

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

<i>d_trunkId</i>	The trunk pattern ID whose anchor must be returned.

Value Returned

<i>d_anchorId</i>	The group type associated with the specified topology pattern ID.
<i>nil</i>	The anchor is not set for the specified topology pattern ID.

Examples

```
dbGetTrunkAnchor(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkDirection](#)

dbGetTrunkDirection

```
dbGetTrunkDirection(  
    d_trunkId  
)  
=> t_direction / nil
```

Description

Returns the direction of the specified trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose direction must be returned.

Value Returned

<i>t_direction</i>	The direction of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkDirection(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkAnchor](#)

dbGetTrunkLongOffset1

```
dbGetTrunkLongOffset1(  
    d_trunkId  
)  
=> f_longitudinalOffset1 / nil
```

Description

Returns the longitudinal offset1 value of the specified trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose longitudinal offset1 value must be returned.

Value Returned

<i>f_longitudinalOffset1</i>	The longitudinal offset1 value of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkLongOffset1(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffset2](#)

dbGetTrunkLongOffset2

```
dbGetTrunkLongOffset2(  
    d_trunkId  
)  
=> f_longitudinalOffset2 / nil
```

Description

Returns the longitudinal offset2 value of the specified trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose longitudinal offset2 value must be returned.

Value Returned

<i>f_longitudinalOffset2</i>	The longitudinal offset2 value of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkLongOffset2(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffset1](#)

dbGetTrunkLongOffsetSource

```
dbGetTrunkLongOffsetSource(  
    d_trunkId  
)  
=> t_source / nil
```

Description

Returns the longitudinal offset source value of the specified trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose longitudinal offset source must be returned.

Value Returned

<i>t_source</i>	The longitudinal offset source of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkLongOffsetSource(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTrunkLongOffsetSource](#)

[dbGetTrunkLongOffsetSource1](#)

[dbGetTrunkLongOffsetSource2](#)

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

<i>d_trunkId</i>	The trunk pattern ID of which the first longitudinal offset source must be returned.

Value Returned

<i>t_source</i>	The first longitudinal offset source of the specified trunk pattern ID.
<i>nil</i>	The specified trunk pattern ID is incorrect.

Examples

```
dbGetTrunkLongOffsetSource1(trunk1)  
=> "twigExtents"
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffsetSource](#)

[dbGetTrunkLongOffsetSource2](#)

[dbSetTrunkLongOffsetSource1](#)

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

<i>d_trunkId</i>	The trunk pattern ID of which the second longitudinal offset source must be returned.

Value Returned

<i>t_source</i>	The second longitudinal offset source of the specified trunk pattern ID.
<i>nil</i>	The specified trunk pattern ID is incorrect.

Examples

```
dbGetTrunkLongOffsetSource2(trunk1)  
=> "instanceExtents"
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffsetSource1](#)

[dbGetTrunkLongOffsetSource](#)

[dbSetTrunkLongOffsetSource2](#)

dbGetTrunkName

```
dbGetTrunkName (
    d_trunkId
)
=> t_name / nil
```

Description

Returns the name of the trunk specified by the trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose name must be returned.

Value Returned

<i>t_name</i>	The name of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkName (trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTrunkName](#)

dbGetTrunkOrthoOffset

```
dbGetTrunkOrthoOffset(  
    d_trunkId  
)  
=> n_offset / nil
```

Description

Returns the orthogonal offset of the specified trunk pattern ID. The value returned is in user units. An orthogonal offset is meaningful only when there is an anchor set for the trunk pattern.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose orthogonal offset name must be returned.

Value Returned

<i>n_offset</i>	The orthogonal offset of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkOrthoOffset(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTrunkOrthoOffset](#)

dbGetTrunkSide

```
dbGetTrunkSide(  
    d_trunkId  
)  
=> t_side / nil
```

Description

Returns the side of the trunk specified by the trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose side must be returned.

Value Returned

<i>t_side</i>	The side of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkSide(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTrunkSide](#)

dbGetTrunkTopology

```
dbGetTrunkTopology(  
    d_trunkId  
)  
=> d_topologyId / nil
```

Description

Returns the topology pattern that owns the specified trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose owner topology pattern must be returned.

Value Returned

<i>d_topologyId</i>	The topology pattern that owns the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkTopology(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetTrunkTwigs

```
dbGetTrunkTwigs(  
    d_trunkId  
)  
=> l_twigs / nil
```

Description

Returns the list of twigs associated with the specified trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose list of twigs must be returned.

Value Returned

<i>l_twigs</i>	The list of twigs of the specified trunk pattern ID.
<i>nil</i>	The trunk pattern ID is incorrect.

Examples

```
dbGetTrunkTwigs(trunk1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

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

<i>d_trunkId</i>	The trunk pattern ID whose anchor must be set.
<i>d_anchorId</i>	The anchor ID to be associated with the specified trunk pattern ID.

Value Returned

<i>t</i>	The anchor was set for the specified trunk pattern ID.
<i>nil</i>	The anchor was not set for the specified trunk pattern ID.

Examples

```
dbSetTrunkAnchor(trunk1 trunk2)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkAnchor](#)

dbSetTrunkLongOffset1

```
dbSetTrunkLongOffset1(  
    d_trunkId  
    f_longitudinalOffset1  
)  
=> t / nil
```

Description

Sets the longitudinal offset1 value for the specified trunk pattern.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose longitudinal offset1 value must be set.
<i>f_longitudinalOffset1</i>	The longitudinal offset1 value that must be set for the specified trunk pattern ID.

Value Returned

<i>t</i>	The longitudinal offset1 value was set for the specified trunk pattern ID.
<i>nil</i>	The longitudinal offset1 value was not set for the specified trunk pattern ID.

Examples

```
dbSetTrunkLongOffset1(trunk1 4.0)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffset1](#)

dbSetTrunkLongOffset2

```
dbSetTrunkLongOffset2(  
    d_trunkId  
    f_longitudinalOffset2  
)  
=> t / nil
```

Description

Sets the longitudinal offset2 value for the specified trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose longitudinal offset2 value must be set.
<i>f_longitudinalOffset2</i>	The longitudinal offset2 value that must be set for the specified trunk pattern ID.

Value Returned

<i>t</i>	The longitudinal offset2 value was set for the specified trunk pattern ID.
<i>nil</i>	The longitudinal offset2 value was not set for the specified trunk pattern ID.

Examples

```
dbSetTrunkLongOffset2(trunk1 8.1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffset2](#)

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`, and `guardRing`.

Arguments

<code>d_trunkId</code>	The trunk pattern ID whose longitudinal offset source value must be set.
<code>t_source</code>	The longitudinal offset source value that must be set for the specified trunk pattern ID.

Value Returned

<code>t</code>	The longitudinal offset source value was set for the specified trunk pattern ID.
<code>nil</code>	The longitudinal offset source value was not set for the specified trunk pattern ID.

Examples

```
dbSetTrunkLongOffsetSource(trunk1 "twigExtents")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffsetSource](#)

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

<code>d_trunkId</code>	The trunk pattern ID for which the first longitudinal offset source value must be set.
<code>t_source</code>	The first longitudinal offset source value that must be set for the specified trunk pattern ID.

Value Returned

<code>t</code>	The first longitudinal offset source value was set for the specified trunk pattern ID.
<code>nil</code>	The first longitudinal offset source value was not set for the specified trunk pattern ID.

Examples

```
dbSetTrunkLongOffsetSource1(trunk "twigExtents")  
=> t
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffsetSource1](#)

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

<code>d_trunkId</code>	The trunk pattern ID for which the second longitudinal offset source value must be set.
<code>t_source</code>	The second longitudinal offset source value that must be set for the specified trunk pattern ID.

Value Returned

<code>t</code>	The second longitudinal offset source value was set for the specified trunk pattern ID.
<code>nil</code>	The second longitudinal offset source value was not set for the specified trunk pattern ID.

Example

```
dbSetTrunkLongOffsetSource2(trunk "twigExtents")  
=> t
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkLongOffsetSource2](#)

dbSetTrunkName

```
dbSetTrunkName (
    d_trunkId
    t_name
)
=> t / nil
```

Description

Sets the name of the trunk specified by the trunk pattern ID.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose name must be set.
<i>t_name</i>	The new name of the trunk pattern ID.

Value Returned

<i>t</i>	The new name of the trunk pattern ID was set.
<i>nil</i>	The new name of the trunk pattern ID was not set.

Examples

```
dbSetTrunkName (trunk1 "tk1")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkName](#)

dbSetTrunkDirection

```
dbSetTrunkName (
    d_trunkId
    t_direction
)
=> t / nil
```

Description

Specifies whether the given trunk is running in the horizontal or vertical direction.

Arguments

<i>d_trunkId</i>	The trunk pattern ID whose direction must be set.
<i>t_direction</i>	The horizontal or vertical direction that must be set.

Value Returned

<i>t</i>	The direction of the trunk ID was set.
<i>nil</i>	The direction of the trunk ID was not set.

Examples

```
dbSetTrunkDirection(trunkId "horizontal")
dbSetTrunkDirection(trunkId "vertical")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkDirection](#)

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

<i>d_trunkId</i>	The trunk pattern ID whose orthogonal offset value must be set.
<i>n_offset</i>	The orthogonal offset value that must be set for the specified trunk pattern ID.

Value Returned

<i>t</i>	The orthogonal offset value was set for the specified trunk pattern ID.
<i>nil</i>	The orthogonal offset value was not set for the specified trunk pattern ID.

Examples

```
dbSetTrunkOrthoOffset(trunk1 2)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkOrthoOffset](#)

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

<i>d_trunkId</i>	The trunk pattern ID whose side value must be set.
<i>t_side</i>	The side value that must be set for the specified trunk pattern ID.

Value Returned

<i>t</i>	The side value was set for the specified trunk pattern ID.
<i>nil</i>	The side value was not set for the specified trunk pattern ID.

Examples

```
dbSetTrunkSide(trunk1 "top")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkSide](#)

dbCreateTwig

```
dbCreateTwig(  
    t_name  
    d_trunkId  
)  
=> d_twigId / nil
```

Description

Creates a twig pattern on the specified trunk.

Arguments

<i>t_name</i>	The name of the new twig pattern.
<i>d_trunkId</i>	The trunk ID on which the twig pattern is created.

Value Returned

<i>d_twigId</i>	The ID of the new twig pattern.
<i>nil</i>	The twig pattern creation failed.

Examples

```
twig = dbCreateTwig("twig" trunk)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbDeleteTwig](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

dbDeleteTwig

```
dbDeleteTwig(  
    d_twigId  
)  
=> t / nil
```

Description

Deletes the twig pattern.

Arguments

<i>d_twigId</i>	The twig pattern ID to be deleted.

Value Returned

<i>t</i>	The twig pattern was deleted.
<i>nil</i>	The twig pattern deletion failed.

Examples

```
dbDeleteTwig(twig)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbCreateTwig](#)

dbGetTwigName

```
dbGetTwigName (
    d_twigId
)
=> t_name / nil
```

Description

Returns the name of the specified twig pattern ID.

Arguments

<i>d_twigId</i>	The twig pattern ID whose name must be returned.

Value Returned

<i>t_name</i>	The name of the specified twig ID.
<i>nil</i>	The twig ID is incorrect.

Examples

```
dbGetTwigName (twig)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTwigName](#)

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

<i>d_twigId</i>	The twig pattern ID whose database object set must be returned.

Value Returned

<i>d_dbId</i>	The database object set of the specified twig ID.
<i>nil</i>	The twig ID is incorrect.

Examples

```
dbGetTwigObject (twig)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTwigObject](#)

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

<i>d_twigId</i>	The twig pattern ID whose pin name set must be returned.

Value Returned

<i>t_pinName</i>	The pin name set of the specified twig pattern ID.
<i>nil</i>	The twig ID is incorrect.

Examples

```
dbGetTwigPinName (twig)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTwigPinName](#)

dbGetShapeTwigs

```
dbGetShapeTwigs (  
    d_shapeId  
)  
=> l_twigs / nil
```

Description

Returns a list of twigs associated with the given shape ID.

Arguments

<i>d_shapeId</i>	The ID of the shape to query.

Value Returned

<i>l_twigs</i>	The list of twigs associated with the given shape.
<i>nil</i>	The shape is not associated with any twigs.

Examples

```
dbGetShapeTwigs (shapeId)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetInstTermTwigs

```
dbGetInstTermTwigs(  
    d_instTermId  
)  
=> l_twigs / nil
```

Description

Returns a list of twigs associated with the given instTerm ID.

Arguments

<i>d_instTermId</i>	The ID of the instTerm to query.

Value Returned

<i>l_twigs</i>	The list of twigs associated with the given instTerm.
<i>nil</i>	The instTerm is not associated with any twigs.

Examples

```
dbGetInstTermTwigs(instTermId)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetViaTwigs

```
dbGetViaTwigs(  
    d_viaId  
)  
=> l_twigs / nil
```

Description

Returns a list of twigs associated with the given via.

Arguments

<i>d_viaId</i>	The ID of the via to query.

Value Returned

<i>l_twigs</i>	The list of twigs associated with the given via.
<i>nil</i>	The via is not associated with any twigs.

Examples

```
dbGetViaTwigs(viaId)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetTwigTrunk

```
dbGetTwigTrunk(  
    d_twigId  
)  
=> d_trunkId / nil
```

Description

Returns the trunk that owns the specified twig pattern ID.

Arguments

<i>d_twigId</i>	The twig pattern ID whose trunk must be returned.

Value Returned

<i>d_trunkId</i>	The trunk ID of the specified twig pattern ID.
<i>nil</i>	The twig ID is incorrect.

Examples

```
dbGetTwigTrunk(twig)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbSetTwigName

```
dbSetTwigName(  
    d_twigId  
    t_name  
)  
=> t / nil
```

Description

Sets the name of the specified twig pattern ID.

Arguments

<i>d_twigId</i>	The twig pattern ID whose name must be set.
<i>t_name</i>	The new name of the specified twig pattern ID.

Value Returned

<i>t</i>	The new name was set for the specified twig pattern ID.
<i>nil</i>	The new name was not set for the specified twig pattern ID.

Examples

```
dbSetTwigName(twig "newTwig")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTwigName](#)

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

<i>d_twigId</i>	The twig ID whose database object must be set.
<i>d_dbId</i>	The database object ID that must be set for the specified twig ID.

Value Returned

<i>t</i>	The database object was set for the specified twig ID.
<i>nil</i>	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.

Examples

```
dbSetTwigObject(twig via)  
dbSetTwigObject(twig instTerm)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTwigObject](#)

dbSetTwigPinName

```
dbSetTwigPinName (
    d_twigId
    t_pinName
)
=> t / nil
```

Description

Sets the pin name set for the specified twig.

Arguments

<i>d_twigId</i>	The twig ID whose pin name must be set.
<i>t_pinName</i>	The pin name that must be set for the specified twig ID.

Value Returned

<i>t</i>	The pin name was set for the specified twig ID.
<i>nil</i>	The pin name was not set for the specified twig ID.

Examples

```
dbSetTwigPinName (twig "pinN")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTwigPinName](#)

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

<i>t_name</i>	The name of the trunk connect pattern.
<i>d_topologyId</i>	The topology ID on which the trunk connect pattern is created.
<i>d_startTrunkId</i>	The start trunk ID.
<i>d_endTrunkId</i>	The end trunk ID.

Value Returned

<i>d_trunkConnectId</i>	The ID of the new trunk connect pattern.
<i>nil</i>	The trunk connect pattern creation failed.

Examples

```
trunkConnect = dbCreateTrunkConnect("trunkConnect" topology startTrunk endTrunk)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbDeleteTrunkConnect](#)

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

<i>d_trunkConnectId</i>	The trunk connect pattern ID to be deleted.

Value Returned

t	The trunk connect pattern was deleted.
nil	The trunk connect pattern deletion failed.

Examples

```
dbDeleteTrunkConnect(trunkConnect)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbCreateTrunkConnect](#)

dbGetTrunkConnectName

```
dbGetTrunkConnectName(  
    d_trunkConnectId  
)  
=> t_name / nil
```

Description

Returns the name of the specified trunk connect pattern ID.

Arguments

<i>d_trunkConnectId</i>	The trunk connect pattern ID whose name must be returned.

Value Returned

<i>t_name</i>	The name of the specified trunk connect pattern ID.
<i>nil</i>	The trunk connect pattern ID is incorrect.

Examples

```
dbGetTrunkConnectName(trunkConnect)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTrunkConnectName](#)

dbGetTrunkConnectTopology

```
dbGetTrunkConnectTopology(  
    d_trunkConnectId  
)  
=> d_topologyId / nil
```

Description

Returns the owner topology of the specified trunk connect pattern ID.

Arguments

<i>d_trunkConnectId</i>	The trunk connect pattern ID whose owner topology must be returned.

Value Returned

<i>d_topologyId</i>	The owner topology of the specified trunk connect pattern ID.
<i>nil</i>	The trunk connect pattern ID is incorrect.

Examples

```
dbGetTrunkConnectTopology(trunkConnect)
```

Related Topics

[Topology Pattern Database Access Functions](#)

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

<i>d_trunkConnectId</i>	The trunk connect pattern ID whose list of associated trunks must be returned.

Value Returned

<i>l_trunkIds</i>	The list of start and end trunks of the specified trunk connect pattern ID.
<i>nil</i>	The trunk connect pattern ID is incorrect.

Examples

```
dbGetTrunkConnectTrunks(tc)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetTrunkConnectTrunks](#)

dbSetTrunkConnectName

```
dbSetTrunkConnectName(  
    d_trunkConnectId  
    t_name  
)  
=> t / nil
```

Description

Sets the name of the specified trunk connect pattern.

Arguments

<i>d_trunkConnectId</i>	The trunk connect pattern ID whose name must be set.
<i>t_name</i>	The new name of the specified trunk connect pattern ID.

Value Returned

<i>t</i>	The new name of the trunk connect pattern ID was set.
<i>nil</i>	The new name of the trunk connect pattern ID was not set.

Examples

```
dbSetTrunkConnectName(trunkConnect "newTrunkConnect")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkConnectName](#)

dbSetTrunkConnectTrunks

```
dbSetTrunkConnectTrunks(  
    d_trunkConnectId  
    d_startTrunkId  
    d_endTrunkId  
)  
=> t / nil
```

Description

Sets the start and end trunks for the specified trunk connect ID.

Arguments

<i>d_trunkConnectId</i>	The trunk connect ID whose start and end trunks must be set.
<i>d_startTrunkId</i>	The start trunk for the specified trunk connect ID.
<i>d_endTrunkId</i>	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.

Examples

```
dbSetTrunkConnectTrunks(trunkConnect startTrunk endTrunk)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetTrunkConnectTrunks](#)

dbCreateStrap

```
dbCreateStrap(  
    t_name  
    d_topologyId  
)  
=> d_strapId / nil
```

Description

Creates a strap pattern with the given name on the specified topology.

Arguments

<i>t_name</i>	The name of the new strap pattern.
<i>d_topologyId</i>	The topology ID to which the strap pattern must belong.

Value Returned

<i>d_strapId</i>	The ID of the new strap pattern.
<i>nil</i>	The strap pattern creation failed.

Examples

```
dbCreateStrap("strap1" top)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbDeleteStrap](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

dbDeleteStrap

```
dbDeleteStrap(  
    d_strapId  
)  
=> t / nil
```

Description

Deletes the specified strap pattern.

Arguments

<i>d_strapId</i>	The strap ID to be deleted.

Value Returned

t	The strap was deleted.
nil	The strap deletion failed.

Examples

```
dbDeleteStrap(strap)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbCreateStrap](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

dbGetStrapName

```
dbGetStrapName(  
    d_strapId  
)  
=> t_name / nil
```

Description

Returns the name of the specified strap pattern ID.

Arguments

<i>d_strapId</i>	The strap pattern ID whose name must be returned.

Value Returned

<i>t_name</i>	The name associated with the strap pattern ID.
<i>nil</i>	The specified strap pattern ID is incorrect.

Examples

```
dbGetStrapName(strap)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetStrapName](#)

dbGetStrapObjects

```
dbGetStrapObjects(  
    d_strapId  
)  
=> l_objects / nil
```

Description

Returns a list of database objects associated with the specified strap pattern ID.

Arguments

<i>d_strapId</i>	The strap pattern ID whose list of associated database objects must be returned.

Value Returned

<i>l_objects</i>	The list of database objects associated with the specified strap pattern ID.
<i>nil</i>	The specified strap pattern ID is incorrect.

Examples

```
dbGetStrapObjects(strap)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetStrapTopology

```
dbGetStrapTopology(  
    d_strapId  
)  
=> d_topologyId / nil
```

Description

Returns the topology pattern ID that owns the specified strap pattern.

Arguments

<i>d_strapId</i>	The strap pattern ID whose topology pattern ID must be returned.

Value Returned

<i>d_topologyId</i>	The topology pattern ID of the specified strap pattern ID.
<i>nil</i>	The specified strap pattern ID is incorrect.

Examples

```
dbGetStrapTopology(strap)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetStrapTwig

```
dbGetStrapTwig(  
    d_strapId  
)  
=> t_name / nil
```

Description

Returns the twig associated with the specified strap pattern ID.

Arguments

<i>d_strapId</i>	The strap pattern ID whose twig name must be returned.

Value Returned

<i>t_name</i>	The name of the twig associated with the specified strap pattern ID.
<i>nil</i>	The strap pattern ID does not have an associated twig.

Examples

```
dbGetStrapTwig(strap)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbSetStrapTwig](#)

dbGetInstTermStraps

```
dbGetInstTermStraps(  
    d_instTermId  
)  
=> l_straps / nil
```

Description

Returns a list of straps associated with the given instTerm ID.

Arguments

<i>d_instTermId</i>	The ID of the instTerm to query.

Value Returned

<i>l_straps</i>	The list of straps associated with the given instTerm.
<i>nil</i>	The instTerm is not associated with any straps.

Examples

```
dbGetInstTermStraps(instTermId)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetViaStraps

```
dbGetViaStraps(  
    d_viaId  
)  
=> l_straps / nil
```

Description

Returns a list of straps associated with the given via ID.

Arguments

<i>d_viaId</i>	The ID of the via to query.

Value Returned

<i>l_straps</i>	The list of straps associated with the given via.
<i>nil</i>	The via is not associated with any straps.

Examples

```
dbGetViaStraps(viaId)
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbGetShapeStraps

```
dbGetShapeStraps (
    d_shapeId
)
=> l_straps / nil
```

Description

Returns a list of straps associated with the given shape ID.

Arguments

<i>d_shapeId</i>	The ID of the shape to query.

Value Returned

<i>l_straps</i>	The list of straps associated with the given shape.
<i>nil</i>	The given shape is not associated with any straps.

Examples

```
dbGetShapeStraps (shapeId)
```

Related Topics

[Topology Pattern Database Access Functions](#)

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

<i>d_strapId</i>	The strap pattern ID to which the database object will be added.
<i>d_dbId</i>	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)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbRemoveStrapObject](#)

dbRemoveStrapObject

```
dbRemoveStrapObject(  
    d_strapId  
    d_dbId  
)  
=> t / nil
```

Description

Removes the database object from the specified strap pattern.

Arguments

<i>d_strapId</i>	The strap pattern ID from where the specified database object must be removed.
<i>d_dbId</i>	The database object ID that must be removed from the specified strap pattern ID.

Value Returned

<i>t</i>	The database object ID was removed from the specified strap pattern ID.
<i>nil</i>	The database object ID was not removed from the specified strap pattern ID.

Examples

```
dbRemoveStrapObject(strap rect1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbAddStrapObject](#)

dbSetStrapName

```
dbSetStrapName(  
    d_strapId  
    t_name  
)  
=> t / nil
```

Description

Sets the name of the specified strap pattern ID.

Arguments

<i>d_strapId</i>	The strap pattern ID whose name must be set.
<i>t_name</i>	The new name of the specified strap pattern ID.

Value Returned

<i>t</i>	The name is set for the specified strap pattern ID.
<i>nil</i>	The name was not set for the specified strap pattern ID.

Examples

```
dbSetStrapName(strap "strap2")
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetStrapName](#)

dbSetStrapTwig

```
dbSetStrapTwig(  
    d_strapId  
    t_name  
)  
=> t / nil
```

Description

Changes the twig associated with the specified strap pattern.

Arguments

<i>d_strapId</i>	The strap pattern ID whose twig association must be changed.
<i>t_name</i>	The name of the twig.

Value Returned

<i>t</i>	The twig association was changed with the specified strap pattern ID.
<i>nil</i>	The twig association was not changed with the specified strap pattern ID.

Examples

```
dbSetStrapTwig(strap twig1)
```

Related Topics

[Topology Pattern Database Access Functions](#)

[dbGetStrapTwig](#)

dbIsTopologyPatternId

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

Description

Returns a Boolean indicating whether the specified object ID is a `dbTopologyPattern`.

Arguments

<i>d_objectId</i>	The object ID whose <code>dbTopologyPattern</code> status must be returned.

Value Returned

<code>t</code>	The specified object ID is a <code>dbTopologyPattern</code> .
<code>nil</code>	The specified object ID is not a <code>dbTopologyPattern</code> .

Examples

```
dbIsTopologyPatternId(top)  
=>t  
dbIsTopologyPatternId(trunk)  
=>t  
dbIsTopologyPatternId(twig)  
=>t  
dbIsTopologyPatternId(net)  
=>nil
```

Related Topics

[Topology Pattern Database Access Functions](#)

dbNetHasTopology

```
dbNetHasTopology(  
    d_netId  
)  
=> t / nil
```

Description

Returns a Boolean indicating whether the specified net has a topology.

Arguments

<i>d_netId</i>	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.

Examples

```
dbNetHasTopology(netA)
```

Related Topics

[Topology Pattern Database Access Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Topology Pattern Database Access Functions

Generator Database Access Functions

This topic provides a list of Cadence® SKILL functions that are associated with the generator database.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- [dbStartGenAnyInstToInstTerm](#)
- [dbStartGenInstHeader](#)
- [dbStartGenInstHeaderToAnyInst](#)
- [dbStartGenLPPHeaderToShape](#)
- [dbStartGenLPToShape](#)
- [dbStartGenMarker](#)
- [dbStartGenNet](#)
- [dbStartGenNetToInstTerm](#)
- [dbStartGenNetToRoute](#)
- [dbStartGenRoute](#)
- [dbStartGenShape](#)
- [dbStartGenTerm](#)
- [dbStartGenViaHeader](#)
- [dbStartGenViaHeaderToVia](#)
- [dbStopGen](#)

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

<i>d_instId</i>	Database ID of an instance.

Value Returned

<i>I_genId</i>	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)
```


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

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_instHeaderId</i>	Database ID of an instance header.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_LPPHeaderId</i>	Database ID of an LPP header ID.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_cellViewId</i>	Database ID of a cellview.
<i>l_layerPurpose</i>	Specifies the of layer-purpose pair.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_netId</i>	Database ID of a net.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_netId</i>	Database ID of a net

Value Returned

<i>I_genId</i>	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)
```


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

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>I_genId</i>	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)
```

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

<i>d_viaHeaderId</i>	Database ID of a via header.

Value Returned

<i>I_genId</i>	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)
```

dbStopGen

```
dbStopGen(  
    I_genId  
)  
=> t / nil
```

Description

Forces a dbGenerator object reference to close and removes it from virtual memory.

Arguments

<i>I_genId</i>	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)
```

InPlace Cover Obstruction Database Access Functions

This topic provides a list of basic editing Cadence® SKILL functions associated with InPlaceCoverObstruction objects. You can use these functions to create or destroy InPlaceCoverObstruction objects and to retrieve or set InPlaceCoverObstruction attributes.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- [dbCreateInPlaceCoverObstruction](#)
- [dbDestroyInPlaceCoverObstruction](#)
- [dbGetInPlaceCoverObstructionAbstractFileName](#)
- [dbGetInPlaceCoverObstructionAbstractFilePath](#)
- [dbGetInPlaceCoverObstructionAbstractViewName](#)
- [dbGetInPlaceCoverObstructionBloat](#)
- [dbGetInPlaceCoverObstructionBlockageAttributeValue](#)
- [dbGetInPlaceCoverObstructionBlockageModel](#)
- [dbGetInPlaceCoverObstructionDoughnutHalo](#)
- [dbGetInPlaceCoverObstructionLayers](#)
- [dbGetInPlaceCoverObstructionMaxMask](#)
- [dbGetInPlaceCoverObstructionMergeDistance](#)
- [dbGetInPlaceCoverObstructionMinMask](#)
- [dbGetInPlaceCoverObstructionNeedsRemodeling](#)
- [dbGetInPlaceCoverObstructionPinCutOut](#)

Virtuoso Studio Design Environment SKILL Reference

InPlace Cover Obstruction Database Access Functions

- [dbGetInPlaceCoverObstructionPinRemodeling](#)
- [dbGetInPlaceCoverObstructionSpacingModel](#)
- [dbInPlaceCoverObstructionExists](#)
- [dbSetInPlaceCoverObstructionAbstractFileName](#)
- [dbSetInPlaceCoverObstructionAbstractFilePath](#)
- [dbSetInPlaceCoverObstructionAbstractViewName](#)
- [dbSetInPlaceCoverObstructionBloat](#)
- [dbSetInPlaceCoverObstructionBlockageAttributeValue](#)
- [dbSetInPlaceCoverObstructionBlockageModel](#)
- [dbSetInPlaceCoverObstructionDoughnutHalo](#)
- [dbSetInPlaceCoverObstructionMergeDistance](#)
- [dbSetInPlaceCoverObstructionNeedsRemodeling](#)
- [dbSetInPlaceCoverObstructionPinCutOut](#)
- [dbSetInPlaceCoverObstructionPinRemodeling](#)
- [dbSetInPlaceCoverObstructionSpacingModel](#)

dbCreateInPlaceCoverObstruction

```
dbCreateInPlaceCoverObstruction(  
    d_InstHeaderId  
    l_intRange  
)  
=> t/nil
```

Description

Creates an InPlaceCoverObstruction object on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>l_intRange</i>	Integer mask range (minMask, maxMask) for the InPlaceCoverObstruction to be created on the instance header.

Values Returned

<i>t</i>	An InPlaceCoverObstruction object is created on the specified instance header.
<i>nil</i>	The operation failed.

Examples

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.

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)  
instHeaderId = inst~>instHeader  
dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))
```

dbDestroyInPlaceCoverObstruction

```
dbDestroyInPlaceCoverObstruction(  
    d_InstHeaderId  
)  
=> t / nil
```

Description

Destroys an `InPlaceCoverObstruction` object on the given instance header.

Arguments

<code>d_InstHeaderId</code>	ID of an instance header.

Values Returned

<code>t</code>	The <code>InPlaceCoverObstruction</code> object on the given instance header is destroyed.
<code>nil</code>	The operation failed.

Examples

Destroys the `InPlaceCoverObstruction` object on the instance header `instHeaderId`.

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)  
instHeaderId = inst~>instHeader  
dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))  
dbDestroyInPlaceCoverObstruction(instHeaderId)
```

dbGetInPlaceCoverObstructionAbstractFileName

```
dbGetInPlaceCoverObstructionAbstractFileName(  
    d_InstHeaderId  
)  
=> t_fileName / nil
```

Description

Returns value of the InPlaceCoverObstruction AbstractFileName attribute for the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.

Values Returned

<i>t_fileName</i>	Valid value of the InPlaceCoverObstruction AbstractFileName attribute.
<i>nil</i>	The operation failed.

Examples

Returns *newFileName* as the InPlaceCoverObstruction AbstractFileName attribute value for the instance header *instHeaderId*.

```
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")
```

dbGetInPlaceCoverObstructionAbstractFilePath

```
dbGetInPlaceCoverObstructionAbstractFilePath(  
    d_InstHeaderId  
)  
=> t_filePath / nil
```

Description

Returns the value of the InPlaceCoverObstruction AbstractFilePath attribute for the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.

Values Returned

<i>t_filePath</i>	Value of the InPlaceCoverObstruction AbstractFilePath attribute for the specified instance header.
<i>nil</i>	The operation failed.

Examples

Returns *newFilePath* as the InPlaceCoverObstruction AbstractFilePath attribute value for the instance header *instHeaderId*.

```
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")
```

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.

Values Returned

t_viewName Value of the `AbstractViewName` attribute.
nil The operation failed.

Examples

Returns `newViewName` as the `InPlaceCoverObstruction AbstractViewName` attribute value for the instance header `instHeaderId`.

```
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")
```

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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.

Values Returned

<i>x_bloat</i>	Valid Bloat attribute values. If the operation fails, returns 0 and displays a warning message.

Examples

Returns 10 as the InPlaceCoverObstruction Bloat attribute value for the instance header *instHeaderId* on the layer *Layer1*.

```
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)
```

dbGetInPlaceCoverObstructionBlockageAttributeValue

```
dbGetInPlaceCoverObstructionBlockageAttributeValue(  
    d_InstHeaderId  
    t_layerName  
)  
=> f_blockageAttributeValue
```

Description

Returns value of the `InPlaceCoverObstructionBlockageAttributeValue` attribute from the specified instance header on the given layer.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.

Values Returned

f_blockageAttributeValue

Valid `BlockageAttributeValue` values.

If the operation fails, returns 0 and displays a warning message.

Examples

Returns 10.0 as the `InPlaceCoverObstructionBlockageAttributeValue` on the instance header `instHeaderId` on the layer `Layer1`.

```
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)
```

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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.

Values Returned

<i>t_blockageModelName</i>	Valid BlockageModel value set on the specified instance header on the given layer.
<i>nil</i>	The operation failed.

Examples

Returns "doughnut" as the InPlaceCoverObstruction BlockageModel attribute value for the instance header *instHeaderId* on the layer *Layer1*.

```
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")
```


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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.

Values Returned

<i>x_doughnutHalo</i>	Valid DoughnutHalo values. If the operation fails, returns 0 and displays a warning message.

Examples

Returns 10 as the `InPlaceCoverObstruction DoughnutHalo` attribute value for the instance header `instHeaderId` on the layer `Layer1`.

```
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)
```

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

<i>d_InstHeaderId</i>	ID of the instance header on which InPlaceCoverObstruction object is present.

Values Returned

<i>l_layerNames</i>	List of valid physical layer names.
<i>nil</i>	The operation failed.

Examples

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.

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)  
instHeaderId = inst~>instHeader  
dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))  
  
dbGetInPlaceCoverObstructionLayers(instHeaderId)
```

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.

Values Returned

x_maxMask Valid `InPlaceCoverObstruction MaxMask` values for the specified instance header.
If the operation fails, returns 0 and displays a warning message.

Examples

Returns 4 as the `InPlaceCoverObstruction MaxMask` attribute value for the instance header `instHeaderId`.

```
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)
```

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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	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.

Examples

Returns 10.0 as the `InPlaceCoverObstruction MergeDistance` attribute value for the instance header `instHeaderId` on the layer `Layer1`.

```
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)
```

dbGetInPlaceCoverObstructionMinMask

```
dbGetInPlaceCoverObstructionMinMask(  
    d_InstHeaderId  
)  
=> x_minMask
```

Description

Returns value of the InPlaceCoverObstruction MinMask attribute from the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.

Values Returned

<i>x_minMask</i>	Valid InPlaceCoverObstruction MinMask values for the specified instance header. If the operation fails, returns 0 and displays a warning message.

Examples

Returns 1 as the InPlaceCoverObstruction MinMask attribute value for the instance header *instHeaderId*.

```
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)
```

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.

Values Returned

<i>t</i>	The NeedsRemodeling attribute value for the specified instance is set to <i>t</i> .
<i>nil</i>	The NeedsRemodeling attribute value for the specified instance is set to <i>nil</i> . This is also the default value of NeedsRemodeling attribute when the InPlaceCoverObstruction is created.

Examples

Returns *t* as the InPlaceCoverObstruction NeedsRemodeling attribute value for the instance header *instHeaderId*.

```
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)
```

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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.

Values Returned

<i>t</i>	The <code>PinCutOut</code> attribute value for the specified instance is set to <i>t</i> .
<i>nil</i>	The <code>PinCutOut</code> attribute value for the specified instance is set to <i>nil</i> . This is also the default value of <code>PinCutOut</code> attribute when the <code>InPlaceCoverObstruction</code> is created.

Examples

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*.

```
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)
```

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.

Values Returned

t_remodeling Valid PinRemodeling attribute values.
nil The operation failed.

Examples

Returns "perimeterOnly" as the `InPlaceCoverObstruction PinRemodeling` attribute value for the specified instance header `instHeaderId`.

```
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")
```


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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.

Values Returned

<i>t_blockageModelName</i>	Valid <code>SpacingModel</code> values.
<code>nil</code>	The operation failed.

Examples

Returns "asIs" as the `InPlaceCoverObstruction SpacingModel` attribute value for the instance header `instHeaderId` on the layer `Layer1`.

```
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")
```

dbInPlaceCoverObstructionExists

```
dbInPlaceCoverObstructionExists(  
    d_InstHeaderId  
)  
=> t / nil
```

Description

Checks whether an `InPlaceCoverObstruction` object is present on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.

Values Returned

<i>t</i>	Returned in case the <code>InPlaceCoverObstruction</code> object is present on the specified instance header.
<i>nil</i>	Returned in case the <code>InPlaceCoverObstruction</code> object is not present on the specified instance header.

Examples

Returns `t` as the `InPlaceCoverObstruction` object is present on the instance header `instHeaderId`.

```
inst = dbCreateInst(cv master "inst" list(3 4) "R0" 1)  
instHeaderId = inst~>instHeader  
dbCreateInPlaceCoverObstruction(instHeaderId list("closed" 1 4))  
  
equal(dbInPlaceCoverObstructionExists(instHeaderId) t)
```

dbSetInPlaceCoverObstructionAbstractFileName

```
dbSetInPlaceCoverObstructionAbstractFileName(  
    d_InstHeaderId  
    t_fileName  
)  
=> t / nil
```

Description

Sets value of the `InPlaceCoverObstruction AbstractFileName` attribute on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_fileName</i>	A valid <code>AbstractFileName</code> value.

Values Returned

<i>t</i>	The specified <code>AbstractFileName</code> attribute value is set for the specified instance header.
<i>nil</i>	The operation failed.

Examples

Sets the `InPlaceCoverObstruction AbstractFileName` attribute value `newFileName` for the instance header `instHeaderId`.

```
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")
```

dbSetInPlaceCoverObstructionAbstractFilePath

```
dbSetInPlaceCoverObstructionAbstractFilePath(  
    d_InstHeaderId  
    t_filePath  
)  
=> t / nil
```

Description

Sets value of the `InPlaceCoverObstruction AbstractFilePath` attribute on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_filePath</i>	A valid <code>AbstractFilePath</code> value.

Values Returned

<i>t</i>	The specified <code>AbstractFilePath</code> attribute value is set for the given instance header.
<i>nil</i>	The operation failed.

Examples

Sets the `InPlaceCoverObstruction AbstractFilePath` attribute value `newFilePath` for the instance header `instHeaderId`.

```
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")
```

dbSetInPlaceCoverObstructionAbstractViewName

```
dbSetInPlaceCoverObstructionAbstractViewName (
    d_InstHeaderId
    t_viewName
)
=> t / nil
```

Description

Sets value of the `InPlaceCoverObstruction AbstractViewName` attribute on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_viewName</i>	A valid <code>AbstractViewName</code> value.

Values Returned

<i>t</i>	The specified <code>AbstractViewName</code> attribute value is set for the specified instance header.
<i>nil</i>	The operation failed.

Examples

Sets the `InPlaceCoverObstruction AbstractViewName` attribute value `newViewName` for the instance header `instHeaderId`.

```
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")
```

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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.
<i>n_value</i>	A valid Bloat value.

Values Returned

<i>t</i>	The specified InPlaceCoverObstruction Bloat attribute value is set on the given layer.
<i>nil</i>	The operation failed.

Examples

Sets the InPlaceCoverObstruction Bloat attribute value 10 for the instance header *instHeaderId* on the layer "Layer1".

```
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)
```

dbSetInPlaceCoverObstructionBlockageAttributeValue

```
dbSetInPlaceCoverObstructionBlockageAttributeValue(  
    d_InstHeaderId  
    t_layerName  
    n_value  
)  
=> t / nil
```

Description

Sets value of the `InPlaceCoverObstructionBlockageAttributeValue` attribute on the specified instance header on the given layer.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.
<i>n_value</i>	Value of <code>BlockageAttributeValue</code> attribute to be set.

Values Returned

<i>t</i>	The specified <code>BlockageAttributeValue</code> is set on the given instance header.
<i>nil</i>	The operation failed.

Examples

Sets the `InPlaceCoverObstructionBlockageAttributeValue` attribute value 10.0 for the instance `instHeaderId` on the layer "Layer1".

```
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)
```

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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.
<i>t_blockageModelName</i>	Valid BlockageModel value.

Values Returned

<i>t</i>	Returned in case of success.
<i>nil</i>	The operation failed.

Examples

Sets value of the InPlaceCoverObstruction BlockageModel attribute "doughnut" on the instance header *instHeaderId* on the layer *Layer1*.

```
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")
```


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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.
<i>n_value</i>	Valid <code>DoughnutHalo</code> value to be set.

Values Returned

<i>t</i>	The specified <code>InPlaceCoverObstruction DoughnutHalo</code> attribute Value is set on the specified instance.
<i>nil</i>	The operation failed.

Example

Sets value of the `InPlaceCoverObstruction DoughnutHalo` attribute 10 on the instance header `instHeaderId` on the layer `Layer1`.

```
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)
```

dbSetInPlaceCoverObstructionMergeDistance

```
dbSetInPlaceCoverObstructionMergeDistance (
    d_InstHeaderId
    t_layerName
    n_value
)
=> t / nil
```

Description

Sets value of the `InPlaceCoverObstruction MergeDistance` attribute on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.
<i>n_value</i>	A valid <code>MergeDistance</code> value.

Values Returned

<i>t</i>	The specified <code>MergeDistance</code> attribute value is set for the specified instance header.
<i>nil</i>	The operation failed.

Examples

Sets the `InPlaceCoverObstruction MergeDistance` attribute value as 10.0 for the instance header `instHeaderId` on the layer `Layer1`.

```
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)
```

dbSetInPlaceCoverObstructionNeedsRemodeling

```
dbSetInPlaceCoverObstructionNeedsRemodeling(  
    d_InstHeaderId  
    g_value  
)  
=> t/nil
```

Description

Sets value of the `InPlaceCoverObstruction NeedsRemodeling` attribute on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>g_value</i>	Valid NeedsRemodeling value.

Values Returned

<i>t</i>	The <code>InPlaceCoverObstruction NeedsRemodeling</code> attribute value is set on the specified instance header.
<i>nil</i>	The operation failed.

Examples

Sets `InPlaceCoverObstruction NeedsRemodeling` attribute on the instance header `instHeaderId`.

```
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)
```

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

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_layerName</i>	Name of a physical layer specified in the technology file.
<i>g_value</i>	A valid PinCutOut value.

Values Returned

<i>t</i>	The specified PinCutOut attribute value is set for the specified instance header.
<i>nil</i>	The operation failed.

Examples

Sets the InPlaceCoverObstruction PinCutOut attribute value true for the instance header *instHeaderId* on the layer *Layer1*.

```
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)
```

dbSetInPlaceCoverObstructionPinRemodeling

```
dbSetInPlaceCoverObstructionPinRemodeling(  
    d_InstHeaderId  
    t_remodelingName  
)  
=> t / nil
```

Description

Sets value of the `InPlaceCoverObstruction PinRemodeling` attribute on the specified instance header.

Arguments

<i>d_InstHeaderId</i>	ID of an instance header.
<i>t_remodelingName</i>	Valid PinRemodeling attribute value.

Values Returned

<i>t</i>	The specified <code>InPlaceCoverObstruction PinRemodeling</code> attribute Value is set on the specified instance.
<i>nil</i>	The operation failed.

Examples

Sets value of the `InPlaceCoverObstruction PinRemodeling` attribute "perimeterOnly" on the instance header `instHeaderId`.

```
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")
```

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

<code>d_InstHeaderId</code>	ID of an instance header.
<code>t_layerName</code>	Name of a physical layer specified in the technology file.
<code>t_spacingModelName</code>	A valid <code>SpacingModel</code> value.

Values Returned

<code>t</code>	The <code>InPlaceCoverObstruction SpacingModel</code> attribute value is set on the specified instance header on the given layer.
<code>nil</code>	The operation failed.

Examples

Sets the `InPlaceCoverObstruction SpacingModel` attribute value "asIs" for the instance `instHeaderId` on the layer `Layer1`.

```
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")
```

Virtuoso Studio Design Environment SKILL Reference

InPlace Cover Obstruction Database Access Functions

Virtuoso Studio Design Environment SKILL Reference

InPlace Cover Obstruction Database Access Functions

Figure Database Access Functions

This topic provides a list of Cadence® Figure Database Access SKILL functions.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[FigGroup Creation and Modification Functions](#)

[Figure Creation and Retrieval Functions](#)

FigGroup Creation and Modification Functions

The following functions create and modify figGroups.

- [dbCreateFigGroup](#)
- [dbAddFigToFigGroup](#)
- [dbRemoveFigFromFigGroup](#)
- [dbGetFigGroupByName](#)

Related Topics

[Figure Creation and Retrieval Functions](#)

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

<i>d_cellView</i>	The cellview ID.
<i>t_name</i>	The name of the figGroup.
<i>g_ordered</i>	Boolean specifying order.
<i>l_origin</i>	Origin of the new instance.
<i>t_orient</i>	Specifies the orientation of the figGroup.

Value Returned

<i>d_figGroupId</i>	Returns the database ID of the figGroup created.
<i>nil</i>	An error occurred during execution.

Examples

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")
```

Related Topics

[FigGroup Creation and Modification Functions](#)

dbAddFigToFigGroup

```
dbAddFigToFigGroup(  
    d_figGroupId  
    d_figId  
)  
=> t / nil
```

Description

Adds a figure to the specified figGroup.

Arguments

<i>d_figGroup</i>	The figGroup ID.
<i>d_fig</i>	The ID of the figure to be added.

Value Returned

<i>t</i>	The figure was added successfully.
<i>nil</i>	An error occurred during execution.

Examples

Creates a figGroup.

```
figGroup1 = dbCreateFigGroup(cv "figGroup1" t 0:0 "R0")
```

Creates a figure.

```
figOfRect = dbCreateRect(cv "METAL1" list(0:0 6.3:8.7))
```

Adds the figure to the figGroup.

```
dbAddFigToFigGroup(figGroup1 figOfRect)
```

Related Topics

[FigGroup Creation and Modification Functions](#)

dbRemoveFigFromFigGroup

```
dbRemoveFigFromFigGroup(  
    d_figGroupId  
    d_figId  
)  
=> t / nil
```

Description

Removes a figure from a figGroup object.

Arguments

<i>d_figGroup</i>	The figGroup ID.
<i>d_fig</i>	The ID of the figure to be removed.

Value Returned

<i>t</i>	The figure was removed from the figGroup successfully.
<i>nil</i>	An error occurred during execution.

Examples

Removes the figure *figOfRect* from the figGroup *figGroup1*.

```
dbRemoveFigFromFigGroup(figGroup1 figOfRect)
```

Deletes a figGroup and all its members. First retrieves all the members and removes them from the figGroup and then deletes the group.

```
for(fig figGroup~>members  
dbRemoveFigFromFigGroup(fig figGroup)  
)  
dbDeleteObject(figGroup)
```

Related Topics

[FigGroup Creation and Modification Functions](#)

dbGetFigGroupName

```
dbGetFigGroupName(  
    d_cellView  
    t_name  
)  
=> d_figGroupId / nil
```

Description

Retrieves a figGroup by name.

Arguments

<i>d_cellView</i>	The cellview ID.
<i>t_name</i>	The name of the figGroup.

Value Returned

<i>d_figGroupId</i>	The figGroup ID is returned successfully.
<i>nil</i>	An error occurred during execution.

Examples

Returns the database ID of the specified figGroup.

```
dbGetFigGroupName(cv "myFigGroup")
```

Related Topics

[FigGroup Creation and Modification Functions](#)

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](#)
- [dbGetPathBoundary](#)
- [dbCheckParentChild](#)
- [dbProduceOverlap](#)
- [dbProduceOverlapInst](#)

Related Topics

[FigGroup Creation and Modification Functions](#)

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: 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.

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

Arguments

<i>d_fig</i>	Original <i>dbObject</i> of the instance to be copied.
<i>d_cellView</i>	The cellview to which the figure is copied; if <i>nil</i> , the figure is copied to the same cellview.
<i>l_transform</i>	Describes the transformation of the original figure during the copy process, such as offset, rotation, and magnification.

Value Returned

<i>d_fig</i>	A new <i>dbObject</i> .
<i>nil</i>	The figure is not copied or transformed.

Examples

```
dstFig = dbCopyFig(srcFig dstCellView list(0:0 "R90" 2.0))
```

Related Topics

[Figure Creation and Retrieval Functions](#)

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: By default, coloring information is not copied. To copy coloring information, set the *cdba_copyMPAttributes* environment variable to *t* before using *dbMoveFig*.

Arguments

<i>d_fig</i>	Original <i>dbObject</i> of the instance to be moved.
<i>d_cellView</i>	The cellview to which the figure is moved; if <i>nil</i> , the figure is moved within the same cellview.
<i>l_transform</i>	Describes the transformation of the figure during the move, such as offset, rotation, and magnification.

Value Returned

<i>d_fig</i>	The <i>dbObject</i> of the figure.
<i>nil</i>	The figure is not moved or transformed.

Examples

```
dstFig = dbMoveFig(srcFig nil list(-1:2 "R180"))
```

Related Topics

[Figure Creation and Retrieval Functions](#)

dbGetOverlaps

```
dbGetOverlaps (
    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, vias, 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 that of *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

<i>d_cellView</i>	Database ID of a cellview.
<i>l_bBox</i>	The searching box.
<i>l_layerPurposePair</i>	The layer-purpose pair. If <i>l_layerPurposePair</i> , <i>tx_layer</i> , or <i>g_value</i> is specified, the function produces vias, instances, mosaics, and shapes. If none of these is specified, the function produces only vias, instances, and mosaics.
<i>tx_layer</i>	Specifies the layer name or number.

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

<i>g_value</i>	A Boolean value where: ■ <code>t</code> indicates that the function should produce vias, instances, mosaics, and shapes. ■ <code>nil</code> indicates that the function should produce only vias, instances, and mosaics.
<i>lx_level</i>	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.
<i>g_doRowCol</i>	If set to <code>true</code>, displays row and column value, if the overlapped object is a mosaic instance.

Value Returned

<i>ld_fig</i>	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.
<i>nil</i>	An error occurred.

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.

```
figList = dbGetOverlaps(cv cv~>bBox)
```

figList finds all the shapes, mosaics and instances from the top level to the level 5.

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

```
figList = dbGetOverlaps(cv cv~>bBox t 0:5)
```

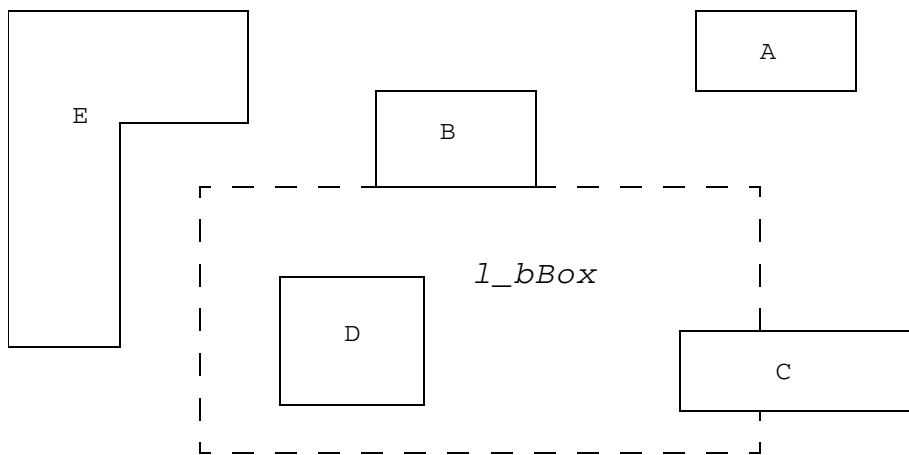
figList finds all the mosaics and instances on the top level.

```
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.



Additional Information

The SKILL functions `dbOpenHier`, `dbShapeQuery`, and `dbGetOverlaps` perform different functions and have different implementations that may traverse the hierarchy in different ways. Although it is possible to use combinations of these and other functions to get similar results, code that is sensitive to the order of traversal may produce different results. For example, code that is traversing the hierarchy through `dbOpenHier` and `dbShapeQuery`, which needs to expand through Pcells, may produce a result that is different from `dbGetOverlaps`.

if there is a Pcell evaluation failure at some point of the traversal because these functions cannot guarantee order and it cannot be predicted whether or when a Pcell will fail to evaluate, some data that is produced by one set of SKILL functions may not be produced by a different set of SKILL functions.

Related Topics

[dbGetOverlaps](#)

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

dbGetTrueOverlaps

dbOpenHier

dbShapeQuery

Figure Creation and Retrieval Functions

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, vias, 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.

l_layerPurposePair

The layer purpose pair, followed by an optional list of start and stop levels (*lx_level*) and optional rowCol boolean (*g_doRowCol*).

If *l_layerPurposePair*, *tx_layer*, or *g_value* is specified, the function produces vias, instances, mosaics, and shapes. If none of these is specified, the function produces only vias, instances, and mosaics.

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

<i>tx_layer</i>	A layer name or number, followed by an optional list of start and stop levels (<i>lx_level</i>) and optional rowCol boolean (<i>g_doRowCol</i>).
<i>g_value</i>	A Boolean value where: ■ <code>t</code> indicates that the function should produce vias, instances, mosaics, and shapes. ■ <code>nil</code> indicates that the function should produce only vias, instances, and mosaics.
<i>lx_level</i>	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.
<i>g_doRowCol</i>	If set to <code>true</code> , displays row and column value, if the overlapped object is a mosaic instance.

Value Returned

<i>ld_fig</i>	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.
<i>nil</i>	An error occurred.

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)
```


Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

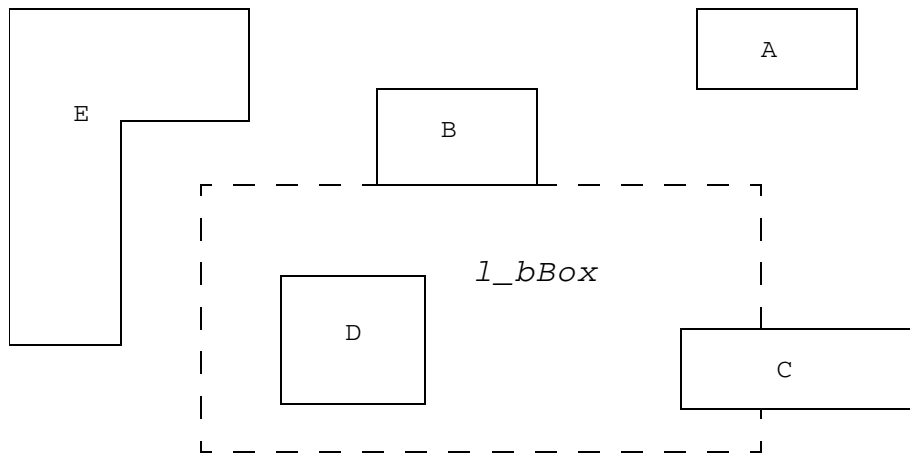
figList finds all the shapes, mosaics, and instances on the top level.

```
figList = dbGetTrueOverlaps(cv cv~>bBox)
```

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 environment variable.

Related Topics

[dbGetOverlaps](#)

[noDetailedRow](#)

[Figure Creation and Retrieval Functions](#)

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

<i>d_cellViewId</i>	Cellview ID.
<i>l_bBox</i>	Specifies a bounding box that defines the search area.
<i>t_direction</i>	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.
<i>t_layer</i>	Layer specification with drawing purpose. Only the bounding boxes of the shapes on this layer-drawing pair are checked against the searching box.
<i>t_purpose</i>	Layer-purpose pair specification. Only the bounding boxes of the shapes on this layer-purpose pair are checked against the searching box.
<i>x_depth</i>	The depth hierarchy up to which the search needs to be performed. If <i>x_depth</i> 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:

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

- ❑ 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.

Examples

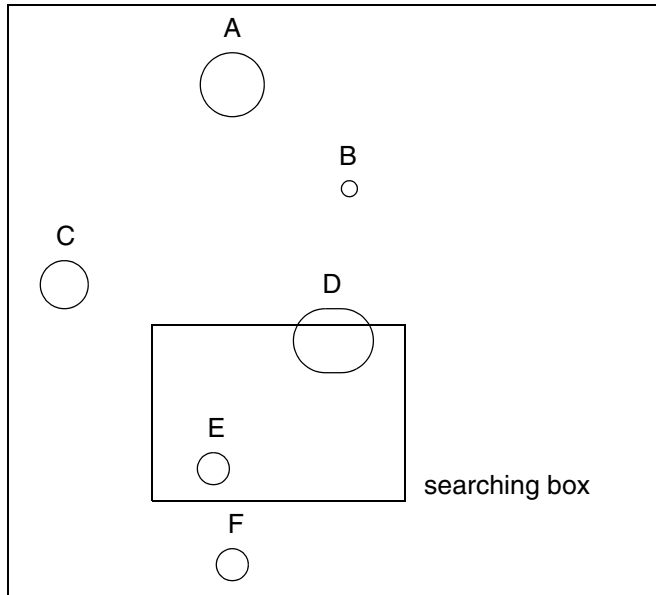
Searches for the closest neighboring shape above the searching box and returns the information about B because B is the closest shape above the specified searching box.

```
closestID = dbGetNeighbor(cell3 ((10 5) (25 12)) "T")
```

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

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.



Related Topics

[Figure Creation and Retrieval Functions](#)

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

<i>d_cellView</i>	Cellview.
<i>l_area</i>	Specifies a bounding box that defines the search area.
<i>t_direction</i>	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.
<i>t_layer</i>	Specifies layer with the drawing purpose. Only the bounding boxes of the shapes on this layer-drawing pair are checked against the searching box.
<i>t_purpose</i>	Specifies layer-purpose pair. Only the bounding boxes of the shapes on this layer-purpose pair are checked against the searching box.
<i>lx_depth</i>	Search level, that is, the depth hierarchy up to which the search needs to be performed. If <i>lx_depth</i> 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,

- ❑ If *t_purpose* is not specified, the (t_layer, drawing) layer purpose pair is used.

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

- ❑ 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

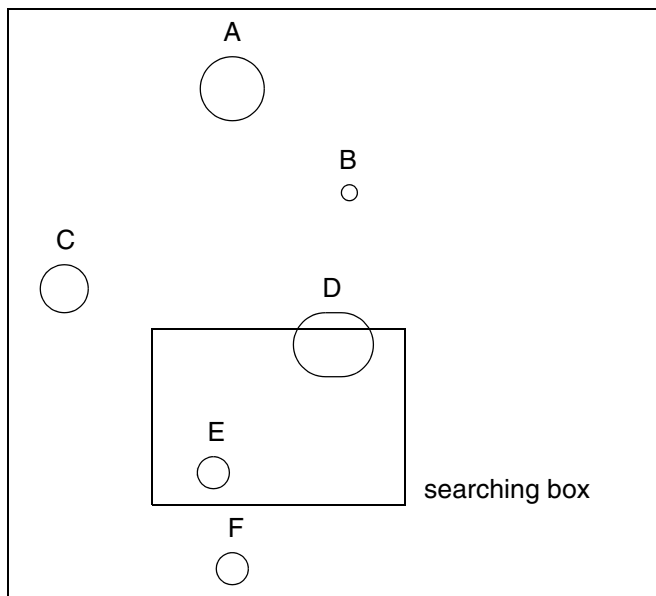
`l_closestNeighbor` A list of `ld_closestNeighbor` (in the search direction). The list is in no specific order, but you can put the list into order of increasing distance with a sorting function.

Examples

Searches for the closest neighboring shapes that are above the searching box.

```
closestIDs = dbGetNeighborList(cell3 ((10 5) (20 12)) "T")
```

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.



Related Topics

[dbGetNeighbor](#)

[Figure Creation and Retrieval Functions](#)

dbGetMaxHierDepth

```
dbGetMaxHierDepth(  
    )  
=> x_maxHierDepth
```

Description

Returns the `dbcMaxHierDepth` constant in SKILL.

Arguments

None

Value Returned

x_maxHierDepth The `dbcMaxHierDepth` constant.

Related Topics

[Figure Creation and Retrieval Functions](#)

dbGetPathSegStyle

```
dbGetPathSegStyle(  
    d_pathSeg  
)  
=> l_pathSegStyle
```

Description

Returns the path segment end styles for the supplied pathSeg database object.

Arguments

<i>d_pathSeg</i>	Specifies the database identifier of the path segment object.

Value Returned

<i>l_pathSegStyle</i>	List of lists containing the following path seg style parameters: (<i>n_width</i> <i>t_beginStyle</i> <i>t_endStyle</i> [<i>n_beginExt</i> <i>n_endExt</i> [<i>n_beginLeftDiagExt</i> <i>n_beginRightDiagExt</i> <i>n_beginRightHalfWidth</i> <i>n_endLeftDiagExt</i> <i>n_endRightDiagExt</i> <i>n_endRightHalfWidth</i>]])

Examples

```
dbGetPathSegStyle(d_pathSeg)  
(1.0 "custom" "custom"  
    (0.5 0.5  
      (0.71 0.71 0.5 0.71 0.71  
        0.5  
      )  
    )  
)
```


Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

Related Topics

[Figure Creation and Retrieval Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

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.

Examples

```
dbSetPathSegStyle(p2 1.0 "custom" "custom" list(0.5 0.5 list(0.71 0.71 0.5 0.71  
0.71 0.5)))
```

Related Topics

[Figure Creation and Retrieval Functions](#)

dbGetPathBoundary

```
dbGetPathBoundary(  
    d_path  
)  
=> l_points / nil
```

Description

Returns the polygon boundary for the specified path.

Arguments

<i>d_path</i>	Database ID of a path.

Value Returned

<i>l_path</i>	List of vertices for the polygon boundary.
<i>nil</i>	Returned in case of failure.

Examples

```
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)  
)
```

Related Topics

[Figure Creation and Retrieval Functions](#)

dbCheckParentChild

```
dbCheckParentChild(  
    d_cellViewId  
    [ g_remove ]  
)  
=> t /nil
```

Description

Checks the specified cellview for parent-child groups with no child members. The function optionally lets you remove these groups from the cellview.

Parent-child groups with no child members can result from different edit actions. For example, they can get created if you delete the markers created in layout or schematic explicitly or after fixing the related issue.

Arguments

<i>d_cellViewId</i>	Database ID of the cellview to check.
<i>g_remove</i>	A Boolean value that when set to <i>t</i> , removes any parent-child group with no child members. The default is <i>nil</i> , which means that no parent-child groups are removed.

Value Returned

<i>t</i>	The cellview has no parent-child groups without a child member. When <i>g_remove</i> is specified, the parent-child groups that have no child member are removed.
<i>nil</i>	The operation failed.

Examples

In this example, the parent-child group is created by setting and unsetting the parent attribute on a database ID object.

Create a wire and a label associated with the wire

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

```
line = dbCreateLine(cellviewId ("wire" "drawing") list(0:0 20:0))
label = dbCreateLabel(cellViewId ("text" "drawing") list(4 0) "Line from 0 to 20"
        "centerCenter" "R0" "stick" 0.5)
label~>parent = line
```

Here, `line` is the parent and `label` is the child. By removing the child, `label`, from the parent, `line`, the parent-child group still exists but it no longer contains any child members.

```
label~>parent = nil
dbCheckParentChild(cellview)
*WARNING* (DB-220838): dbCheckParentChild: Found parent/child group on parent
object 'line' with no child member.
```

Related Topics

Figure Creation and Retrieval Functions

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

<i>d_cellView</i>	The cellview.
<i>l_bBox</i>	Searching box.
<i>lx_level</i>	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.
<i>l_layerPurposePair</i>	Layer purpose pair. Defaults to <code>all</code> if not specified.

Value Returned

<i>ld_shape</i>	A list identifying all the overlapping shapes.
<code>nil</code>	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)
```

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

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")
```

Related Topics

Figure Creation and Retrieval Functions

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

<i>d_cellView</i>	The cellview.
<i>l_bBox</i>	Searching box.
<i>lx_level</i>	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

<i>ld_inst</i>	A list identifying all the overlapping instances.
<i>nil</i>	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)
```


Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

Related Topics

[Figure Creation and Retrieval Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Figure Database Access Functions

Figure Set Database Access Functions

This topic provides a list of Cadence® Figure Set Database Access SKILL functions.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[FigSet Functions](#)

[FigSetGroup Functions](#)

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.

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`

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

- [txGetCellViewFigSets](#)
- [txGetFigSetBBox](#)
- [txIsFigSet](#)
- [txMakeFigSet](#)
- [txMoveFigSet](#)
- [txPrependObjectToFigSet](#)
- [txRemoveFigSetData](#)
- [txRemoveObjectFromFigSet](#)

Related Topics

[FigSetGroup Functions](#)

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

<i>d_cellView</i>	ID of the cellview.
<i>t_name</i>	Name of the FigSet.

Value Returned

<i>T_figSet</i>	ID of the created FigSet.
<i>nil</i>	Returned in case of failure.

Examples

Creates an empty FigSet with the name `figset0` in the cellview `cv`.

```
txCreateFigSet(cv "figset0")  
> tx002af68cf
```

Related Topics

[FigSet Functions](#)

txGetFigSetData

```
txGetFigSetData(  
    T_figSet  
    n_entry  
)  
=> g_object / nil
```

Description

Returns the n^{th} object from the specified FigSet. This object is either a dbObject (CDB figure) or txObject (a FigSet or a transient shape).

Arguments

<i>T_figSet</i>	ID of the FigSet.
<i>n_entry</i>	Index of the object to return.

Value Returned

<i>g_object</i>	The n^{th} object in the specified FigSet.
<i>nil</i>	Returned in case of failure.

Examples

Returns the first and the second object from the FigSet *fs*, respectively.

```
id = txGetFigSetData(fs 0)  
> tx@0x1fc106d0  
id~>objType  
> "FigSet"  
id = txGetFigSetData(fs 1)  
> db:0x27a9dc1a  
id~>objType  
> "polygon"
```

Related Topics

FigSet Functions

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

<i>T_figSet</i>	ID of the FigSet.
<i>g_figObject</i>	ID of the appended object.

Value Returned

<i>t</i>	The object was appended.
<i>nil</i>	Returned in case of failure.

Examples

Appends a FigSet object (*fs0*) and a CDB object (a rectangle) to the FigSet *fs12*.

```
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
```

Related Topics

FigSet Functions

txClearFigSet

```
txClearFigSet(  
    T_figSet  
)  
=> t / nil
```

Description

Clears the contents of the specified FigSet.

Arguments

<i>T_figSet</i>	ID of the FigSet.

Value Returned

<i>t</i>	The contents were cleared.
<i>nil</i>	Returned in case of failure.

Examples

Clears the contents of the FigSet *fs1*.

```
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
```

Related Topics

FigSet Functions

txCloneFigSet

```
txCloneFigSet(  
    T_figSet1  
    T_figSet2  
)  
=> t / nil
```

Description

Makes FigSet1 a clone of FigSet2 by replacing the contents of FigSet1 with 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

<i>T_figSet1</i>	ID of the target FigSet.
<i>T_figSet2</i>	ID of the source FigSet.

Value Returned

<i>t</i>	The cloning is successful.
<i>nil</i>	Returned in case of failure.

Examples

Makes *fs_dest* a clone of *fs_src*.

```
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"
```

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

Related Topics

[FigSet Functions](#)

txConcatFigSet

```
txConcatFigSet(  
    T_figSet1  
    T_figSet2  
)  
=> t / nil
```

Description

Appends the contents of FigSet2 to the end of FigSet1.

Arguments

<i>T_figSet1</i>	ID of the target FigSet.
<i>T_figSet2</i>	ID of the source FigSet.

Value Returned

t	The contents were appended successfully.
nil	Returned in case of failure.

Examples

Appends the contents of `fs_dest` to the end of `fs_src`.

```
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"
```

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

Related Topics

[FigSet Functions](#)

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

<i>T_srcFigSet</i>	ID of the source FigSet.
<i>l_transform</i>	<code>list(g_offset t_transform [f_scale])</code> A transformation object. where, ■ <i>g_offset</i>: <code>list(f_offsetX f_offsetY)</code> An offset vector. ■ <i>t_transform</i> 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'. ■ <i>f_scale</i> A scale ratio (optional). Default value is 1.0.

Value Returned

<i>T_newFigSet</i>	ID of the created FigSet.
<i>nil</i>	Returned in case of failure.

Examples

Copies the source FigSet `fs0` and applies to it the `R90` transform to create a new FigSet `fs1`.

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

```
fs0 = txCreateFigSet(cv "fs0")
> 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)
> )
```

Related Topics

FigSet Functions

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

<i>d_cellView</i>	ID of the cellview.
<i>t_name</i>	Name of the FigSet.

Value Returned

<i>T_figSet</i>	ID of the found FigSet.
<i>nil</i>	Returned in case of failure.

Examples

Searches for FigSet FS_0 in the cellview cv.

```
txFindFigSet(cv "FS_0")  
> tx002af68ce
```

Related Topics

[FigSet Functions](#)

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

<i>T_figSet</i>	ID of the FigSet.
<i>g_deep</i>	A flag that determines whether the FigSets referenced by the specified FigSet should be deleted.

Value Returned

<i>t</i>	The specified FigSet was deleted successfully.
<i>nil</i>	Returned in case of failure.

Examples

Deletes the FigSet *fs1* and the FigSet referenced by it, *fs0*, from the cellview *cv*.

```
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
```

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

Related Topics

[FigSet Functions](#)

txGetCellViewFigSets

```
txGetCellViewFigSets(  
    d_cellView  
)  
=> l_figSets / nil
```

Description

Returns the list of FigSets in the specified cellview.

Arguments

<i>d_cellView</i>	ID of the cellview.

Value Returned

<i>l_figSets</i>	List of FigSets.
<i>nil</i>	Returned in case of failure.

Examples

Returns the FigSets, tx@0x19d35cf0 and tx@0x19a46350, in the cellview cv.

```
txGetCellViewFigSets(cv)  
> (tx@0x19d35cf0 tx@0x19a46350)
```

Related Topics

[FigSet Functions](#)

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

<i>T_figSet</i>	ID of the FigSet.
<i>l_onlyTheseLayers</i>	<code>list([t_layerName n_layerNum] ...)</code> A list of layer names or layer numbers. If specified, only the layers specified in the list are considered during the bounding box calculation.
<i>l_onlyTheseLPPs</i>	<code>list([l_LPP] ...)</code> 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,
	<code>l_LPP: list([t_layerName n_layerNum] t_purposeName)</code> A layer-purpose pair.
<i>l_notTheseLayers</i>	<code>list([t_layerName n_layerNum] ...)</code> A list of layer names or layer numbers. If specified, the layers specified in the list are excluded from the bounding box calculation.

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

<i>l_notTheseLPPs</i>	<code>list([l_LPP] ...)</code> A list of layer-purpose pairs. If specified, the layer-purpose pairs specified in the list are excluded from the bounding box calculation.
<i>l_onlyTheseTypes</i>	<code>list(t_typeName ...)</code> 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.
<i>l_notTheseTypes</i>	<code>list(t_typeName ...)</code> 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

<i>l_bBox</i>	Bounding box for the specified objects.
<i>nil</i>	Returned in case of failure.

Examples

Calculates the bounding box for the objects in the FigSet *fs0*.

```
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 of type *rect* in the FigSet *fs0*.

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

```
txGetFigSetBBox(fs0 ?onlyTheseTypes '("rect"))  
> ((0.0 0.0)  
>      (100.0 100.0)  
> )
```

Related Topics

[FigSet Functions](#)

txIsFigSet

```
txIsFigSet(  
    g_object  
)  
=> t / nil
```

Description

Determines whether the specified object is a FigSet object.

Arguments

<i>g_object</i>	ID of the object.

Value Returned

t	The object is a FigSet object.
nil	Returned in case of failure.

Examples

Confirms that object `fs0` is a FigSet object.

```
txIsFigSet(fs0)  
> t
```

Related Topics

[FigSet Functions](#)

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

<i>d_cellView</i>	ID of the cellview.
<i>d_figSetGroup</i>	ID of the FigSetGroup.

Value Returned

<i>T_figSet</i>	ID of the created FigSet.
<i>nil</i>	Returned in case of failure.

Examples

Creates a FigSet *fs* from the FigSetGroup *group0* in the cellview *cv*.

```
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
```


Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

```
fs = txMakeFigSet(cv group0)
> tx@0x2af32df0
txGetFigSetData(fs 0)~>name
> "figset1"
txGetFigSetData(fs 1)~>name
> "figset2"
```

Related Topics

[FigSet Functions](#)

txMoveFigSet

```
txMoveFigSet (  
    T_figSet  
    l_transform  
)  
=> t / nil
```

Description

Moves each figure in the given FigSet using the specified transform.

Arguments

<i>T_figSet</i>	ID of the FigSet.
<i>l_transform</i>	<code>list(g_offset t_transform [f_scale])</code> A transformation object. where, ■ <code>g_offset: list(f_offsetX f_offsetY)</code> An offset vector. ■ <code>t_transform</code> 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. ■ <code>f_scale</code> A scale ratio (optional). Default value is 1.0.

Value Returned

<i>t</i>	The figures were transformed successfully.
<i>nil</i>	Returned in case of failure.

Examples

Transforms the figures in the FigSet `fs0` by the transformation value of `R90`.

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

```
fs0 = txCreateFigSet(cv "fs0")
> tx@0x254996a0
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)
> )
```

Related Topics

FigSet Functions

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

<i>T_figSet</i>	ID of the FigSet.
<i>g_figObject</i>	ID of the object.

Value Returned

<i>t</i>	The object was inserted.
<i>nil</i>	Returned in case of failure.

Examples

Inserts the object *fs0* in the beginning of the FigSet *fs12*.

```
fs12 = txCreateFigSet(cv "fs12")  
> tx@0x19d35cf0  
txAppendObjectToFigSet(fs12 txCreateFigSet(cv "fs1"))  
> t  
txPrependObjectToFigSet(fs12 txCreateFigSet(cv "fs0"))  
> t  
txGetFigSetData(fs12 0)~>name  
> "fs0"
```

Related Topics

FigSet Functions

txRemoveFigSetData

```
txRemoveFigSetData(  
    T_figSet  
    n_entry  
)  
=> t / nil
```

Description

Removes the n^{th} object from the specified FigSet.

Arguments

<i>T_figSet</i>	ID of the FigSet.
<i>n_entry</i>	The index of the object to remove.

Value Returned

<i>t</i>	The object was removed successfully.
<i>nil</i>	Returned in case of failure.

Examples

Removes the object at the first position from the FigSet *fs12*.

```
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"
```

Related Topics

FigSet Functions

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

<i>T_figSet</i>	ID of the FigSet.
<i>g_object</i>	ID of the object.

Value Returned

<i>t</i>	The object was removed successfully.
<i>nil</i>	Returned in case of failure.

Examples

Removes the object *fs0* from the FigSet *fs1*.

```
fs0 = txCreateFigSet(cv "fs0")  
> tx@0x19d35cf0  
fs1 = txCreateFigSet(cv "fs1")  
> tx@0x19a46350  
txAppendObjectToFigSet(fs1 fs0)  
> t  
txRemoveObjectFromFigSet(fs1 fs0)  
> t
```

Related Topics

FigSet Functions

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.

Use of the SKILL Operator ~> with FigGroups

You can use the SKILL Operator ~> to query the design for FigGroups. Here are some examples.

To check if the specified instance or database ID is in the FigGroup, use the following command:

```
fg = inst~>figGroup
```

If the output is not `nil`, it means that the specified instance or database ID already exists in the FigGroup.

To check the type of the FigGroup, use the following command:

```
fg~>type
```

Valid values for this command are "syncClone", "modgen", or "none".

Use the following command to check what the FigGroup contains. For example, you might want to check whether it contains instances, or only shapes.

```
fg~>members
```

Suppose the FigGroup contains instances, you can further query based on your requirements in the following way:

```
insts = setof(x fg~>members x~>objType == "inst")
```

Supported functions include:

- [dbMakeFigSetGroup](#)
- [dbFindFigSetGroup](#)
- [dbFindFigSetGroupByName](#)

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

- [dbGetFigSetGroupName](#)
- [dbGetCellViewFigSetGroups](#)
- [dbIsFigSetGroup](#)
- [dbDeleteFigSetGroup](#)

Related Topics

[FigSet Functions](#)

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

<i>d_cellview</i>	ID of the cellview.
<i>T_figSet</i>	ID of the FigSet.

Value Returned

<i>d_figSetGroup</i>	ID of the created FigSetGroup.
<i>nil</i>	Returned in case of failure.

Examples

Creates a FigSetGroup *fsgroup1* in the cellview *cv*.

```
fs1 = txCreateFigSet(cv "figSet1")  
> tx@0x362ed0d0  
txAppendObjectToFigSet(fs1 txCreateFigSet(cv "figSet2"))  
> t  
fsgroup1 = dbMakeFigSetGroup(cv fs1)  
> db:0x32e3f19b  
(mapcar 'dbGetFigSetGroupName dbGetCellViewFigSetGroups(cv))  
> ("figSet1" "figSet2")
```

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

Related Topics

[FigSet Functions](#)

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

<i>d_cellView</i>	ID of the cellview.
<i>T_figSet</i>	ID of the FigSet.

Value Returned

<i>d_figSetGroup</i>	ID of the FigSetGroup found.
<i>nil</i>	Returned in case of failure.

Examples

Searches for the FigSet name `figset1` in the cellview `cv`.

```
dbFindFigSetGroup(cv figset1)  
> db:0x32e3f19b
```

Related Topics

[FigSet Functions](#)

dbFindFigSetGroupByName

```
dbFindFigSetGroupByName(  
    d_cellView  
    t_name  
)  
=>d_figSetGroup / nil
```

Description

Searches for the specified FigSetGroup name in the given cellview.

Arguments

<i>d_cellView</i>	ID of the cellview.
<i>t_name</i>	Name of the FigSetGroup.

Value Returned

<i>d_figSetGroup</i>	ID of the FigSetGroup found.
<i>nil</i>	Returned in case of failure.

Examples

Searches for the FigSetGroup name `fs0` in the cellview `cv`.

```
dbFindFigSetGroupByName(cv "fs0")  
> db:0x32e3f19b
```

Related Topics

[FigSet Functions](#)

dbGetFigSetGroupName

```
dbGetFigSetGroupName(  
    d_figSetGroup  
)  
=> t_name / nil
```

Description

Returns the name of the specified FigSetGroup object.

Arguments

<i>d_figSetGroup</i>	ID of the FigSetGroup object.

Value Returned

<i>t_name</i>	Name of the FigSetGroup object.
<i>nil</i>	Returned in case of failure.

Examples

Returns the name of the `fsg1` object as `FS_0`.

```
dbGetFigSetGroupName(fsg1)  
> "FS_0"
```

Related Topics

[FigSet Functions](#)

dbGetCellViewFigSetGroups

```
dbGetCellViewFigSetGroups(  
    d_cellView  
)  
=> l_figSetGroups / nil
```

Description

Returns a list of FigSetGroups in the specified cellview.

Arguments

<i>d_cellView</i>	ID of the cellview.

Value Returned

<i>l_figSetGroups</i>	The list of FigSetGroups.
<i>nil</i>	Returned in case of failure.

Examples

Returns the FigSetGroups `db:0x32e3f19b` and `db:0x32e3f19c` in the cellview `cv`.

```
dbGetCellViewFigSetGroups(cv)  
> (db:0x32e3f19b db:0x32e3f19c)
```

Related Topics

[FigSet Functions](#)

dbIsFigSetGroup

```
dbIsFigSetGroup(  
    d_object  
)  
=> t / nil
```

Description

Determines whether the specified object is a FigSetGroup.

Arguments

<i>d_object</i>	ID of the object.

Value Returned

t	The object is a FigSetGroup.
nil	The object is not a FigSetGroup.

Examples

Confirms that the object `dbId` is not a FigSetGroup.

```
dbIsFigSetGroup(dbId)  
> nil
```

Related Topics

[FigSet Functions](#)

dbDeleteFigSetGroup

```
dbDeleteFigSetGroup(  
    d_figSetGroup  
)  
=> t / nil
```

Description

Deletes the specified FigSetGroup, including the hierarchy of FigSetGroups that may exist in this FigSetGroup.

Arguments

<i>d_figSetGroup</i>	ID of the FigSetGroup to delete.

Value Returned

t	The FigSetGroup was deleted.
nil	Returned in case of failure.

Examples

```
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
```

Related Topics

FigSet Functions

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

Virtuoso Studio Design Environment SKILL Reference

Figure Set Database Access Functions

Multi-Patterning Technology Database Access Functions

Multi-patterning technology uses two or more mask processes to manufacture a design layer. Colors are used to represent the individual masks. This topic 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](#)

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

- [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](#)

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

- [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](#)

dbCellViewAreLayerShiftsValid

```
dbCellViewAreLayerShiftsValid(  
    d_cellviewId  
)  
=> t / nil
```

Description

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

<i>d_cellviewId</i>	Specifies the cellview ID

Value Returned

<i>t</i>	Returns <i>t</i> if there is a valid per-layer shift on the instances in the cellview
<i>nil</i>	Returns <i>nil</i> if there is no valid per-layer shift on the instances in the cellview

Example

```
if( !dbCellViewAreLayerShiftsValid(cvId)  
    dbCellViewUpdateLayerShifts(cvId)  
)
```

dbCellViewClearLayerShifts

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

Description

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

<i>d_cellviewID</i>	Specifies the cellview ID

Value Returned

<i>t</i>	Returns <i>t</i> if the per-layer color shifting information is successfully removed
<i>nil</i>	Returns <i>nil</i> if the per-layer color shifting information is not removed from the specified cellview

Example

```
dbCellViewClearLayerShifts(dbId)
```

dbCellViewHasLayerShifts

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

Description

Returns a Boolean indicating whether there is layer shift information on any instance in the specified cellview.

Arguments

<i>d_cellviewID</i>	Specifies the cellview ID

Value Returned

<i>t</i>	Returns <i>t</i> if there is layer shift information on any instance in the specified cellview
<i>nil</i>	Returns <i>nil</i> if there is no layer shift information on any instance in the specified cellview

Example

```
hasShifts = dbCellViewHasLayerShifts(cvId)
```


dbCellViewUpdateLayerShifts

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

Description

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

<code>d_cellviewID</code>	Specifies the cellview ID

Value Returned

<code>t</code>	Returns <code>t</code> if per-layer color specifications are successfully updated
<code>nil</code>	Returns <code>nil</code> if per-layer color specifications are not updated

Example

```
if( !dbCellViewAreLayerShiftsValid(cvId)  
    dbCellViewUpdateLayerShifts(cvId)  
)
```

dbCellViewInitForLayerShifting

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

Description

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

<i>d_cellviewID</i>	Specifies the cellview ID

Value Returned

<i>t</i>	Returns <i>t</i> if initialization is successful.
<i>nil</i>	Returns <i>nil</i> if initialization failed.

Example

```
dbCellViewInitForLayerShifting(dbObject)
```

dbClearUnusedOccShapeColorData

```
dbClearUnusedOccShapeColorData(  
    d_cellView  
    [ ?dryRun { t | nil } ]  
    [ ?ignoreUnboundRefs { t | nil } ]  
)  
=> x_numClearedObjects / nil
```

Description

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

<i>d_cellView</i>	Database ID of the cellview from which the unused occShape coloring data will be cleared.
<i>?dryRun</i>	When set to <code>t</code> , 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: <code>nil</code>
<i>?ignoreUnboundRefs</i>	When set to <code>t</code> , clears unused occShape coloring data even if hierarchy cannot be fully opened. Default Value: <code>nil</code>

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

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)
```

dbInstGetLayerShifts

This function is no longer supported. Use [dbInstGetLayerShifts2](#) instead.

dbInstGetLayerShifts2

```
dbInstGetLayerShifts2(  
    d_instId  
)  
=> l_layerShifts / nil
```

Description

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

<i>d_instId</i>	Specifies the instance ID

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

Returns the layer-shift pairs for instance *inst*.

```
shifts = dbInstGetLayerShifts2(inst)
=> shifts = ((20 "shift1Shift") (21 "fixedMask2Shift"))
```

dbInstHasLayerShifts

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

Description

Returns a Boolean value indicating whether per-layer color shifts are specified on the instance.

Arguments

<i>d_instId</i>	Specifies instance ID

Value Returned

<i>t</i>	Returns <i>t</i> if per-layer color shifts exist on the specified instance
<i>nil</i>	Returns <i>nil</i> if per-layer color shifts do not exist on the specified instance

Example

```
hasShift = dbInstHasLayerShifts(instId)
```


dbInstSetLayerShifts

This function is no longer supported. Use [dbInstSetLayerShifts2](#) instead.

dbInstSetLayerShifts2

```
dbInstSetLayerShifts2(  
    d_instId  
    l_layerShifts  
)  
=> t / nil
```

Description

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.

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

<i>d_instId</i>	Specifies the instance ID

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

Sets the layer-shift pairs for instance `inst` on layer number 30, layer number 31, and Metal3 to `noShift`, `fixedMask2Shift`, and `shift1Shift`, respectively.

```
shiftVals = list(list(30 "noShift") list(31 "fixedMask2Shift") list("Metal3"
"shift1Shift"))
dbInstSetLayerShifts2(inst shiftVals)
```

dbInstClearLayerShifts

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

Description

Removes all per-layer color shifting information from the specified instance.

Arguments

<i>d_instId</i>	Specifies instance ID

Value Returned

<i>t</i>	Returns <i>t</i> if the per-layer color shifting information is successfully removed from the specified instance
<i>nil</i>	Returns <i>nil</i> if the per-layer color shifting information is not removed from the specified instance

Example

```
dbInstClearLayerShifts(instId)
```

dbIsShapeColoringAllowed

```
dbIsShapeColoringAllowed(  
    d_shapeID  
)  
=> t / nil
```

Description

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

<i>d_shapeID</i>	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
```

dbIsShapeColored

```
dbIsShapeColored(  
    d_shapeID  
)  
=> t / nil
```

Description

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

<code>t</code>	Returns <code>t</code> if the shape is not colored in <code>grayColor</code>
<code>nil</code>	Returns <code>nil</code> if the shape is colored in <code>grayColor</code> 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.
```

dbGetShapeColor

```
dbGetShapeColor(  
    d_shapeId  
)  
=> t_maskColor / nil
```

Description

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.
```

dbSetShapeColor

```
dbSetShapeColor(  
    d_shapeId  
    t_maskColor  
)  
=> t / nil
```

Description

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

<i>d_shapeId</i>	Specifies ID of the shape whose color is to be set. Note: To check the shape's colorability use <u>dbIsShapeColoringAllowed</u> .
<i>t_maskColor</i>	Color to be set for the shape. Valid values: <code>grayColor</code> , <code>mask<mask number>Color</code> , where <code><mask number></code> is 1, 2, and so on, up to the number of masks defined in the technology library for the layer.

Value Returned

<i>t</i>	Returns <i>t</i> if the function runs successfully.
<i>nil</i>	Returns <i>nil</i> if there is an error.

Example

```
dbSetShapeColor(coloredRect "mask1Color" ) => t  
dbGetShapeColor(coloredRect) => "mask1Color"
```


dbIsShapeColorLocked

```
dbIsShapeColorLocked(  
    d_shapeID  
)  
=> t / nil
```

Description

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
```

dbSetShapeColorLocked

```
dbSetShapeColorLocked(  
    d_shapeId  
    g_setLocked  
)  
=> t / nil
```

Description

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

<i>d_shapeId</i>	Specifies the shape ID. Note: To check the shape's colorability use <u>dbIsShapeColoringAllowed</u> .
<i>g_setLocked</i>	Specifies one of the two boolean values: <code>t</code> or <code>nil</code> . <code>t</code> to change the state to <code>locked</code> and <code>nil</code> to change the state to <code>unlocked</code> .

Value Returned

<code>t</code>	Returns <code>t</code> if the function runs successfully
<code>nil</code>	Returns <code>nil</code> if the function encounters an error during execution

Example

```
dbSetShapeColorLocked( coloredRect t ) => t  
dbGetShapeColor( coloredRect ) => "mask2Color"  
  
dbSetShapeColorLocked( coloredRect nil ) => t  
dbGetShapeColor( coloredRect ) => "mask1Color"
```

dbIsBlockageColored

```
dbIsBlockageColored(  
    d_blockageId  
)  
=> t / nil
```

Description

Indicates whether the specified blockage is colored in a color other than `grayColor`.

Arguments

<i>d_blockageId</i>	Specifies the blockage ID

Value Returned

<code>t</code>	Returns <code>t</code> if the specified blockage is not colored in <code>grayColor</code>
<code>nil</code>	Returns <code>nil</code> if the blockage is colored in <code>grayColor</code> 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.
```

dbIsBlockageColoringAllowed

```
dbIsBlockageColoringAllowed(  
    d_blockageId  
)  
=> t / nil
```

Description

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

<i>d_blockageId</i>	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
```

dbGetBlockageColor

```
dbGetBlockageColor(  
    d_blockageId  
)  
=> t_maskColor / nil
```

Description

Returns the color assigned to the given blockage. The default color type is `grayColor`.

Arguments

<i>d_blockageId</i>	Specifies the blockage ID

Value Returned

<i>t_maskColor</i>	Returns the color assigned to the blockage. Possible values include: "grayColor", "mask1Color", "mask2Color", "mask3Color", "mask4Color", "mask5Color", "mask6Color", "mask7Color", "mask8Color", "blackColor", or "multiColor"
<i>nil</i>	Returns <code>nil</code> if the function encounters an error during execution

Example

```
dbSetBlockageColor( layerBlockage "multiColor" ) => t  
dbGetBlockageColor( layerBlockage ) => "multiColor"
```

dbSetBlockageColor

```
dbSetBlockageColor(  
    d_blockageID  
    t_maskColor  
)  
=> t / nil
```

Description

Assigns color to the specified blockage.

If you try to assign color to the blockage that cannot be colored, results in an error.

Arguments

<i>d_blockageID</i>	Specifies the blockage ID
<i>t_maskColor</i>	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

<i>t</i>	Returns <i>t</i> if the function runs successfully
<i>nil</i>	Returns <i>nil</i> if the function encounters an error

Example

```
dbSetBlockageColor( areaBlockage "mask1Color" ) => nil  
dbSetBlockageColor( layerBlockage "blackColor" ) => t  
dbGetBlockageColor( layerBlockage ) => "blackColor"
```

dbIsViaColorStateLocked

```
dbIsViaColorStateLocked(  
    d_viaId  
)  
=> t / nil
```

Description

Returns a Boolean indicating whether the coloring on the via is locked. The default value is `nil`.

Arguments

<i>d_viaId</i>	Specifies the via ID

Value Returned

<code>t</code>	Returns <code>t</code> if the coloring on the via is locked
<code>nil</code>	Returns <code>nil</code> 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
```

dbSetViaColorStateLocked

```
dbSetViaColorStateLocked(  
    d_viaId  
    g_locked  
)  
=> t / nil
```

Description

Sets the color state for the specified via.

Arguments

<i>d_viaId</i>	Specifies the via ID
<i>g_locked</i>	Specifies one of the two boolean values: <code>t</code> or <code>nil</code> . <code>t</code> to change the state to <code>locked</code> and <code>nil</code> to change the state to <code>unlocked</code> .

Value Returned

<code>t</code>	Returns <code>t</code> if the coloring on the via is successfully set
<code>nil</code>	Returns <code>nil</code> if the coloring is not set or an error occurs during execution

Example

```
dbSetViaColorStateLocked( via t ) => t  
dbIsViaColorStateLocked( via ) => t  
dbSetViaCutLayerControl( via "shift1" ) => nil
```


dbIsViaColorStateLayerLocked

```
dbIsViaColorStateLayerLocked(  
    d_dbViaId  
    t_dbViaLayerType  
)  
=> t / nil
```

Description

Indicates whether the coloring for the specified individual layer in a via is locked.

Arguments

<i>d_dbViaId</i>	Database ID of a via.
<i>t_dbViaLayerType</i>	Name of a valid via layer type. Valid values: layer1, layer2, cutLayer

Value Returned

<i>t</i>	Coloring state for the specified individual layer in a via is locked.
<i>nil</i>	Coloring state for the specified individual layer in a via is not locked or if there is an error.

Example

Coloring state for the layer `layer1` in the via `via1t_m2_m1` is locked.

```
dbSetViaColorStateLayerLocked(via1t_m2_m1 "layer1" t) => t  
dbIsViaColorStateLayerLocked(via1t_m2_m1 "layer1") => t  
dbIsViaColorStateLocked(via1t_m2_m1) => t
```

dbSetViaColorStateLayerLocked

```
dbSetViaColorStateLayerLocked(  
    d_dbViaId  
    t_dbViaLayerType  
    g_setLocked  
)  
=> t / nil
```

Description

Locks and unlocks the color state for individual layers in a via.

Arguments

<i>d_dbViaId</i>	Database ID of a via.
<i>t_dbViaLayerType</i>	Name of a valid via layer type. Valid values: <code>layer1</code> , <code>layer2</code> , <code>cutLayer</code>
<i>g_setLocked</i>	Specifies the lock state for the specified layer: <code>t</code> locks the layer, <code>nil</code> unlocks the layer

Value Returned

<code>t</code>	Color state for the specified layer in the given via is changed.
<code>nil</code>	The operation failed.

Example

Color state for the layer `layer1` in the via `via1t_m2_m1` is changed to locked.

```
dbSetViaColorStateLayerLocked(via1t_m2_m1 "layer1" t) => t  
dbIsViaColorStateLayerLocked(via1t_m2_m1 "layer1") => t  
dbIsViaColorStateLocked(via1t_m2_m1) => t
```

dbGetViaLayerNumColorMasks

```
dbGetViaLayerNumColorMasks (
    d_viaId
    t_viaLayerType
)
=> t_layerName / nil
```

Description

Returns the number of supported color masks for specified layer of the given via.

Arguments

<i>d_viaId</i>	Specifies the via ID.
<i>t_viaLayerType</i>	Specifies the via layer type. Valid values: layer1, layer2, cutLayer

Value Returned

<i>n_numMasks</i>	Returns the number of supported color masks for the specified layer of the via.
<i>nil</i>	Returns <i>nil</i> 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" )
```

dbGetViaLayer1Control

```
dbGetViaLayer1Control(  
    d_viaId  
)  
=> t_colorControl / nil
```

Description

Returns the color control assigned to layer1 of the via. Default value for color control is noShift.

Arguments

<i>d_viaId</i>	Specifies the via ID

Value Returned

<i>t_colorControl</i>	Returns the color control assigned to the layer2 of the via. Valid values: noColor, noShift, shift1, shift2, shift2, shift3, shift4, shift5, shift6, or shift7.
<i>nil</i>	Returns nil if an error occurs during execution

Example

```
dbGetViaLayer1Control( dbCreateVia( cvId viaDefId '(0 0) "R0" ) )  
=> "noShift"  
  
dbGetViaLayer1Control( grayVia )  
=> "noColor"
```

dbGetViaLayer2Control

```
dbGetViaLayer2Control(  
    d_viaId  
)  
=> t_colorControl / nil
```

Description

Returns the color control assigned to layer 2 of the via. Default value for color control is noShift.

Arguments

<i>d_viaId</i>	Specifies the via ID

Value Returned

<i>t_colorControl</i>	Returns the color control assigned to the layer2 of the via. Valid values: noColor, noShift, shift1, shift2, shift2, shift3, shift4, shift5, shift6, or shift7.
<i>nil</i>	Returns <i>nil</i> if an error occurs during execution

Example

```
dbGetViaLayer2Control( dbCreateVia( cvId viaDefId '(0 0) "R0" ) )  
=> "noShift"  
  
dbGetViaLayer2Control( grayVia )  
=> "noColor"
```

dbGetViaCutLayerControl

```
dbGetViaCutLayerControl(  
    d_viaId  
)  
=> t_colorControl / nil
```

Description

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"
```

dbGetViaLayerControl

```
dbGetViaLayerControl(  
    d_dbViaId  
    t_dbViaLayerType  
)  
=> t_colorControl / nil
```

Description

Returns the color control assigned to the specified layer in the given via.

Arguments

<i>d_dbViaId</i>	Database ID of a via.
<i>t_dbViaLayerType</i>	Name of a valid via layer type. Valid values: layer1, layer2, cutLayer

Value Returned

<i>t_colorControl</i>	Returns the color control assigned to the specified layer in the given via. Valid values: noColor, noShift, shift1, shift2, shift3, shift4, shift5, shift6, or shift7.
<i>nil</i>	The operation failed.

Example

Color control shift2 is set on the layer layer2 in the via via1t_m2_m1.

```
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")
```

Returns color control for layer2 in the via via1t_m2_m1 as shift2, using the function dbGetViaLayer2Control.

```
dbSetViaLayerControl(via1t_m2_m1 "layer2" "shift2")  
=> t
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
dbGetViaLayer2Control( via1t_m2_m1 )  
=> "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.

```
dbGetViaLayerControl( via1t_m2_m1 "layer2")  
=> "shift2"
```


dbSetViaLayer1Control

```
dbSetViaLayer1Control(  
    d_viaId  
    t_colorControl  
)  
=> t / nil
```

Description

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

<i>d_viaId</i>	Specifies the via ID
<i>t_colorControl</i>	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

<i>t</i>	Returns <i>t</i> if the color controls are successfully set
<i>nil</i>	Returns <i>nil</i> 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
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

; Returns an error / nil because the color state is locked

dbSetViaLayer2Control

```
dbSetViaLayer2Control(  
    d_viaId  
    t_colorControl  
)  
=> t / nil
```

Description

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

<i>d_viaId</i>	Specifies the via ID
<i>t_colorControl</i>	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

<i>t</i>	Returns <i>t</i> if the color controls are successfully set
<i>nil</i>	Returns <i>nil</i> if the color controls are not set or an error occurs during execution

Example

Returns an error / nil because the color state is locked.

```
dbSetViaLayer2Control( via "shift2" )  
=> t  
dbSetViaColorStateLocked( via t )  
=> t  
dbSetViaLayer2Control( via "shift1" )
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

=> nil

dbSetViaCutLayerControl

```
dbSetViaCutLayerControl(  
    d_viaId  
    t_colorControl  
)  
=> t / nil
```

Description

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

<i>d_viaId</i>	Specifies the via ID
<i>t_colorControl</i>	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

<i>t</i>	Returns <i>t</i> if the color control is successfully set
<i>nil</i>	Returns <i>nil</i> if the color control is not set or an error occurs during execution

Example

Returns *nil* because the color state is locked.

```
dbSetViaCutLayerControl( via "shift2" ) => t  
dbSetViaColorStateLocked( via t ) => t  
dbSetViaCutLayerControl( via "shift1" ) => nil
```

dbSetViaLayerControl

```
dbSetViaLayerControl(  
    d_dbViaId  
    t_dbViaLayerType  
    t_colorControl  
)  
=> t / nil
```

Description

Sets the color control for the specified layer in the given via.

Arguments

<i>d_dbViaId</i>	Database ID of a via.
<i>t_dbViaLayerType</i>	Name of a valid via layer type. Valid values: <code>layer1</code> , <code>layer2</code> , <code>cutLayer</code>
<i>t_colorControl</i>	Color control that can be assigned to the specified layer in the given via. Valid values: <code>noColor</code> , <code>noShift</code> , <code>shift1</code> , and so on, up to <code>shiftx</code> , where <code>x</code> 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: <code>noColor</code> , <code>noShift</code> , <code>shift1</code> , and <code>shift2</code> .

Value Returned

<i>t</i>	Color controls for the specified layer in the given via are successfully changed.
<i>nil</i>	The operation failed.

Example

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`.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

■ Color control shift1 to the layer cutLayer.

```
cv = dbOpenCellViewByType("tech" "design" "layout" "maskLayout" "w")
viaDef_M2_M1 = techFindViaDefByName(tf "M2_M1")
vialt_m2_m1 = dbCreateVia(cv viaDef_M2_M1 10:10 "R0")

dbSetViaLayerControl(vialt_m2_m1 "layer2" "shift2")
=> t
dbSetViaLayerControl(vialt_m2_m1 "layer1" "shift1")
=> t
dbSetViaLayerControl(vialt_m2_m1 "cutLayer" "shift1")
=> t
```

dbSetViaColorInfo

```
dbSetViaColorInfo(  
    d_dbViaId  
    l_layer1ControlInfo  
    l_layer2ControlInfo  
    l_cutLayerControlInfo  
)  
=> t / nil
```

Description

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`).

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

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
; 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
```

dbIsTrackPatternColored

```
dbIsTrackPatternColored(  
    d_trackPatternId  
)  
=> t / nil
```

Description

Returns a Boolean value that indicates whether the first track of the specified track pattern is colored in a color other than gray.

To color the track pattern, use [dbSetTrackPatternFirstTrackColor](#) or [dbSetTrackPatternColorAlternating](#).

Arguments

d_trackPatternId Specifies the track pattern ID

Value Returned

<i>t</i>	Returns <i>t</i> if the track pattern is colored
<i>nil</i>	Returns <i>nil</i> if the track pattern is not colored or an error occurs during execution

Examples

Creates a track pattern, then checks for coloring on the track pattern. Newly created track patterns are not colored.

```
dbIsTrackPatternColored( dbCreateTrackPattern( cvId t 100 10 10 "M1" ) ) => nil
```

Creates a track pattern, then sets the first track color to `mask2Color`, which makes the track pattern colored.

```
tpID = dbCreateTrackPattern( cvId t 100 10 10 "M1" )  
dbSetTrackPatternFirstTrackColor( tpID "mask2Color" )  
dbIsTrackPatternColored( tpID ) => t
```

dbIsTrackPatternColorAlternating

```
dbIsTrackPatternColorAlternating(  
    d_trackPatternId  
)  
=> t / nil
```

Description

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

<i>t</i>	Returns <i>t</i> if the track pattern coloring is alternating. Note: Newly created track patterns are alternating by default, even though they are not colored.
<i>nil</i>	Returns <i>nil</i> 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
```

dbIsTrackPatternColoringAllowed

```
dbIsTrackPatternColoringAllowed(  
    d_trackPatternId  
)  
=> t / nil
```

Description

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

<i>t</i>	Returns <i>t</i> if the track pattern can be colored
<i>nil</i>	Returns <i>nil</i> 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
```

dbGetTrackPatternFirstTrackColor

```
dbGetTrackPatternFirstTrackColor(  
    d_trackPatternId  
)  
=> t_maskColor / nil
```

Description

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"
```

dbSetTrackPatternFirstTrackColor

```
dbSetTrackPatternFirstTrackColor(  
    d_trackPatternId  
    t_maskColor  
)  
=> t / nil
```

Description

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

<i>d_trackPatternId</i>	Specifies the track pattern ID
<i>t_maskColor</i>	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

<i>t</i>	Returns <i>t</i> if the color is successfully assigned to the first track
<i>nil</i>	Returns <i>nil</i> 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"
```

dbSetTrackPatternColorAlternating

```
dbSetTrackPatternColorAlternating(  
    d_trackPatternId  
    g_isAlternating  
)  
=> t / nil
```

Description

Sets the coloring type of the track pattern.

Arguments

<i>d_trackPatternId</i>	Specifies the track pattern ID
<i>g_isAlternating</i>	Specifies whether the same color should be used for all tracks in the pattern (<code>nil</code>) or the track pattern should cycle through the colors for the layer (<code>t</code>). The default is <code>t</code> .

Value Returned

<code>t</code>	Returns <code>t</code> if the specified coloring type is successfully set
<code>nil</code>	Returns <code>nil</code> if the coloring type is not set

Example

```
dbSetTrackPatternColorAlternating( trackPattern nil ) => t  
dbIsTrackPatternColorAlternating( trackPattern ) => nil
```


dbCreateSameMaskGroup

```
dbCreateSameMaskGroup(  
    d_cellView  
    t_groupName  
)  
=> d_group / nil
```

Description

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

<i>d_cellView</i>	Specifies the cellview ID
<i>t_groupName</i>	Specifies the name of a group that is to be created

Value Returned

<i>d_group</i>	Returns the ID of the diff-mask group created
<i>nil</i>	Returns <code>nil</code> if an error occurs during execution

Example

```
groupMC1 = dbCreateSameMaskGroup( cvId "MC1Group" )  
dbAddObjectToGroup( groupMC1 rectM1 )  
dbAddObjectToGroup( groupMC1 circleM3 )
```

Related Information

[dbGetSameMaskDiffMaskGroups](#)

dbCreateDiffMaskGroup

```
dbCreateDiffMaskGroup(  
    d_cellView  
    t_groupName  
    d_sameMaskGroup1  
    d_sameMaskGroup2  
)  
=> d_group / nil
```

Description

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

<i>d_cellView</i>	Specifies the cellview ID
<i>t_groupName</i>	Specifies the name of a group that is to be created
<i>d_sameMaskGroup1</i>	Specifies the first group from the pair of groups of the shapes (of different colors)
<i>d_sameMaskGroup2</i>	Specifies the second group from the pair of groups of the shapes (of different colors)

Value Returned

<i>d_group</i>	Returns the ID of the diff-mask group created
<i>nil</i>	Returns <i>nil</i> if an error occurs during execution

Example

```
groupMC1 = dbCreateSameMaskGroup( cvId "MC1Group" )  
dbAddObjectToGroup( groupMC1 rectM1 )  
dbAddObjectToGroup( groupMC1 circleM3 )  
  
groupMC2 = dbCreateSameMaskGroup( cvId "MC2Group" )
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
dbAddObjectToGroup( groupMC2 rectM2 )
```

```
groupDMC = dbCreateDiffMaskGroup( cvId "DMCGroup" groupMC1 groupMC2 )
```

```
groupMC3 = dbCreateSameMaskGroup( cvId "MC3Group" )
```

```
dbAddObjectToGroup( groupMC3 polygonM1 )
```

```
dbAddObjectToGroup( groupDMC groupMC3 )
```

dbGetSameMaskDiffMaskGroups

```
dbGetSameMaskDiffMaskGroups (  
    d_sameMaskGroupId  
)  
=> l_listofGroups / nil
```

Description

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

<i>l_listofGroups</i>	Returns a list of all groups of type 'diffMaskColor'
<i>nil</i>	Returns <i>nil</i> if no groups of type 'diffMaskColor' exist or error occurs during execution

Example

```
dbGetSameMaskDiffMaskGroups (groupId)  
=> (db:0x32fadclc)
```

dbGetShapeSameMaskGroups

```
dbGetShapeSameMaskGroups (
    d_shapeId
)
=> l_listofGroups / nil
```

Description

Accepts the object of the shape for which you want to retrieve a list of all groups of type 'sameMaskColor'.

Arguments

<i>d_shapeId</i>	Specifies the shape ID.

Value Returned

<i>l_listofGroups</i>	Returns a list of all groups of type 'sameMaskColor'
<i>nil</i>	Returns <i>nil</i> if no groups of type 'sameMaskColor' exist or error occurs during execution

Example

```
dbGetShapeSameMaskGroups (rectId)
=> (db:0x32fadclc)
```

dbColorShapeQuery

```
dbColorShapeQuery(  
    d_cellViewId  
    l_layerPurposePair  
    l_bBox  
    [ x_startLevel ]  
    [ x_stopLevel ]  
    [ n_filterSize ]  
    [ l_transform ]  
)  
=> l_shapeColorPairs / nil
```

Description

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.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Arguments

<i>d_cellviewId</i>	Specifies the cellview ID
<i>l_layerPurposePair</i>	Specifies the layer-purpose pair
<i>l_bBox</i>	Specifies the region that should overlap bounding boxes of all returned shapes
<i>x_startLevel</i>	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.
<i>x_stopLevel</i>	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 <i>x_stopLevel</i> if it encounters a cellview which is not currently loaded.
<i>n_filterSize</i>	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.
<i>l_transform</i>	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

<i>l_shapeColorPairs</i>	Returns a list of pairs (shape ID, shape color)
<i>nil</i>	Returns <i>nil</i> if an error occurs during execution

Example

```
dbColorShapeQuery( cvId '("M1" "drawing") '((0 0) (100 100)) )  
=> ((sh1Id "mask1Color") (sh2Id "grayColor"))
```

dbGetIntegrationColorModel

```
dbGetIntegrationColorModel(  
    d_cellViewId  
)  
=> t_integType / nil
```

Description

Returns a string value of the *integration color model* constraint for the specified cellview. Possible values are `any` or `locked`.

Arguments

<code>d_cellViewId</code>	Specifies the cellview ID

Value Returned

<code>t_integType</code>	Returns a string value representing the integration color model constraint for specified cellview
<code>nil</code>	Returns <code>nil</code> if an error occurs during execution

Example

```
dbGetIntegrationColorModel( cvId )  
=> "locked"
```


dbGetColorModel

```
dbGetColorModel(  
    d_cellViewId  
)  
=> t_type / nil
```

Description

Returns a string value of the *color model* constraint for the specified cellview. Possible values are any, layerBased, or locked.

Arguments

<code>d_cellViewId</code>	Specifies the cellview ID

Value Returned

<code>t_type</code>	Returns a string value representing the color model constraint for specified cellview
<code>nil</code>	Returns <code>nil</code> if an error occurs during execution

Example

```
dbSetColorModel( cvId "layerBased" ) ; setting the color model constraint  
=> t  
dbGetColorModel( cvId ) ; retrieving the color model constraint  
=> "layerBased"
```

dbSetIntegrationColorModel

```
dbSetIntegrationColorModel(  
    d_cellViewId  
    t_integType  
)  
=> t / nil
```

Description

Sets the value of the *integration color model* constraint for the specified cellview. Possible values are any or locked.

Arguments

<i>d_cellViewId</i>	Specifies the cellview ID
<i>t_integType</i>	Specifies the value of the integration <i>color model</i> constraint you want to set

Value Returned

<i>t</i>	Returns <i>t</i> if the value is successfully set
<i>nil</i>	Returns <i>nil</i> if an error occurs during execution

Example

```
dbSetIntegrationColorModel( cvId "any" ) => t
```

```
dbSetIntegrationColorModel( cvId "layerBased" ) => nil    ; no "layerBased"  
integration color model
```

dbSetColorModel

```
dbSetColorModel(  
    d_cellViewId  
    t_type  
)  
=> t / nil
```

Description

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

<i>d_cellViewId</i>	Specifies the cellview ID
<i>t_type</i>	Specifies the value of the <i>color model</i> constraint you want to set

Value Returned

<i>t</i>	Returns <i>t</i> if the value is successfully set
<i>nil</i>	Returns <i>nil</i> if an error occurs during execution

Example

```
dbSetColorModel( cvId "layerBased" )  
=> t
```

dbRemoveLayerColorData

```
dbRemoveLayerColorData(  
    d_cellViewId  
    t_layerName  
    g_locked  
    [ g_via g_hcl g_instColorShift g_blockage g_trackPattern ]  
)  
=> t / nil
```

Description

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.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Arguments

<i>d_cellViewId</i>	ID of the cellview.
<i>t_layerName</i>	Name of the layer.
<i>g_locked</i>	If <i>t</i> , removes the layer color data regardless of the locked color state. If <i>nil</i> , the locked shapes and vias on that layer are not affected.
<i>g_via</i>	(Default) If <i>t</i> , removes the via color data. If <i>nil</i> , the via color data is not affected.
<i>g_hcl</i>	(Default) If <i>t</i> , removes the hierarchical color locks. If <i>nil</i> , the hierarchical color locks are not affected.
<i>g_instColorShift</i>	(Default) If <i>t</i> , removes the layer shift information for the cellview. If <i>nil</i> , the layer shift information for the cellview is not affected.
<i>g_blockage</i>	(Default) If <i>t</i> , removes the blockage color data. If <i>nil</i> , the blockage color data is not affected.
<i>g_trackPattern</i>	(Default) If <i>t</i> , removes the track pattern color data. If <i>nil</i> , the track pattern color data is not affected.

Value Returned

<i>t</i>	Returns <i>t</i> if the color data is successfully removed.
<i>nil</i>	Returns <i>nil</i> in case of a failure.

Example

Removes all the color data from the *M1* layer of the *cv* cellview.

```
dbRemoveLayerColorData( cv "M1" t )  
> t
```

Removes all the color data from the *M1* layer of the *cv* cellview, except the track pattern color data.

```
dbRemoveLayerColorData( cv "M1" t t t t t nil )
```

dbColorShapeQuery2

```
dbColorShapeQuery2(  
    d_cellViewId  
    l_layerPurposePair  
    l_bBox  
    [ x_startLevel ]  
    [ x_stopLevel ]  
    [ n_filterSize ]  
    [ l_transform ]  
)  
=> l_occShapesWithColorAndState / nil
```

Description

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.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Arguments

<i>d_cellViewId</i>	ID of the cellview.
<i>l_layerPurposePair</i>	List of layer-purpose pairs.
<i>l_bBox</i>	List of coordinates representing a bounding box.
<i>x_startLevel</i>	The start hierarchy level for the query.
<i>x_stopLevel</i>	The stop hierarchy level for the query.
<i>n_filterSize</i>	The filter size for shapes.
<i>l_transform</i>	The transform applied on the query.

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.

Virtuoso Studio Design Environment SKILL Reference

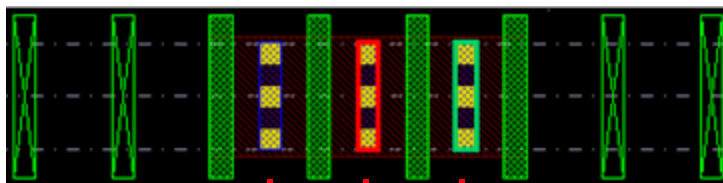
Multi-Patterning Technology Database Access Functions

- *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.

Example

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.



Metal1:drawing shape 3: mask2Color, locked, HCL

Metal1:drawing shape 2: mask1Color, locked, color attribute

Metal1:drawing shape 1: grayColor (no color)

```
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)
)
```

Converts the hierarchical path output from `dbColorShapeQuery2` using the **asNested** function to the format accepted by `dbGetInstTransform`.

```
(defun asNested (l)
  (if (cdr l) then
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
        (list (car l) (asNested (cdr l)))
    else
        1
    ) ; 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
```

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)
```

dbGetColoredOccShapes

```
dbGetColoredOccShapes (
    d_refId
    [ b_lockedOnly ]
)
=> l_occShapesWithColorAndState / nil
```

Description

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

<i>d_refId</i>	ID of an instance or a via.
<i>b_lockedOnly</i>	A Boolean value that determines whether only the locked colored occurrence shapes are returned. When set to <i>t</i>, returns only the locked colored occurrence shapes. When set to <i>nil</i>, returns all the colored occurrence shapes. Valid values are <i>t</i> and <i>nil</i>. The default value is <i>nil</i>.

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.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

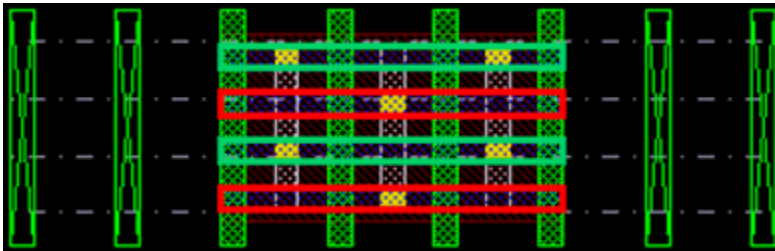
- *d_shapeId*
The database ID of the shape.
- *t_color*
Name of the color mask ("mask1Color", "mask2Color", or "mask3Color", "mask4Color", "mask5Color", "mask6Color", "mask7Color", or "mask8Color").
- *b_state*
A Boolean value that determines whether the color of the shape is locked.

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.

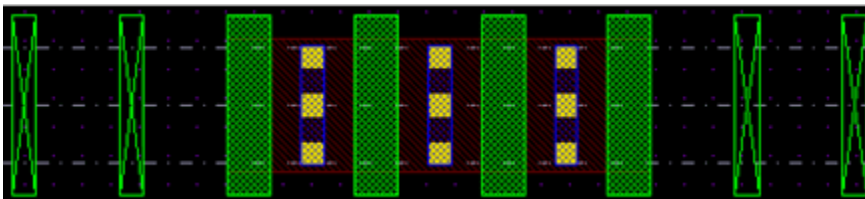
Example

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.



```
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)
)
```

In the figure below, none of the terminals is colored, so the query returns *nil*.

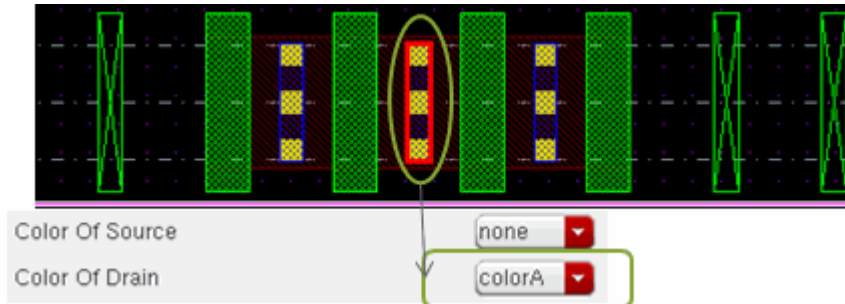


```
dbGetColoredOccShapes(css())
nil
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

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.



```
dbGetColoredOccShapes(css())  
nil
```

dbGetShapeEffectiveColor

```
dbGetShapeEffectiveColor(  
    l_occShape  
)  
=> l_colorAndColorState / nil
```

Description

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.

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](#).

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_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.

Example

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.

```
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 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.

```
shapes = dbShapeQuery( deGetCellView() list("M1" "drawing") list(0:0 0:0) )
foreach( occ shapes
  if( listp(occ) then
    path = reverse(cdr(reverse(occ))) ; gets all elements of the list but last
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
        shape = car(last(occ))
        println(dbGetShapeEffectiveColor(list(path shape)))
    else
        println(list(dbGetShapeColor(occ) dbIsShapeColorLocked(occ)))
    ) ; if
) ; foreach
```

dbSetOccShapeColor

```
dbSetOccShapeColor(  
    l_occShape  
    t_color  
)  
=> t / nil
```

Description

Assigns the specified color to the given occurrence shape. You can set the color for occurrence shapes with types `Path`, `PathSeg`, `Polygon`, and `Rect`.

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.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

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.

Example

Creates a one-level occurrence shape `rect0`, then sets its color to `mask2Color`.

```
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 via, then sets the color of a randomly picked shape from the via master to `mask1Color`.

```
viaDef_M2_M1 = techFindViaDefByName(tf "M2_M1")
vialt_m2_m1 = dbCreateVia(cv viaDef_M2_M1 10:10 "R0")
path = list(list(vialt_m2_m1) nth(1 viaDef_M2_M1~>viaHeader~>master~>shapes))
dbSetOccShapeColor(path "mask1Color")
```

Sets the color of the instance of `rect1` inside the mosaic to `mask2Color`.

```
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(mosaic 1 1) rect1)
dbSetOccShapeColor(path "mask2Color")
dbGetShapeEffectiveColor(path)
```

Sets the color of the `rect` shapes in the instances given by `inst1` and `inst2` to `mask1Color`.

```
dbSetOccShapeColor((list(list(inst1 inst2) rect) "mask1Color")
```

dbSetOccShapeColorLocked

```
dbSetOccShapeColorLocked(  
    l_occShape  
    g_setLocked  
)  
=> t / nil
```

Description

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`.

Arguments

<i>l_occShape</i>	<pre>list(l_hierPath d_shapeId)</pre> where, ■ <i>l_hierPath</i> is <code>list((d_refId l_mosaicElem)...) </code> where, □ <i>d_refId</i> The database ID of the non-mosaic instance or via. □ <i>l_mosaicElem</i> is <code>list(d_mosaicId x_row x_col)</code> Represents a mosaic element in the hierarchy path. where, ○ <i>d_mosaicId</i> The database ID of the mosaic instance. ○ <i>x_row</i> Row in which <i>d_shapeId</i> occurs inside this mosaic instance. ○ <i>x_col</i> Column in which <i>d_shapeId</i> occurs inside this mosaic instance. ■ <i>d_shapeId</i> The database ID of the shape. Supported shapes: Path, PathSeg, Polygon, Rect
<i>g_setLocked</i>	<pre>t nil</pre> ■ <i>t</i> The occurrence shape will be locked. ■ <i>nil</i> The occurrence shape will be unlocked.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Value Returned

<code>t</code>	Color state of the specified occurrence shape is set.
<code>nil</code>	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 <code>Path</code>, <code>PathSeg</code>, <code>Polygon</code>, or <code>Rect</code>.

Examples

Creates a one-level occurrence shape `rect0`, then sets its color to `mask1Color` and locks the color.

```
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 via, then sets the color of a randomly picked shape from the via master to `mask2Color` 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.

dbGetHierColorOverride

```
dbGetHierColorOverride(  
    l_occShape  
)  
=> l_occShapeHCL / nil
```

Description

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.

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*).

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

`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

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.

```
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))
```

Removes the color and hierarchical color lock on all the `metal2:drawing` shapes with a hierarchical color lock in the `top` cellview.

```
foreach( l dbColorShapeQuery2(top list("metal2" "drawing") top~>bBox )
  occShape = car(l)
  locked   = caddr(l)
  isHCL    = caddr(l)
  when(locked && isHCL
    hcl = dbGetHierColorOverride( occShape )
    dbSetOccShapeColorLocked( hcl nil )
    dbSetOccShapeColor( hcl "grayColor" )
  )
)
```

dbGetShapeColorLockType

```
dbGetShapeColorLockType(  
    d_shapeId  
)  
=> t_lockType / nil
```

Description

Returns the color lock type assigned to the specified shape. If the shape is not locked, `nil` is returned.

Arguments

<code>d_shapeId</code>	The database ID of a shape.

Value Returned

<code>t_lockType</code>	Color lock type assigned to the specified shape. Valid values: ■ <code>user</code> Coloring of the shape is locked and the lock type "system" is not set. ■ <code>system</code> Coloring of the shape is locked and the lock type "system" has been set using the <u>dbSetShapeColorLockType</u> SKILL function or some other Cadence application. By default, "user" lock type is assigned to locked shapes.
<code>nil</code>	Shape is not locked.

Example

Returns database ID of the currently selected shape.

```
shape = css()
```

Unlocks the shape color.

```
dbSetShapeColorLocked(shape nil)
```

Changes shape color.

```
dbSetShapeColor(shape "mask1Color")
```

Shape color is unlocked.

```
dbGetShapeColorLockType(shape)  
=> nil
```

Locks the shape color.

```
dbSetShapeColorLocked(shape t)
```

Color lock type as user.

```
dbGetShapeColorLockType(shape)  
=> "user"
```

Changes color lock type to system.

```
dbSetShapeColorLockType(shape "system")
```

Color lock type is system.

```
dbGetShapeColorLockType(shape)  
=> "system"
```

Unlocks color of the locked shape. Shows that `dbSetShapeColorLockType` can be used to unlock color of the locked shape.

```
dbSetShapeColorLockType(shape nil)
```

Shape color is locked.

```
dbGetShapeColorLockType(shape)  
=> nil
```

Confirms that shape color is not locked.

```
dbIsShapeColorLocked(shape)  
=> nil
```

Sets color lock type as `system` directly on the unlocked shape.

```
dbSetShapeColorLockType(shape "system")
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Color lock type is system.

```
dbGetShapeColorLockType(shape)  
=> "system"
```

Confirms that shape color is locked.

```
dbIsShapeColorLocked(shape)  
=> t
```

dbSetShapeColorLockType

```
dbSetShapeColorLockType(  
    d_shapeId  
    g_lockType  
)  
=> t / nil
```

Description

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.

Example

Returns database ID of the currently selected shape.

```
shape = css()
```

Unlocks the shape color.

```
dbSetShapeColorLocked(shape nil)
```

Changes shape color.

```
dbSetShapeColor(shape "mask1Color")
```

Shape color is unlocked.

```
dbGetShapeColorLockType(shape)  
=> nil
```

Locks shape color.

```
dbSetShapeColorLocked(shape t)
```

Color lock type is user.

```
dbGetShapeColorLockType(shape)  
=> "user"
```

Changes color lock type to system.

```
dbSetShapeColorLockType(shape "system")
```

Unlocks color of the locked shape. Shows that `dbSetShapeColorLockType` can be used to unlock color of the locked shape.

```
dbGetShapeColorLockType(shape)  
=> "system"  
dbSetShapeColorLockType(shape nil)
```

Shape color is unlocked.

```
dbGetShapeColorLockType(shape)  
=> nil
```

Confirms that shape color is unlocked.

```
dbIsShapeColorLocked(shape)  
=> nil
```

Color lock type set directly on unlocked shape using `dbSetShapeColorLockType`.

```
dbSetShapeColorLockType(shape "system")
```

Color lock type is system.

```
dbGetShapeColorLockType(shape)
```


Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
=> "system"
```

Confirms that shape color is locked.

```
dbIsShapeColorLocked(shape)
```

```
=> t
```

dbGetViaLayerColorLockType

```
dbGetViaLayerColorLockType(  
    d_viaId  
    t_viaLayerType  
)  
=> t_lockType / nil
```

Description

Returns the color lock type assigned to the specified via layer.

Arguments

<i>d_viaId</i>	The database ID of a via layer.
<i>t_viaLayerType</i>	Specifies the via layer type. Valid values: <code>layer1</code> , <code>layer2</code> , <code>cutLayer</code>

Value Returned

<i>t_lockType</i>	Color lock type assigned to the specified via layer. Valid values: ■ <code>user</code> Coloring of the via layer is locked and the lock type "system" is not set. ■ <code>system</code> 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.
<code>nil</code>	Via layer is not locked.

Example

Returns database ID of the currently selected via.

```
via = css()
```

Sets coloring of via layer1 to unlocked shift1.

```
dbSetViaColorInfo(via
                  list("shift1" nil)
                  nil
                  nil
                  )
```

Locks color of layer1.

```
dbGetViaLayerLockType(via "layer1")
=> nil
dbSetViaColorStateLayerLocked(via "layer1" t)
```

Lock type for layer1 is user.

```
dbGetViaLayerLockType(via "layer1")
=> "user"
```

Changes color lock type to system.

```
dbSetViaLayerColorLockType(via "layer1" "system")
```

Lock type for layer1 is system.

```
dbGetViaLayerLockType(via "layer1")
=> "system"
```

Via layer1 is unlocked

```
dbSetViaLayerColorLockType(via "layer1" nil)
```

Lock via layer1 using dbSetViaLayerColorLockType.

```
dbGetViaLayerLockType(via "layer1")
=> nil
```

Confirms that vial layer1 is unlocked.

```
dbIsViaColorStateLayerLocked(via "layer1")
=> nil
```

Sets color lock type system directly on unlocked via layer1.

```
dbSetViaColorInfo(via
                  list("noShift" "system")
                  nil
                  )
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
nil)
```

Lock type for layer1 is system.

```
dbGetViaLayerLockType(via "layer1")  
=> "system"
```

Confirms that via layer1 is locked.

```
dbIsViaColorStateLayerLocked(via "layer1")  
=> t
```

dbSetViaLayerColorLockType

```
dbSetViaLayerColorLockType(  
    d_viaId  
    t_viaLayer  
    g_lockType  
)  
=> t / nil
```

Description

Sets the color lock type for the specified via layer.

Arguments

<i>d_viaId</i>	The database ID of a via layer.
<i>t_viaLayerType</i>	Specifies the via layer type. Valid values: <code>layer1</code> , <code>layer2</code> , <code>cutLayer</code>
<i>g_lockType</i>	Color lock type to assign to the specified via layer. Valid values: ■ <code>"user"</code> Sets coloring of the via layer to locked, with lock type <code>"user"</code>. All via layers locked in the older versions of Virtuoso have these lock type. ■ <code>"system"</code> Sets coloring of the via layer to locked, with lock type <code>"system"</code>. This lock type indicates that the via layer was locked by an automated coloring tool. ■ <code>nil</code> Unlocks coloring of the via layer and clears information about its lock type. Default value for existing locked via layers is <code>"user"</code>

Value Returned

<code>t</code>	Specified lock type is successfully set.
<code>nil</code>	Lock type is not set.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Example

Returns database ID of the currently selected via.

```
via = css()
```

Sets coloring of via layer1 to unlocked shift1.

```
dbSetViaColorInfo(via
                  list("shift1" nil)
                  nil
                  nil
                  )
```

Locks color of layer1.

```
dbGetViaLayerLockType(via "layer1")
=> nil
dbSetViaColorStateLayerLocked(via "layer1" t)
```

Lock type for layer1 is user.

```
dbGetViaLayerLockType(via "layer1")
=> "user"
```

Changes color lock type to system.

```
dbSetViaLayerColorLockType(via "layer1" "system")
```

Lock type for layer1 is system.

```
dbGetViaLayerLockType(via "layer1")
=> "system"
```

Lock via layer1 using dbSetViaLayerColorLockType.

```
dbSetViaLayerColorLockType(via "layer1" nil)
```

Via layer1 is unlocked.

```
dbGetViaLayerLockType(via "layer1")
=> nil
```

Confirms that vial layer1 is unlocked.

```
dbIsViaColorStateLayerLocked(via "layer1")
=> nil
```

Sets color lock type system directly on unlocked via layer1.

```
dbSetViaColorInfo(via
                  list("noShift" "system")
                  nil
                  )
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
nil)
```

Lock type for layer1 is system.

```
dbGetViaLayerLockType(via "layer1")  
=> "system"
```

Confirms that via layer1 is locked.

```
dbIsViaColorStateLayerLocked(via "layer1")  
=> t
```

dbCellViewHasUnknownColorLockChanges

```
dbCellViewHasUnknownColorLockChanges (
    d_cellViewId
    [ g_shapes g_vias g_HCLs ]
)
=> t / nil
```

Description

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

<i>d_cellViewId</i>	Specifies the cellview ID.
<i>g_shapes</i>	A Boolean specifying whether the function should check color lock type of the shapes in the specified cellview. The default is <i>t</i> .
<i>g_vias</i>	A Boolean specifying whether the function should check color lock type of the vias in the cellview. The default is <i>t</i> .
<i>g_HCLs</i>	A Boolean specifying whether the function should check color lock type of the HCLs in the cellview. The default is <i>t</i> .

Value Returned

<i>t</i>	Specified cellview has objects with unknown edits.
<i>nil</i>	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))
```


Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
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
```

dbGetShapeEffectiveColorLockType

```
dbGetShapeEffectiveColorLockType(  
    l_occShape  
)  
=> t_colorLockType / nil
```

Description

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Value Returned

<i>t_colorLockType</i>	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.
<i>nil</i>	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
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
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     = caddr(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)))
```

dbCellViewResetAllColorLockTypes

```
dbCellViewResetAllColorLockTypes (
    d_cellViewId
    t_value
    [g_shapes g_vias g_HCLs]
)
=> t / nil
```

Description

Changes color lock type for all color-locked objects of the specified object types to the given value.

Arguments

<i>d_cellViewId</i>	Specifies the cellview ID.
<i>t_value</i>	Color lock type to reset the specified objects. Valid values: "user", "system"
<i>g_shapes</i>	A Boolean specifying whether the function should reset color lock type of the shapes in the specified cellview. The default is <i>t</i> .
<i>g_vias</i>	A Boolean specifying whether the function should reset color lock type of the vias in the cellview. The default is <i>t</i> .
<i>g_HCLs</i>	A Boolean specifying whether the function should reset color lock type of the HCLs in the cellview. The default is <i>t</i> .

Value Returned

<i>t</i>	Color lock type is reset to the specified value for all color-locked objects of the specified object types.
<i>nil</i>	An error occurs.

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
```

dbSetOccShapeColorLockType

```
dbSetOccShapeColorLockType(  
    l_occShape  
    g_colorLockType  
)  
=> t / nil
```

Description

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.

You can use the [dbGetShapeEffectiveColorLockType](#) function to check the color lock type of a given occShape.

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.

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

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")
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

```
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     = caddr(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)))
```

Virtuoso Studio Design Environment SKILL Reference

Multi-Patterning Technology Database Access Functions

Width Spacing Pattern Database Access Functions

This chapter includes the following topics: This topic provides a list of Cadence® SKILL functions associated with interactive and associated routing.

- [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](#)

Width Spacing Pattern Functions

This topic provides a list of Cadence® SKILL functions associated with width spacing pattern database access functions:

- [dbCreateWidthSpacingPattern](#)
- [dbCreateWidthSpacingPatternWithColor](#)
- [dbFindWidthSpacingPattern](#)
- [dbGetWidthSpacingPatternAllowedRepeatMode](#)
- [dbGetWidthSpacingPatternDefaultRepeatMode](#)
- [dbGetWidthSpacingPatterns](#)
- [dbSetWidthSpacingPatternRepeatMode](#)

dbCreateWidthSpacingPattern

```
dbCreateWidthSpacingPattern(  
    d_cellView  
    t_name  
    l_patternSpecs  
    [ n_offset ]  
    [ g_repeatOffset ]  
    [ g_shiftColor ]  
    [ t_allowedRepeatMode ]  
    [ t_defaultRepeatMode ]  
)  
=> d_widthSpacingPattern / nil
```

Description

Creates a width spacing pattern in the specified cellview.

Arguments

<i>d_cellView</i>	ID of the cellview for which the width spacing pattern must be created.
<i>t_name</i>	Name of the width spacing pattern.
<i>l_patternSpecs</i>	<pre>list(l_patternSpec)</pre> List of track group specifications. where, ■ <i>l_patternSpec</i>: <pre>list(l_specs [d_repeat [l_wireTypes [l_colorNames] [l_displayPacketNames]])</pre> A track group specification. □ <i>l_specs</i>: <pre>list(l_spec)</pre> List of individual track specifications for the specified track group. ○ <i>l_spec</i>: <pre>list(g_width g_space [t_wireType [t_colorName] [t_displayPacketName]])</pre> A track specification. <i>g_width</i> Width of shapes on the specified track.

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.

- *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.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

<i>n_offset</i>	Distance of the first track from the period track.
<i>g_repeatOffset</i>	Determines whether the offset is applied when a pattern repeats. When set to <i>t</i> the offset is repeated every time and when set to <i>nil</i> the offset is applied only once.
<i>g_shiftColor</i>	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 <i>nil</i> . The tracks cannot be colored automatically.
<i>t_allowedRepeatMode</i>	Name of allowed repeat mode. Valid values: <i>any</i> , <i>none</i> , <i>steppedOnly</i> , and <i>flippedOnly</i> . The default value is "any".
<i>t_defaultRepeatMode</i>	Name of default repeat mode. Valid values are Valid values: <i>noRepeat</i> , <i>stepped</i> , <i>flippedStartsWithOdd</i> , and <i>flippedStartsWithEven</i> . The default value is "noRepeat".

Value Returned

<i>d_widthSpacingPattern</i>	The ID of the created width spacing pattern.
<i>nil</i>	Returned in case of failure.

Examples

Example 1

A width spacing pattern *wsp1* is created in the cellview *cv* with two track specifications.

```
dbCreateWidthSpacingPattern(cv "wsp1" '(((0.04 0.08 "mywire1" "mask1Color") (0.09 0.12)))) 0.05 )
```

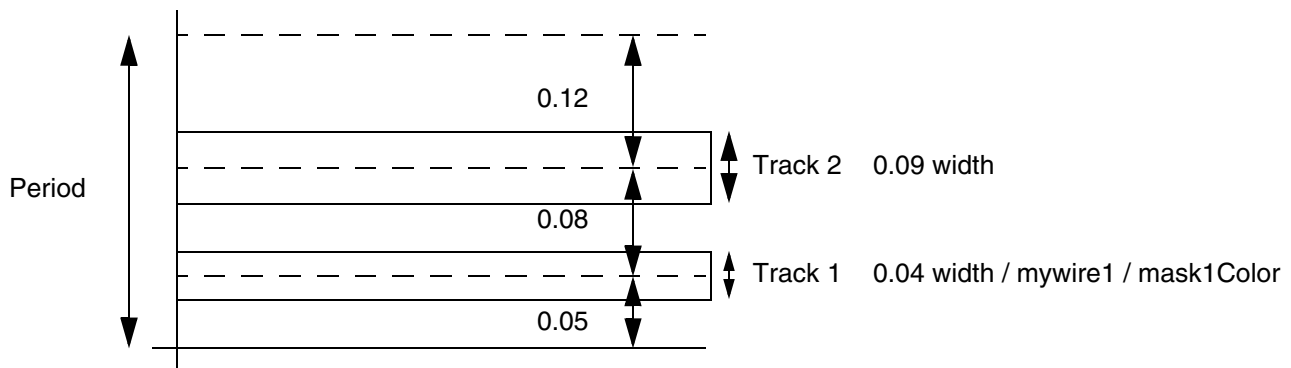
■ For the first track,

- ❑ 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.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

- ❑ `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.

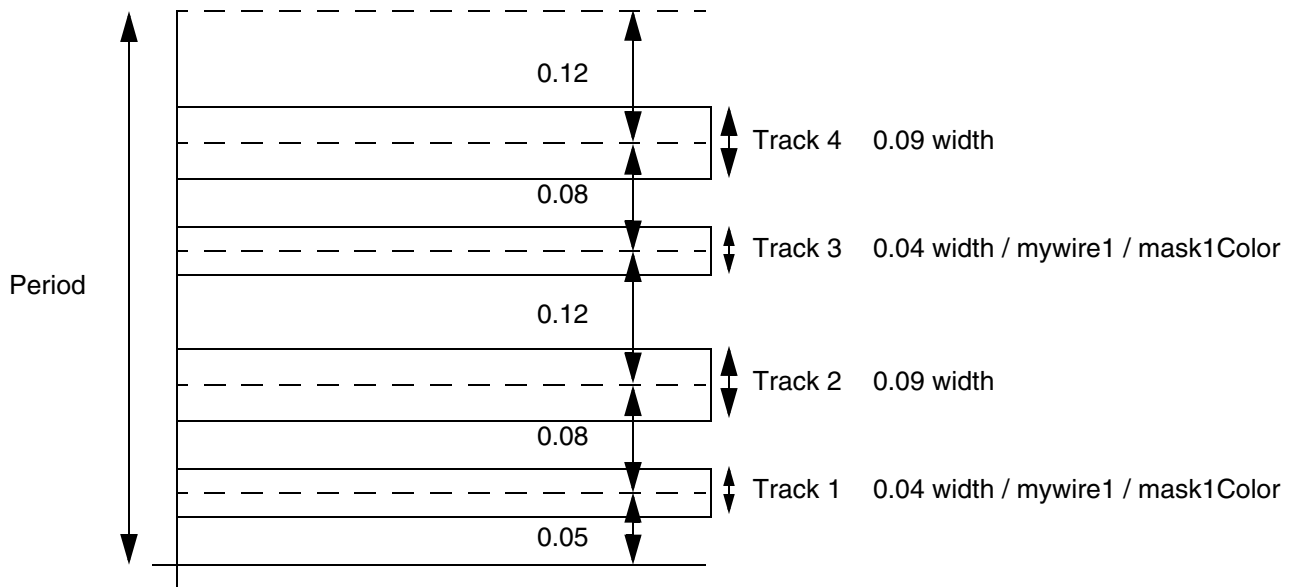


Example 2

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.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions



```
dbCreateWidthSpacingPattern(cv "wsp2"
'(((0.04 0.08 "mywire1" "mask1Color") (0.09 0.12)) 2))
0.05
)
```

Example 3

In this example, a width spacing pattern `wsp1` is created in the cellview `cv` with two track groups, defining a total of six tracks.

```
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"
```

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

) "stepped"

- For the first track group:
 - ❑ The width of the track is 0.04.
 - ❑ 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.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

- *allowedRepeatMode* is *steppedOnly*, which implies that the pattern can only be stepped, but not flipped. In conformance, *defaultRepeatMode* is *stepped*.

dbCreateWidthSpacingPatternWithColor

```
dbCreateWidthSpacingPatternWithColor(  
    d_cellView  
    t_name  
    l_patternSpecs  
    t_startingColor  
    [ n_offset ]  
    [ g_repeatOffset ]  
    [ t_allowedRepeatMode ]  
    [ t_defaultRepeatMode ]  
)  
=> d_widthSpacingPattern / nil
```

Description

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

<i>d_cellView</i>	The ID of the cellview for which the width spacing pattern must be created.
<i>t_name</i>	Name of the width spacing pattern.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

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).

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

- *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.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

<i>g_repeatOffset</i>	Determines whether the offset is applied when a pattern repeats. When set to <code>t</code> the offset is repeated every time and when set to <code>nil</code> the offset is applied only once.
<i>t_allowedRepeatMode</i>	Name of allowed repeat mode. Valid values are "any", "none", "steppedOnly" and "flippedOnly". The default value is "any".
<i>t_defaultRepeatMode</i>	Name of default repeat mode. Valid values are "noRepeat", "stepped", "flippedStartsWithOdd" and "flippedStartsWithEven". The default value is "noRepeat".

Value Returned

<i>d_widthSpacingPattern</i>	The ID of the created width spacing pattern.
<i>nil</i>	Returned in case of failure.

Examples

A width spacing pattern `wsp` is created in the cellview `cv` with two track specifications.

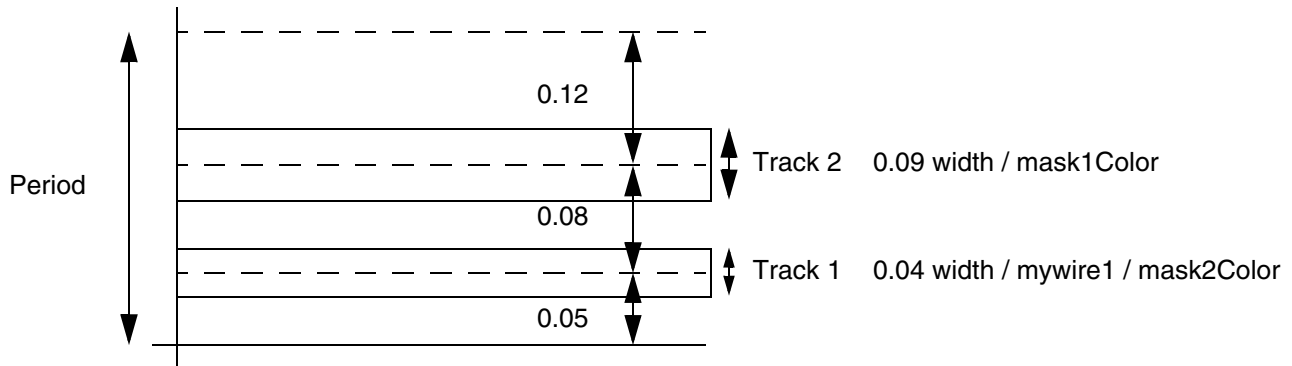
```
dbCreateWidthSpacingPatternWithColor(cv "wscp" '(((0.04 0.08 "mywire1"
"mask1Color") (0.09 0.012)))) "mask2Color" 0.05 t "steppedOnly" "stepped")
```

- 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`.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

- 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`.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

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

<i>d_cellView</i>	The ID of the cellview in which the search is performed.
<i>t_name</i>	Name of the width spacing pattern for which the search is performed.

Value Returned

<i>d_widthSpacingPattern</i>	The ID of the specified width spacing pattern, if found.
<i>nil</i>	Returned in case of failure.

Example

```
dbFindWidthSpacingPattern(cv "wspl")
```

dbGetWidthSpacingPatternAllowedRepeatMode

```
dbGetWidthSpacingPatternAllowedRepeatMode(  
    d_wspId  
)  
=> t_allowedRepeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the allowed repeat mode for the specified width spacing pattern.

Arguments

<i>d_wspId</i>	ID of a width spacing pattern.

Value Returned

<i>t_allowedRepeatMode</i>	Value of the allowed repeat mode.
<i>nil</i>	Returned in case of failure.

Example

Returns the allowed repeat mode for the width spacing pattern *wspId*.

```
dbGetWidthSpacingPatternAllowedRepeatMode(wspId)
```

dbGetWidthSpacingPatternDefaultRepeatMode

```
dbGetWidthSpacingPatternDefaultRepeatMode(  
    d_wspId  
)  
=> t_defaultRepeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the default repeat mode of the specified width spacing pattern.

Arguments

<i>d_wspId</i>	ID of a width spacing pattern.

Value Returned

<i>t_defaultRepeatMode</i>	Value of the default repeat mode.
<i>nil</i>	Returned in case of failure.

Example

Returns default repeat mode of the width spacing pattern *wspId*.

```
dbGetWidthSpacingPatternDefaultRepeatMode(wspId)
```

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

<i>d_cellView</i>	The ID of the cellview.

Value Returned

<i>l_widthSpacingPatterns</i>	The list of width spacing pattern IDs.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetWidthSpacingPatterns(cv)  
=> ("wsp1")
```

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

<i>d_wspId</i>	ID of a width spacing pattern.
<i>t_allowedRepeatMode</i>	Name of the allowed repeat mode. Valid values: any, none, steppedOnly, flippedOnly
<i>t_defaultRepeatMode</i>	Name of the default repeat mode. Valid values: noRepeat, stepped, flippedStartsWithOdd, flippedStartsWithEven

Value Returned

<i>t</i>	A valid combination of allowed and default repeat modes was set successfully on the specified width spacing pattern.
<i>nil</i>	Returned in case of failure.

Example

Sets the combination of allowed repeat mode as `flippedOnly` and default repeat mode as `flippedStartsWithOdd` for the width spacing pattern `wspId`.

```
dbSetWidthSpacingPatternRepeatMode(wspId "flippedOnly" "flippedStartsWithOdd ")
```


Width Spacing Pattern Group Functions

Supported functions include:

- [dbCreateWidthSpacingPatternGroup](#)
- [dbGetWidthSpacingPatternGroups](#)
- [dbFindWidthSpacingPatternGroup](#)

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

<i>d_cellView</i>	The ID of the cellview in which the width spacing pattern group is created.
<i>t_name</i>	Name of the width spacing pattern group.
<i>l_patternNames</i>	<code>list (t_patternName)</code> List of the width spacing patterns that must be included in the group.

Value Returned

<i>d_widthSpacingPatternGroup</i>	The ID of the created width spacing pattern group.
<code>nil</code>	Returned in case of failure.

Example

```
dbCreateWidthSpacingPatternGroup(cv "group1" '("wsp1" "wsp2"))
```

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

<i>d_cellView</i>	The ID of the cellview.

Value Returned

<i>l_widthSpacingPatternGroups</i>	List of width spacing pattern group IDs.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetWidthSpacingPatternGroups (cv)
=> ("wsp1")
```

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

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

<i>d_cellView</i>	The ID of the cellview in which the search is performed.
<i>t_name</i>	Name of the width spacing pattern group for which the search is performed.

Value Returned

<i>d_widthSpacingPatternGroup</i>	ID of the specified width spacing pattern group, if found.
<i>nil</i>	Returned in case of failure.

Example

```
dbFindWidthSpacingPatternGroup(cv "wsp_group1")
```

Width Spacing Snap Pattern Def Functions

Supported functions include:

- [dbCreateWidthSpacingSnapPatternDef](#)
- [dbFindWidthSpacingSnapPatternDefByName](#)
- [dbDeleteWidthSpacingSnapPatternDef](#)
- [dbGetWidthSpacingSnapPatternDefsByLP](#)

For information on Active Wire Type functions related to widthSpacingSnapPatternDef object, see [Active Wire Types Functions](#).

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

<i>d_cellviewId</i>	ID of the cellview in which the widthSpacingSnapPatternDef object is created.
<i>t_name</i>	Name of the widthSpacingSnapPatternDef object.
<i>(tx_layer [tx_purpose])</i>	The layer-purpose pair.
<i>t_direction</i>	The pattern direction. Valid values are horizontal and vertical.
<i>g_period</i>	The spacing between the coarse-grain period tracks (heart-beat period).
<i>t_defaultActivePattern</i>	The pattern used in the areas in which no regions have been drawn.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

<i>l_snappingLayers</i>	<code>list(l_snappingLayer)</code> The list of snapping layer definitions, where: ■ <code>l_snappingLayer: list(t_layer [l_purposeList])</code> The snapping layer definition, where: □ <code>t_layer</code> The name of the snapping layer. ■ <code>l_purposeList: list(t_purpose)</code> The list of purpose names.
<i>g_offset</i>	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.
<i>l_patternNames</i>	<code>list(t_patternName)</code> The list of the allowed width spacing pattern names.
<i>l_patternGroupNames</i>	<code>list(t_patternGroupName)</code> The list of the allowed width spacing pattern group names.
<i>g_gridType</i>	The WSP grid to use for line-ends. The following values are valid: ■ <code>upperLower</code>: Applies to both ends of the wires. ■ <code>lower</code>: Applies to the lower line-ends of vertical wires and the left line-ends of horizontal wires. ■ <code>upper</code>: Applies to the upper line-ends of vertical wires and the right line-ends of horizontal wires. ■ <code>nil</code>: The default value, which is used when this argument is not specified. In this case, the line-end grid is not configured for the <code>widthSpacingSnapPatternDef</code> object.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

Value Returned

d_widthSpacingSnapPatternDefId

The ID of the widthSpacingSnapPatternDef object that was created.

nil

The widthSpacingSnapPatternDef object was not created.

Example

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`.

```
dbCreateWidthSpacingSnapPatternDef(  
  cellView  
  "wsspDef"  
  ("layer1" "purpose1")  
  "vertical"  
  0.2  
  "wsp3"  
  (("layer2" ("dummy1" "dummy2")) ("layer3" ("dummy3" "dummy4")))  
  0.1  
  ("wsp" "wsp2")  
  ("basePatterns" "wspg2"))
```


dbFindWidthSpacingSnapPatternDefByName

```
dbFindWidthSpacingSnapPatternDefByName (
    d_cellViewId
    t_name
)
=> d_widthSpacingSnapPatternDefId / nil
```

Description

(ICADVM20.1 Only) Searches for the given widthSpacingSnapPatternDef object in the specified cellview.

Arguments

<i>d_cellviewId</i>	The ID of the cellview in which the search is performed.
<i>t_name</i>	Name of the widthSpacingSnapPatternDef object for which the search is performed.

Value Returned

<i>d_widthSpacingSnapPatternDefId</i>	The ID of the widthSpacingSnapPatternDef object that was found.
<i>nil</i>	The widthSpacingSnapPatternDef object was not found.

Example

Searches for the widthSpacingSnapPatternDef object `wsspDef` in the cellview `cv`.

```
dbFindWidthSpacingSnapPatternDefByName (cv "wsspDef")
```

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

Deletes the widthSpacingSnapPatternDef object wsspDef.

```
dbDeleteWidthSpacingSnapPatternDef(wsspDef)
```

dbGetWidthSpacingSnapPatternDefsByLP

```
dbGetWidthSpacingSnapPatternDefsByLP(  
    d_cellViewId  
    l_LP  
)  
=> l_widthSpacingSnapPatternDefIDList / nil
```

Description

(ICADVM20.1 Only) Returns all the widthSpacingSnapPatternDefs on a layer-purpose pair.

Arguments

<i>d_cellViewId</i>	ID of a cellview.
<i>l_LP</i>	List of layer-purpose names. <code>list(t_layer t_purpose)</code>

Return value

<i>l_widthSpacingSnapPatternDefIDList</i>	List of widthSpacingSnapPatternDef IDs on the given layer-purpose pair.
<i>nil</i>	Returned in case of a failure.

Examples

Returns the database IDs for all the widthSpacingSnapPatternDefs defined in the specified cellview *cv* for layer *Metal2* and purpose *localWSP*.

```
wSSPDef = dbGetWidthSpacingSnapPatternDefsByLP(cv list("Metal2" "localWSP"))  
> (db:0x1872da1a db:0x1872da1c)
```

Returns all the widthSpacingSnapPatternDefs defined in the specified cellview *cv* for layer *Metal2* and purpose *localWSP*.

```
dbGetWidthSpacingSnapPatternDefsByLP(cv list("Metal2" "localWSP"))~>name  
> ("M2WSP" "M2WSP_double")
```

Related Snap Pattern Functions

Supported functions include:

- [dbCreateRelatedSnapPatterns](#)
- [dbGetRelatedSnapPatterns](#)
- [dbFindRelatedSnapPatterns](#)

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

<i>d_cellView</i>	The ID of the cellview in which the related snap patterns are to be created.
<i>t_name</i>	Name of the group of the related snap patterns.
<i>l_relatedSnapPatterns</i>	<pre>list(l_snapPatternDef)</pre> List of the snap pattern definitions to include in the group. where, ■ <i>l_snapPatternDef</i>: <pre>list(t_snapPatternDefName [l_widthSpacingPatternNames] [l_widthSpacingPatternGroupNames])</pre> A snap pattern definition. □ <i>t_snapPatternDefName</i>: Name of the snap pattern definition. □ <i>l_widthSpacingPatternNames</i>: <pre>list(t_widthSpacingPatternName)</pre> 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.

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

- *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 )
```

dbGetRelatedSnapPatterns

```
dbGetRelatedSnapPatterns(  
    d_cellView  
)  
=> l_relatedSnapPatterns / nil
```

Description

(ICADVM20.1 Only) Returns a list of related snap pattern IDs from the specified cellview.

Arguments

<i>d_cellView</i>	The ID of the cellview.

Value Returned

<i>l_relatedSnapPatterns</i>	List of related snap pattern IDs.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetRelatedSnapPatterns(cv)  
=> ("rsp1")
```

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

<i>d_cellView</i>	The ID of the cellview in which the search is performed.
<i>t_name</i>	Name of the group of related snap patterns for which a search is to be performed.

Value Returned

<i>d_relatedSnapPatterns</i>	The ID of the specified related snap patterns group, if found.
<i>nil</i>	Returned in case of failure.

Example

```
dbFindRelatedSnapPatterns(cv "rsp1")
```


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](#).

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

<i>d_shapeId</i>	The ID of the shape.

Value Returned

<i>t_patternName</i>	The name of the active pattern.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionActivePattern(rect)  
=> "wsp1"  
dbGetPatternRegionActivePattern(rect1)  
=> nil
```

dbSetPatternRegionActivePattern

```
dbSetPatternRegionActivePattern(  
    d_shapeId  
    t_patternName  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Sets the given active pattern for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.
<i>t_patternName</i>	The name of the active pattern.

Value Returned

<i>t</i>	The active pattern was set successfully for the specified shape.
<i>nil</i>	Returned in case of failure.

Example

```
dbSetPatternRegionActivePattern(rect "wsp1")  
=> t  
dbGetPatternRegionActivePattern(rect)  
=> "wsp1"
```

dbHasPatternRegionActivePattern

```
dbHasPatternRegionActivePattern(  
    d_shapeId  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Determines whether the specified shape has an associated active pattern.

Arguments

<i>d_shapeId</i>	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 "wspl")  
=> t  
dbHasPatternRegionActivePattern(rect)  
=> t
```

dbGetPatternRegionAllowedPatterns

```
dbGetPatternRegionAllowedPatterns(  
    d_shapeId  
)  
=> l_patternNameList / nil
```

Description

(ICADVM20.1 Only) Returns a list of allowed pattern names for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.

Value Returned

<i>l_patternNameList</i>	The list of allowed pattern names.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetPatternRegionAllowedPatterns(rect)  
=> ("wsp1" "wsp2")
```

dbSetPatternRegionAllowedPatterns

```
dbSetPatternRegionAllowedPatterns(  
    d_shapeId  
    l_patternNameList  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Sets the given allowed patterns for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.
<i>l_patternNameList</i>	The list of allowed patterns.

Value Returned

<i>t</i>	The allowed patterns were set successfully for the specified shape.
<i>nil</i>	Returned in case of failure.

Example

```
dbSetPatternRegionAllowedPatterns(rect '("wsp1" "wsp2"))  
=> t
```

dbGetPatternRegionAllowedPatternGroups

```
dbGetPatternRegionAllowedPatternGroups (
    d_shapeId
)
=> l_groupNameList / nil
```

Description

(ICADVM20.1 Only) Returns a list of allowed pattern groups for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.

Value Returned

<i>l_groupNameList</i>	The list of allowed pattern groups.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetPatternRegionAllowedPatternGroups (rect)
=> ("wspg1" "wspg2")
```

dbSetPatternRegionAllowedPatternGroups

```
dbSetPatternRegionAllowedPatternGroups (  
    d_shapeId  
    l_groupNameList  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Sets the given allowed pattern groups for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.
<i>l_groupNameList</i>	The list of the allowed pattern groups.

Value Returned

<i>t</i>	The allowed pattern groups were set successfully for the specified shape.
<i>nil</i>	Returned in case of failure.

Example

```
dbSetPatternRegionAllowedPatternGroups (rect ' ("wspg1" "wspg2"))  
=> t
```


dbGetPatternRegionCommittedPatterns

```
dbGetPatternRegionCommittedPatterns(  
    d_shapeId  
)  
=> l_patternNameList / nil
```

Description

(ICADVM20.1 Only) Returns a list of committed pattern names for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.

Value Returned

<i>l_patternNameList</i>	The list of committed pattern names.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetPatternRegionCommittedPatterns(rect)  
=> ("wsp1")
```

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

<i>d_shapeId</i>	The ID of the shape.
<i>t_patternName</i>	The name of the committed pattern.

Value Returned

<i>t</i>	The specified pattern was committed with color shift.
<i>nil</i>	Returned in case of failure.

Example

```
dbGetPatternRegionCommittedPatternShiftColor(rect "wsp1")  
=> nil
```

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

<i>d_shapeId</i>	The ID of the shape.

Value Returned

<i>t_createType</i>	The create type.
<code>nil</code>	Returned in case of failure.

Example

```
dbGetPatternRegionCreateType(rect)  
=> "userCreated"
```

dbGetPatternRegionShiftColor

```
dbGetPatternRegionShiftColor(  
    d_shapeId  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Determines whether shift color is enabled for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.

Value Returned

t	The shift color is enabled.
nil	Returned in case of failure.

Example

```
dbGetPatternRegionShiftColor(rect)  
=> nil
```

dbSetPatternRegionShiftColor

```
dbSetPatternRegionShiftColor(  
    d_shapeId  
    g_shiftColor  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Determines whether shift color is enabled for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.
<i>g_shiftColor</i>	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
```

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

dbHasPatternRegionInfo

```
dbHasPatternRegionInfo(  
    d_shapeId  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Determines whether the specified shape contains pattern region information.

Arguments

<i>d_shapeId</i>	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
```

dbGetPatternRegionName

```
dbGetPatternRegionName(  
    d_shapeId  
)  
=> t_name / nil
```

Description

(ICADVM20.1 Only) Returns the name of the pattern region set for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.

Value Returned

<i>t_name</i>	The name of the pattern region.
nil	Returned in case of failure.

Example

The shape `rect` does not contain pattern region information.

```
rect = dbCreateRect(cv "Metall" "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
```

Pattern region information is assigned to the shape `rect`.

```
dbSetPatternRegionAllowedPatterns(rect "1X")  
> t  
dbHasPatternRegionInfo(rect)  
> t
```

The default name of a pattern region is an empty string.

```
dbGetPatternRegionName(rect)  
> ""
```

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

The pattern region "rect" has been set for the specified shape.

```
dbSetPatternRegionName(rect "rect")
```

```
> t
```

```
dbGetPatternRegionName(rect)
```

```
> "rect"
```


dbSetPatternRegionName

```
dbSetPatternRegionName(  
    d_shapeId  
    t_name  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Sets the specified pattern region for the specified shape.

Arguments

<i>d_shapeId</i>	The ID of the shape.
<i>t_name</i>	The name of the pattern region.

Value Returned

<i>t</i>	The specified pattern region is set.
<i>nil</i>	Returned in case of failure.

Example

The shape `rect` does not contain pattern region information.

```
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
```

Pattern region information is assigned to the shape `rect`.

```
dbSetPatternRegionAllowedPatterns(rect '("1X"))
```

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

```
> t
dbHasPatternRegionInfo(rect)
> t
```

The default name of a pattern region is an empty string.

```
dbGetPatternRegionName(rect)
> ""
```

The pattern region `rect` has been set for the specified shape.

```
dbSetPatternRegionName(rect "rect")
> t
dbGetPatternRegionName(rect)
> "rect"
```

dbGetPatternRegionWidthSpacingSnapPatternDefName

```
dbGetPatternRegionWidthSpacingSnapPatternDefName (
    d_dbShapeId
)
=> t_defName / nil
```

Description

(ICADVM20.1 Only) Returns the widthSpacingSnapPatternDef name from the specified pattern region.

Argument

<i>d_dbShapeId</i>	Database ID of a pattern region.

Return values

<i>t_defName</i>	Returned if a widthSpacingSnapPatternDef name was set successfully on the pattern region.
<i>nil</i>	Returned in case of failure.

Example

Returns "" because widthSpacingSnapPatternDef name is not set on the pattern region.

```
dbGetPatternRegionWidthSpacingSnapPatternDefName(region)
> ""
```

WidthSpacingSnapPatternDef name wSSPDef is set on the pattern region region.

```
dbSetPatternRegionWidthSpacingSnapPatternDefName(region "wSSPDef")
> t
```

Returns wSSPDef as the widthSpacingSnapPatternDef name for the pattern region.

```
dbGetPatternRegionWidthSpacingSnapPatternDefName(region)
> "wSSPDef"
```

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

<i>d_dbShapeId</i>	ID of a pattern region.
<i>t_defName</i>	Name of the widthSpacingSnapPatternDef to be stored for the pattern region.

Return value

<i>t</i>	Returned if the widthSpacingSnapPatternDef name has been set.
<i>nil</i>	Returned in case of failure.

Example

Sets the widthSpacingSnapPatternDef name `newWSSPDef` on the pattern region `region`.

```
dbSetPatternRegionWidthSpacingSnapPatternDefName (region
"newWSSPDef")
```

dbGetPatternRegionDefaultRepeatMode

```
dbGetPatternRegionDefaultRepeatMode(  
    d_patternRegion  
)  
=> t_repeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the default repeat mode for the specified pattern region.

Arguments

<i>d_patternRegion</i>	ID of the pattern region for which the repeat mode is set.

Value Returned

<i>t_repeatMode</i>	The default repeat mode for the specified pattern region.
<i>nil</i>	Returned in case of failure.

Example

Returns the default repeat mode for the pattern region `patternRegionId`.

```
dbGetPatternRegionDefaultRepeatMode(patternRegionId)
```

dbSetPatternRegionDefaultRepeatMode

```
dbSetPatternRegionDefaultRepeatMode(  
    d_patternRegion  
    t_repeatMode  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Sets default repeat mode on the specified pattern region.

Arguments

<i>d_patternRegion</i>	ID of the pattern region for which the repeat mode is to be set.
<i>t_repeatMode</i>	Name of the repeat mode to be set. Valid values: noRepeat, stepped, flippedStartsWithOdd, flippedStartsWithEven

Value Returned

<i>t</i>	Specified repeat mode is successfully set on the pattern region.
<i>nil</i>	Returned in case of failure.

Example

Sets the default repeat mode as `stepped` for the pattern region `patternRegionId`.

```
dbSetPatternRegionDefaultRepeatMode(patternRegionId "stepped")
```

dbGetPatternRegionRepeatMode

```
dbGetPatternRegionRepeatMode (
    d_patternRegion
    t_patternName
)
=> t_repeatMode / nil
```

Description

(ICADVM20.1 Only) Returns the repeat mode for the specified pattern region.

Arguments

<i>d_patternRegion</i>	ID of the pattern region for which the repeat mode is set.
<i>t_patternName</i>	Name of a width spacing pattern.

Return Value

<i>t_repeatMode</i>	The repeat mode for the specified pattern region.
<i>nil</i>	Returned in case of failure.

Example

Returns the repeat mode for the width spacing pattern `wsp` in the pattern region `patternRegionId`.

```
dbGetPatternRegionRepeatMode (patternRegionId "wsp")
```

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

<i>d_patternRegion</i>	ID of the pattern region on which the repeat mode is to be set.
<i>t_patternName</i>	Name of a width spacing pattern.
<i>t_repeatMode</i>	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

In this example, repeat mode `stepped` is successfully set on the pattern region `patternRegionId` for the width spacing pattern `wsp`.

```
dbSetPatternRegionRepeatMode (patternRegionId "wsp" "stepped")
```


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

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

<i>d_dbCellViewId</i>	ID of the cellview where the active wire types are to be returned.

Value Returned

<i>l_activeWireTypeNames</i>	List of active wire types.
<i>nil</i>	Returned in case of failure.

Example

Lists the active wire types in the cellview *cv*.

```
dbGetCellViewActiveWireTypes(cv)
```

dbSetCellViewActiveWireTypes

```
dbSetCellViewActiveWireTypes(  
    d_dbCellViewId  
    l_activeWireTypeNames  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Sets the active wire types on the specified cellview.

Arguments

<i>d_dbCellViewId</i>	ID of the cellview where the active wire types is set.
<i>l_activeWireTypeNames</i>	<pre>list(t_activeWireTypeNames)</pre> List of active wire types to be set on the cellview.

Value Returned

<i>t</i>	The active wire types was set successfully for the specified cellview.
<i>nil</i>	Returned in case of failure.

Example

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*.

```
dbSetCellViewActiveWireTypes(cv list("wire1" "wire2" "wire3"))
```

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

<i>d_dbCellViewId</i>	ID of the cellview where the active wire types are to be cleared.

Value Returned

<i>t</i>	The active wire specifications are successfully filtered.
<i>nil</i>	Returned in case of failure.

Example

Clears the active wire types in the cellview *cv*.

```
dbClearAllActiveWireTypes (cv)
```

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

<i>d_dbCellViewId</i>	ID of the cellview where the active wire types are to be returned.
<i>t_widthSpacingSnapPatternDefName</i>	Name of the widthSpacingSnapPatternDef in the cellview where the active wire types are to be returned.

Value Returned

<i>l_activeWireTypeNames</i>	List of active wire types.
<i>nil</i>	Returned in case of failure.

Example

Lists the active wire types on the cellview *cv* for the widthSpacingSnapPatternDef *wSSPD*.

```
dbGetCellViewActiveWireTypesForWidthSpacingSnapPatternDef (cv "wSSPD")
```

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

<i>d_dbCellViewId</i>	ID of the cellview where the active wire types are to be set.
<i>t_widthSpacingSnapPatternDefName</i>	Name of the widthSpacingSnapPatternDef in the cellview where the active wire types are to be set.
<i>l_activeWireTypeNames</i>	<pre>list(l_activeWireTypeName)</pre> List of active wire types.

Value Returned

<i>t</i>	The active wire types was set successfully on the specified cellview for the given widthSpacingSnapPatternDef.
<i>nil</i>	Returned in case of failure.

Example

Opens a cellview *cv* with a widthSpacingSnapPatternDef named *wSSPD* and the list of active wire types *wire1*, *wire2*, and *wire3* on it.

```
dbSetCellViewActiveWireTypesForWidthSpacingSnapPatternDef(cv "wSSPD" list("wire1"  
"wire2" "wire3"))
```

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

<i>d_PatternRegion</i>	ID of the pattern region where the active wire types are returned.

Value Returned

<i>l_activeWireTypeNames</i>	List of active wire types.
<i>nil</i>	Returned in case of failure.

Example

Lists the active wire types set on the pattern region *shape*.

```
dbGetPatternRegionActiveWireTypes(shape)
```

dbSetPatternRegionActiveWireTypes

```
dbSetPatternRegionActiveWireTypes(  
    d_PatternRegion  
    l_activeWireTypeNames  
)  
=> t / nil
```

Description

(ICADVM20.1 Only) Sets the active wire types on the specified pattern region.

Arguments

<i>d_PatternRegion</i>	ID of the pattern region where the active wire types are to be set.
<i>l_activeWireTypeNames</i>	<code>list(t_activeWireTypeNames)</code> List of active wire types to be set on the pattern region.

Value Returned

<i>t</i>	The active wire types are set on the specified pattern region.
<i>nil</i>	Returned in case of failure.

Example

Sets the active wire types `wire1`, `wire2` and `wire3` on the pattern region `shape`.

```
dbSetPatternRegionActiveWireTypes(shape list("wire1" "wire2" "wire3"))
```


dbSetPatternRegionNoActiveWireTypes

```
dbSetPatternRegionNoActiveWireTypes (
    d_PatternRegion
)
=> t / nil
```

Description

(ICADVM20.1 Only) Sets no active wire types on the specified pattern region.

Arguments

<i>d_PatternRegion</i>	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

Sets no active wire types on the pattern region `shape`.

```
dbSetPatternRegionNoActiveWireTypes(shape)
```

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

<i>d_PatternRegion</i>	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

Returns true if no active wire types is set on the specified pattern region shape.

```
dbHasPatternRegionNoActiveWireTypes(shape)
```

Global Grid Offset Functions

The supported functions include:

- [dbSetGlobalGridOffsetReferenceType](#)
- [dbGetGlobalGridOffsetReferenceType](#)
- [dbGetGlobalGridDefaultRepeatMode](#)
- [dbSetGlobalGridDefaultRepeatMode](#)
- [dbGetGlobalGridRepeatMode](#)
- [dbSetGlobalGridRepeatMode](#)

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 `napPatternGlobalGridAnchoring` environment variable. By default, `napPatternGlobalGridAnchoring` is set to `t`, which corresponds to `boundary`.

Arguments

<i>d_dbCellViewId</i>	ID of the cellview used to store the global grid offset reference type.
<i>t_referenceType</i>	The value of the global grid offset reference type. Valid values: <code>boundary</code> or <code>origin</code> .

Value Returned

<code>t</code>	The global grid offset reference type is set on the specified cellview.
<code>nil</code>	Returned in case of failure.

Example

Virtuoso Studio Design Environment SKILL Reference

Width Spacing Pattern Database Access Functions

Opens a cellview by using the `dbOpenCellViewByType` SKILL command that returns the cellview ID `cv`.

```
cv = dbOpenCellViewByType("design" "top" "layout")
```

Passes `cv` and `origin` to lock the global grid offset reference at the origin.

```
dbSetGlobalGridOffsetReferenceType(cv "origin")
```

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

<i>d_dbCellViewId</i>	ID of the cellview used to obtain the global grid offset reference type.

Value Returned

<i>t_referenceType</i>	The value of the global grid offset reference type. Valid values: boundary or origin.

Example

Opens a cellview by using the `dbOpenCellViewByType` SKILL command that returns the cellview ID `cv`.

```
cv = dbOpenCellViewByType("design" "top" "layout")
```

Passes the `cv` to `getGlobalGridOffsetReferenceType` to obtain the global grid reference type.

```
dbGetGlobalGridOffsetReferenceType(cv)
```

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

<i>d_cellViewId</i>	ID of a cellview.
<i>t_wsspDefName</i>	Name of a widthSpacingSnapPatternDef object.

Value Returned

<i>t_repeatMode</i>	Value of the default repeat mode.
<i>nil</i>	Returned in case of failure.

Example

Returns the default repeat mode for the *wsspDefName* global grid in the cellview *cellViewId*.

```
dbGetGlobalGridDefaultRepeatMode (cellViewId "wsspDefName")
```

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

<i>d_cellViewId</i>	ID of a cellview
<i>t_wsspDefName</i>	Name of a widthSpacingSnapPatternDef object.
<i>t_repeatMode</i>	Name of the repeat mode to be set. Valid values: noRepeat, stepped, flippedStartsWithOdd, flippedStartsWithEven

Value Returned

<i>t</i>	The default repeat mode is successfully set for the specified widthSpacingSnapPatternDef global grid.
<i>nil</i>	Returned in case of failure.

Example

The value `stepped` is set as the default repeat mode for the `wsspDefName` global grid in the cellview `cellViewId`.

```
dbSetGlobalGridDefaultRepeatMode(cellViewId "wsspDefName" "stepped")
```


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

<i>d_cellViewId</i>	ID of a cellview.
<i>t_wsspDefName</i>	Name of a widthSpacingSnapPatternDef object.
<i>t_patternName</i>	Name of a width spacing pattern.

Value Returned

<i>t_repeatMode</i>	Value of a valid repeat mode.
<i>nil</i>	Returned in case of failure.

Example

Returns the repeat mode for the *wsspDefName* global grid in the cellview *cellViewId* for the width spacing pattern *wspName*.

```
dbGetGlobalGridRepeatMode(cellViewId "wsspDefName" "wspName")
```

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

<i>d_cellViewId</i>	ID of a cellview.
<i>t_wsspDefName</i>	Name of the widthSpacingSnapPatternDef.
<i>t_patternName</i>	Name of the width spacing pattern.
<i>t_repeatMode</i>	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

The repeat mode `stepped` is set for the `wsspDefName` global grid in the cellview `cellViewId` for the width spacing pattern `wspName`.

Hierarchical Design Functions

This topic provides a list of Cadence® SKILL functions associated with the hierarchical database.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Hierarchy Database Functions

The Cadence® hierarchy database skill functions can be used to perform operations associated with the hierarchy database, such as create and update hierarchy configuration bags, bind hierarchical instances, and work with hierarchical cellviews.

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

dbDeleteHierCache

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

Description

Deletes the hierarchical cache file for the specified cellview, if the cache file exists.

Arguments

<i>d_cvId</i>	Database ID of the cellview that contains the hierarchical cache.

Value Returned

<i>t</i>	The cache file was successfully deleted.
<i>nil</i>	An error occurred or the cache file does not exist.

Example

Creates a hierarchical cache file and then deletes it.

```
cv = dbOpenCellViewByType("test" "hierTop" "layout" "maskLayout" "a")  
dbSetHierCacheUsage("saveAndLoad" cv)
```

```
dbSave(cv)  
dbHasValidHierCache(cv)  
=> t
```

```
dbDeleteHierCache(cv)  
=> t
```

```
dbHasValidHierCache(cv)  
=> nil
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Related Topics

[dbGetHierCacheUsage](#)

[dbHasValidHierCache](#)

[dbSetHierCacheUsage](#)

dbGetHierCacheUsage

```
dbGetHierCacheUsage (
    [ d_cvId ]
)
=> t_usageType / nil
```

Description

Returns the current hierarchical cache usage for a given design, if specified. If not specified, it returns the usage for the current session. The subset of operations included in both the design and session usages is enabled for the design.

Arguments

<i>d_cvId</i>	Database ID of the cellview that may be using a hierarchical cache. If specified, the cache usage for that cellview is returned. Otherwise, the session cache usage is returned.

Value Returned

<i>t_usageType</i>	The operations that the hierarchical cache may be used for. The <code>loadOnly</code> operation indicates that the hierarchical cache is used when loading the design hierarchy. The <code>saveOnly</code> operation indicates that the hierarchical cache is saved together with the cellview that contains it, which is typically the top design in the hierarchy. Valid returned values: <code>none</code> , <code>saveOnly</code> , <code>loadOnly</code> , <code>saveAndLoad</code>
<i>nil</i>	An error occurred.

Example

Returns the cache usage for the session:

```
dbSetHierCacheUsage ("saveOnly")
dbGetHierCacheUsage ()
=> "saveOnly"
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Returns the cache usage for the cellview:

```
cv = dbOpenCellViewByType("test" "hierTop" "layout" "maskLayout" "a")
dbSetHierCacheUsage("saveAndLoad" cv)
dbGetHierCacheUsage(cv)
=> "saveAndLoad"
```

Related Topics

[dbDeleteHierCache](#)

[dbHasValidHierCache](#)

[dbSetHierCacheUsage](#)

dbHasValidHierCache

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

Description

Checks whether the specified cellview has a valid hierarchical cache file.

Arguments

<i>d_cvId</i>	Database ID of the cellview to be checked for a hierarchical cache.

Value Returned

<i>t</i>	The cellview has a hierarchical cache file.
<i>nil</i>	The cellview does not have a hierarchical cache file.

Example

Checks whether the cellview has a hierarchical cache file before the cache file has been created.

```
cv = dbOpenCellViewByType("test" "hierTop" "layout" "maskLayout" "a")  
dbSetHierCacheUsage("saveAndLoad" cv)  
dbHasValidHierCache(cv)  
=> nil
```

Checks whether the cellview has a hierarchical cache file after the file has been created.

```
dbSave(cv)  
dbHasValidHierCache(cv)  
=> t
```

Related Topics

dbDeleteHierCache

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

dbGetHierCacheUsage

dbSetHierCacheUsage

dbSetHierCacheUsage

```
dbSetHierCacheUsage (
    t_usage
    [ d_cvId ]
)
=> t / nil
```

Description

Sets the intended hierarchical cache usage for a given design, if specified; otherwise, it sets the usage for the session. The subset of operations permitted by both the session and design cache usages is enabled for the design.

Arguments

<i>t_usage</i>	Specifies the operations for which the hierarchical cache can be used. Cache usage defines the intended policy. This policy is then implemented during respective future operations on the cellview such as <code>dbSave()</code> and <code>dbOpenHier()</code>. The <code>loadOnly</code> operation indicates that the hierarchical cache should be used when loading the design hierarchy. The <code>saveOnly</code> operation indicates that the hierarchical cache should be saved together with the cellview that contains it, which is typically the top design in the hierarchy. Valid values: <code>none</code>, <code>saveOnly</code>, <code>loadOnly</code>, <code>saveAndLoad</code>
<i>d_cvId</i>	Database ID of the cellview that contains the hierarchical cache. If specified, the cache usage is set for the cellview and not for the session.

Value Returned

<i>t</i>	The cache usage was set.
<i>nil</i>	The cache usage was not set.

Example

Sets the session usage to allow only the save operations using the hierarchical cache.

```
dbSetHierCacheUsage("saveOnly")
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

=> t

Sets the usage for the cellview to allow both save and load operations on the hierarchical cache. Because the session usage is still `saveOnly`, only the save operation is permitted for this cellview.

```
cv = dbOpenCellViewByType("test" "hierTop" "layout" "maskLayout" "a")
dbSetHierCacheUsage("saveAndLoad" cv)
=> t
```

Related Topics

[dbDeleteHierCache](#)

[dbGetHierCacheUsage](#)

[dbHasValidHierCache](#)

hdbAddConfigToBag

```
hdbAddConfigToBag (
    h_configBagId
    h_configId
)
=> t / nil
```

Description

Adds a configuration handle to a configuration bag.

Arguments

<i>h_configBagId</i>	The ID of the configuration bag.
<i>h_configId</i>	A configuration ID, something returned by <code>hdbOpen</code> .

Value Returned

<i>t</i>	The configuration bag exists and a handle has been added to it.
<i>nil</i>	The <i>h_configBagId</i> was invalid.

Example

```
hdbAddConfigToBag( configBag config )
```

An example for the use of this function can also be found at

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbCreateConfigBag](#)

[hdbCloseConfigsInBag](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<i>h_pathVector</i>	The path vector ID returned by <code>hdbCreatePathVector</code> .
<i>t_masterLibName</i>	The master library name for the instances.
<i>t_masterCellName</i>	The master cell name for the instances.
<i>t_masterViewName</i>	The master view name for the instances.
<i>t_instName</i>	The <i>instName</i> for the instance.
<i>g_fixedLib</i>	The indicator of whether <i>t_masterLibName</i> can be changed (t) or not (nil) by <code>hdbBind</code> .
<i>g_fixedCell</i>	The indicator of whether <i>t_masterCellName</i> can be changed (t) or not (nil) by <code>hdbBind</code> .
<i>g_fixedView</i>	The indicator of whether <i>t_masterViewName</i> can be changed (t) or not (nil) by <code>hdbBind</code> .
<i>g_wantSignature</i>	The indicator of whether a <i>h_bindSignature</i> object is returned (t) or not (nil).
<i>g_wantWhy</i>	The indicator of whether a <i>BindWhy</i> object is returned (t) or not (nil).

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Value Returned

<i>t_boundLib</i>	The library name of the bound instance.
<i>t_boundCell</i>	The cell name of the bound instance.
<i>t_boundView</i>	The view name of the bound instance.
<i>h_bindSignatureId</i>	The ID of an object that can be used to explain the uniqueness of a binding.
<i>h_bindWhyId</i>	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)
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbExplainBinding](#)

[hdbExplainSignature](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbCloseConfigsInBag

```
hdbCloseConfigsInBag(  
    h_configBagId  
)  
=> t / nil
```

Description

Closes all of the configurations held in a configuration bag.

Arguments

<i>h_ConfigBagId</i>	A configuration bag ID; something returned by <code>hdbCreateConfigBag</code> .

Value Returned

<i>t</i>	All of the configurations held in the bag were closed successfully.
<i>nil</i>	An error was encountered while trying to close all the configs in the bag.

Example

```
hdbCloseConfigsInBag( configBag )
```

An example for the use of this function can also be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[hdbCloseConfigsInBag](#)

[hdbAddConfigToBag](#)

[hdbCreateConfigBag](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Parent/Child Database Functions

hdbCreateConfigBag

```
hdbCreateConfigBag(  
    )  
    => h_configBagId / nil
```

Description

Creates a configuration bag to hold configurations.

Arguments

None

Value Returned

<i>h_configBagId</i>	The ID of a bag to put configurations in.
<i>nil</i>	A configuration bag was not created.

Example

```
configBag = hdbCreateConfigBag()
```

Reference

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbCloseConfigsInBag](#)

[hdbAddConfigToBag](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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()
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbGetPathStackStr](#)

[hdbGetPathVectorStackTop](#)

[hdbPushPathVect](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbConfigHasOccRules

```
hdbConfigHasOccRules(  
    h_configId  
)  
=> t / nil
```

Description

Checks whether a configuration contains any occurrence rules.

Arguments

<i>h_configId</i>	A configuration ID returned by the <code>hdbOpen</code> function.

Value Returned

<i>t</i>	The configuration contains one or more occurrence rules.
<i>nil</i>	The configuration does not contain any occurrence rules.

Examples

Checks whether the `configId` contains an occurrence rule.

```
when( hdbConfigHasOccRules(configId)  
=>t  
    printf("The specified configuration has at least one occurrence rule.\n")  
)
```

Related Topics

[hdbOpen](#)

[hdbSetObjBindRule](#)

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

<i>t_indentation</i>	The indentation string that will precede each line.
<i>t_masterLibName</i>	The master library name for the instance.
<i>t_masterCellName</i>	The master cell name for the instance.
<i>t_masterViewName</i>	The master view name for the instance.
<i>t_instName</i>	The <i>instName</i> for the instance.
<i>g_fixedLib</i>	The library is bound.
<i>g_fixedCell</i>	The cell is bound.
<i>g_fixedView</i>	The view is bound.
<i>t_boundLib</i>	The library name bound by <code>hdbBind</code> .
<i>t_boundCell</i>	The cell name bound by <code>hdbBind</code> .
<i>t_boundView</i>	The view name bound by <code>hdbBind</code> .
<i>h_bindSignatureId</i>	The <i>bindSignature</i> object returned by <code>hdbBind</code> .
<i>h_bindWhyId</i>	The <i>bindWhy</i> object returned by <code>hdbBind</code> .

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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)
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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)
```

An example for the use of this function can also be found at:

```
your_install_dir/tools/hdb/examples/hdbTraverse.il
```

Related Topics

[hdbBind](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbExplainStop

```
hdbExplainStop(  
    t_indentation  
    h_pathVectorId  
)  
=> t_string
```

Description

Explains why a stop was declared.

Arguments

<i>t_indentation</i>	The indentation string that precedes each line.
<i>h_pathVectorId</i>	A path vector ID; something returned by <code>hdbCreatePathVector</code> .

Value Returned

<i>t_string</i>	A string that explains why a stop was declared.

Example

```
whyStopStr = hdbExplainStop("    " pathVector)
```

An example for the use of this function can also be found at:

`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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 )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbGetPathVectorStackTop](#)

[hdbCreatePathVectorStack](#)

[hdbPushPathVect](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Parent/Child Database Functions

hdbGetPathStr

```
hdbGetPathStr(  
    h_pathVectorId  
)  
=> t_string
```

Description

Creates a string representation of a path vector.

Arguments

<i>h_pathVectorId</i>	A path vector ID; something returned by hdbCreatePathVector.

Value Returned

<i>t_string</i>	The path vector.

Example

```
path = hdbGetPathStr( pathVector )
```

An example for the use of this function can also be found at:

```
your_install_dir/tools/hdb/examples/hdbTraverse.il
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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 )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbGetPathStackStr](#)

[hdbCreatePathVectorStack](#)

[Attribute Retrieval and Modification Functions](#)

[hdbPushPathVect](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Parent/Child Database Functions

hdbIsAtStopPoint

```
hdbIsAtStopPoint(  
    h_pathVectorId  
)  
=> t / nil
```

Description

Checks if the point just pushed into with `hdbPushCell` is a stop point.

Arguments

<i>h_pathVectorId</i>	A path vector ID; something returned by <code>hdbCreatePathVector</code> .

Value Returned

<i>t</i>	The point just pushed into with <code>hdbPushCell</code> is a stop point.
<i>nil</i>	The point just pushed into with <code>hdbPushCell</code> is not a stop point.

Example

```
stop = hdbIsAtStopPoint( pathVector )
```

An example for the use of this function can also be found at:

`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbIsConfig

```
hdbIsConfig(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> t / nil
```

Description

Checks if a given library/cell/view is a configuration.

Arguments

<i>t_libName</i>	The library name.
<i>t_cellName</i>	The cell name.
<i>t_viewName</i>	The view name.

Value Returned

<i>t</i>	A given library/cell/view is a configuration.
<i>nil</i>	A given library/cell/view is not a configuration.

Example

```
isConfig = hdbIsConfig("lib1" "cell1" "view1")
```

An example for the use of this function can also be found at:

```
your_install_dir/tools/hdb/examples/hdbTraverse.il
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>h_configID</i>	A configuration ID, returned by hdbOpen .

Value Returned

<i>t</i>	The configuration ID is valid.
<i>nil</i>	The configuration ID is not valid.

Related Topics

[hdbOpen](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>h_configId</i>	A configuration ID, returned by <code>hdbOpen</code> .

Value Returned

<code>t</code>	The reference count is incremented.
<code>nil</code>	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
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>h_pathVectorId</i>	The top of the path vector stack.
nil	The stack is empty.

Example

```
pathVectorTop = hdbPopPathVect ( pathVectorStack )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbGetPathStackStr](#)

[hdbCreatePathVectorStack](#)

[hdbGetPathVectorStackTop](#)

[hdbPushPathVect](#)

[Hierarchy Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Occurrence Property Functions

Parent/Child Database Functions

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 )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[hdbGetPathStackStr](#)

[hdbCreatePathVectorStack](#)

[hdbGetPathVectorStackTop](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Hierarchy Database Functions

Occurrence Property Functions

Parent/Child Database Functions

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

<i>h_configId</i>	A configuration ID; something returned by <code>hdbOpen</code> .
<i>t_libName</i>	A library name.
<i>t_cellName</i>	A cell name.
<i>t_viewName</i>	A view name.

Value Returned

<i>t</i>	The configuration is saved to library/cell/view.
<i>nil</i>	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`.

Related Topics

[hdbSaveAs](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Hierarchy Database Functions

Occurrence Property Functions

Parent/Child Database Functions

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

hdbSaveAs

```
hdbSaveAs (
    h_configId
    t_libName
    t_cellName
    t_viewName
)
=> t / nil
```

Description

Saves a configuration to a library/cell/view.

Arguments

<i>h_configId</i>	A configuration ID; something returned by <code>hdbOpen</code> .
<i>t_libName</i>	A library name.
<i>t_cellName</i>	A cell name.
<i>t_viewName</i>	A view name.

Value Returned

<i>t</i>	The configuration is saved to library/cell/view.
<i>nil</i>	The configuration is not saved to library/cell/view.

Example

```
ok = hdbSaveAs( config "lib1" "cell1" "view1" )
```

An example for the use of this function can also be found at:

`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[hdbSaveAs](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Parent/Child Database Functions

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

hdbSave

```
hdbSave(  
    h_configId  
)  
=> t / nil
```

Description

Saves a configuration opened with `hdbOpen`.

Arguments

<code>h_configId</code>	A configuration ID returned from a call to <code>hdbOpen</code> .

Value Returned

<code>t</code>	The configuration is saved.
<code>nil</code>	The save operation failed.

Example

```
hdbSave( hdbConfig )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>h_configId</i>	The ID of a configuration.

Value Returned

<i>t</i>	Closes the configuration.
<i>nil</i>	The close operation failed.

Example

```
closeStatus = hdbClose( hdbConfig )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

hdbOpen

```
hdbOpen (
    t_libName
    t_cellName
    t_viewName
    t_mode
    [ t_nameSpace ]
)
=> h_configId / nil
```

Description

Opens a configuration for read or edit.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Arguments

<i>t_libName</i>	The library name for the configuration you want to open.
<i>t_cellName</i>	The cell name for the configuration you want to open.
<i>t_viewName</i>	The view name for the configuration you want to open.
<i>t_mode</i>	String value of <i>r</i> , <i>w</i> , or <i>a</i> . ■ "<i>r</i>" Opens a configuration for read by reading in an existing configuration file. ■ "<i>w</i>" 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. ■ "<i>a</i>" 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.
<i>t_nameSpace</i>	The name space you want to use for library/cell/view and instance names when using the hdb interface.

Value Returned

<i>h_configId</i>	The ID of the configuration opened.
<i>nil</i>	The open operation failed.

Example

```
hdbConfig = hdbOpen( "lib1" "cell1" "view1" "r" "CDBA" )
```

An example for the use of this function can also be found at:

```
your_install_dir/tools/hdb/examples/hdbTraverse.il
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbCreatePathVector

```
hdbCreatePathVector(  
    h_configId  
)  
=> h_pathVectorId / nil
```

Description

Creates a path vector to use while traversing (elaborating) a design hierarchy.

Arguments

<i>h_configId</i>	The ID of a configuration obtained with <code>hdbOpen</code> .

Value Returned

<i>h_pathVectorId</i>	The path vector for the configuration specified by a configuration ID.
<i>nil</i>	The operation failed.

Example

```
pathVector = hdbCreatePathVector( config )
```

An example for the use of this function can also be found at:

```
your_install_dir/tools/hdb/examples/hdbTraverse.il
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbDestroyPathVector

```
hdbDestroyPathVector(  
    h_pathVectorId  
)  
=> t / nil
```

Description

Destroys a path vector.

Arguments

<i>h_pathVectorId</i>	The path vector obtained with <code>hdbCreatePathVector</code> .

Value Returned

<i>t</i>	The path vector has been destroyed.
<i>nil</i>	The path vector has not been destroyed.

Example

```
status = hdbDestroyPathVector( pathVector )
```

An example for the use of this function can also be found at:

```
your_install_dir/tools/hdb/examples/hdbTraverse.il
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbGetCellName

```
hdbGetCellName(  
    h_configId  
)  
=> t_cellName / nil
```

Description

Gets the cell name of a configuration specified by *h_configId*.

Arguments

<i>h_configId</i>	The ID of a configuration obtained with <code>hdbOpen</code> .

Value Returned

<i>t_cellName</i>	The cell name of the configuration specified by <i>h_configId</i> .
<i>nil</i>	Did not get the cell name of the configuration specified by <i>h_configId</i> .

Example

```
configCellName = hdbGetCellName( config )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

hdbGetLibName

```
hdbGetLibName (
    h_configId
)
=> libName / nil
```

Description

Gets the library name of a configuration specified by *h_configId*.

Arguments

<i>h_configId</i>	The ID of a configuration obtained with <code>hdbOpen</code> .

Value Returned

<i>libName</i>	The library name of the configuration specified by <i>h_configId</i> .
<i>nil</i>	Did not get the library name of the configuration specified by <i>h_configId</i> .

Example

```
configLibName = hdbGetCellName( config )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbGetTopCellName

```
hdbGetTopCellName (
    h_configId
)
=> t_topCellName / nil
```

Description

Gets the name of the top cell of a configuration specified by *h_configId*.

Arguments

<i>h_configId</i>	The ID of a configuration obtained with <code>hdbOpen</code> .

Value Returned

<i>t_topCellName</i>	The top cell name of the configuration specified by <i>h_configId</i> .
<i>nil</i>	Did not get the top cell name of the configuration specified by <i>h_configId</i> .

Example

```
topCell = hdbGetTopCellName( config )
```

An example for the use of this function can also be found at:

your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

hdbGetTopLibName

```
hdbGetTopLibName (
    h_configId
)
=> t_topLibName / nil
```

Description

Gets the name of the top library of a configuration specified by *h_configId*.

Arguments

<i>h_configId</i>	The ID of a configuration obtained with <code>hdbOpen</code> .

Value Returned

<i>t_topLibName</i>	The top library name of the configuration specified by <i>h_configId</i> .
<i>nil</i>	Did not get the top library name of the configuration specified by <i>h_configId</i> .

Example

```
topLib = hdbGetTopLibName( config )
```

An example for the use of this function can be found at
your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbGetTopViewName

```
hdbGetTopViewName (
    h_configId
)
=> t_topViewName / nil
```

Description

Gets the name of the top view pointed to by *h_hdbConfig*.

Arguments

<i>h_configId</i>	The ID of a configuration obtained with <code>hdbOpen</code> .

Value Returned

<i>t_topViewName</i>	The top view name of the configuration specified by <i>h_configId</i> .
<i>nil</i>	Did not get the top view name of the configuration specified by <i>h_configId</i> .

Example

```
topView = hdbGetTopViewName( config )
```

An example for the use of this function can be found at
your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbGetViewName

```
hdbGetViewName(  
    h_configId  
)  
=> t_viewName / nil
```

Description

Gets the view name of a configuration specified by *h_configId*.

Arguments

<i>h_configId</i>	The ID of a configuration obtained with <code>hdbOpen</code> .

Value Returned

<i>t_viewName</i>	The view name of the configuration specified by <i>h_configId</i> .
<i>nil</i>	Did not get the view name of the configuration specified by <i>h_configId</i> .

Example

```
configViewName = hdbGetViewName( config )
```

An example for the use of this function can be found at
your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbPopCell

```
hdbPopCell(  
    h_pathVectorId  
)  
=> t / nil
```

Description

Pops an entry off a path vector.

Arguments

<i>h_pathVectorId</i>	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 )
```

An example for the use of this function can be found at
your_install_dir/tools/hdb/examples/hdbTraverse.il

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbPushCell

```
hdbPushCell(  
    h_pathVectorId  
    t_instanceName  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> t / nil
```

Description

Push a cell onto a path vector.

Arguments

<i>h_pathVectorId</i>	A path vector ID; something obtained from <code>hdbCreatePathVector</code> .
<i>t_instanceName</i>	An instance name.
<i>t_libName</i>	A library name.
<i>t_cellName</i>	A cell name.
<i>t_viewName</i>	A view name.

Value Returned

<i>t</i>	A cell has been pushed on to a path vector.
<i>nil</i>	A cell has not been pushed on to a path vector.

Example

```
status = hdbPushCell( pathVector "inst1" "lib1" "cell1" "view1" )
```

An example for the use of this function can be found at

`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[Hierarchy Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Occurrence Property Functions

Parent/Child Database Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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.
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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	<code>('hdbcLibListRule ("lib1" "lib2" "lib3" ...))</code>
View list	<code>('hdbcViewListRule ("view1" "view2" "view3" ...))</code>
Binding	<code>('hdbcBindingRule ("lib" "cell" "view"))</code> where <i>lib</i> , <i>cell</i> , or <i>view</i> can also be <code>nil</code> . For example: <pre>('hdbcBindingRule ("lib1" "cell1" "view1")) ('hdbcBindingRule (nil "cell1" "view1")) ('hdbcBindingRule (nil nil "view1"))</pre> Note: When setting a binding rule on an occurrence (path), all fields must be specified in the rule. For example, <pre>rule = list ('hdbcBindingRule list ("worklib" "subsub" "schematic2"))</pre>
Stop list	<code>('hdbcStopRule t)</code>

To delete a rule, specify `nil` as the second element of the list. For example: `('hdbcStopRule nil)`

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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 could be specified as:

```
spec= list(list("worklib" "top" "schematic" "IO") list("worklib" "sub"  
"schematic" "IO"))
```

and the corresponding rule could be defined as:

```
rule= list('hdbcBindingRule list("worklib" "subsub" "schematic2"))
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbSetDefaultLibListString

```
hdbSetDefaultLibListString(  
    h_configID  
    t_liblist  
)  
=> t / nil
```

Description

Sets the global library list in the configuration specified by the *h_configID*

Arguments

<i>h_configID</i>	Specifies a configuration ID returned by <code>hdbOpen</code>
<i>t_libList</i>	Specifies the library list to be set

Value Returned

<i>t</i>	If the library list has been set
<i>nil</i>	If the library list has not been set

Example

```
status = hdbSetDefaultLibListString( config "lib1 lib2 lib3" )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbSetDefaultViewListString

```
hdbSetDefaultViewListString(  
    h_configID  
    t_viewList  
)  
=> t / nil
```

Description

Sets the global view list in the configuration specified by *h_configID*

Arguments

<i>h_configID</i>	Specifies a configuration ID returned by <code>hdbOpen</code> .
<i>t_viewList</i>	Specifies the view list needs to be set.

Value Returned

<i>t</i>	If the view list has been set
<i>nil</i>	If the view list has not been set

Example

```
status = hdbSetDefaultViewListString( config "view1 view2 view3" )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

hdbSetDefaultStopListString

```
hdbSetDefaultStopListString(  
    h_configID  
    t_stopList  
)  
=> t / nil
```

Description

Sets the global stop view list in the configuration specified by *h_configID*

Arguments

<i>h_configID</i>	Specifies a configuration ID returned by <code>hdbOpen</code> .
<i>t_stopList</i>	Specifies the stop view list to be set.

Value Returned

<i>t</i>	If the stop view list has been set
<i>nil</i>	If the stop view list has not been set

Example

```
status = hdbSetDefaultStopListString( config "stop1 stop2 stop3" )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<code>h_configId</code>	Specifies a configuration ID returned by <code>hdbOpen</code>
<code>t_libName</code>	Specifies a library name
<code>t_cellName</code>	Specifies a cell name
<code>t_viewName</code>	Specifies a view name

Value Returned

<code>t</code>	If the top of the design hierarchy has been set
<code>nil</code>	If the top of the design hierarchy has not been set

Example

```
status = hdbSetTopCellViewName( config "lib1" "cell1" "view1" )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<i>h_pathVector</i>	The path vector ID returned by <code>hdbCreatePathVector</code> .
<i>t_masterLibName</i>	The master library name of the instance.
<i>t_masterCellName</i>	The master cell name of the instance.
<i>t_masterViewName</i>	The master view name of the instance.
<i>t_baseInstName</i>	The base part of the iterated instance name. For example, if the iterated instance name is <code>I1<0:3></code> , the base instance name is <code>I1</code> .
<i>x_bit</i>	The index number of the bit of the iterated instance being bound.
<i>x_leftRangeIndex</i>	The left range index of the iterated instance.
<i>x_rightRangeIndex</i>	The right range index of the iterated instance.
<i>g_fixedLib</i>	Indicates whether <i>t_masterLibName</i> can be changed (<code>t</code>) or not (<code>nil</code>).
<i>g_fixedCell</i>	Indicates whether <i>t_masterCellName</i> can be changed (<code>t</code>) or not (<code>nil</code>).

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

<i>g_fixedView</i>	Indicates whether <i>t_masterViewName</i> can be changed (t) or not (nil).
<i>g_wantSignature</i>	Indicates whether an <i>h_bindSignature</i> object is returned (t) or not (nil).
<i>g_wantWhy</i>	Indicates whether an <i>h_bindWhy</i> object is returned (t) or not (nil).

Value Returned

<i>t_boundLib</i>	The library name of the bound instance.
<i>t_boundCell</i>	The cell name of the bound instance.
<i>t_boundView</i>	The view name of the bound instance.
<i>h_bindSignatureId</i>	The ID of an object that indicates the uniqueness of a binding.
<i>h_bindWhyId</i>	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 )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>h_hdbPathVector</i>	The path vector ID returned by hdbCreatePathVector .
<i>t_baseInstName</i>	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.
<i>x_bit</i>	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 )
```

Related Topics

[hdbCreatePathVector](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Hierarchy Database Functions

Occurrence Property Functions

Parent/Child Database Functions

hdbHasIterInstBitRules

```
hdbHasIterInstBitRules(  
    h_configId  
)  
=> t /nil
```

Description

Determines whether a configuration contains rules that apply to iterated instance bits.

Argument

<i>h_configId</i>	A configuration ID, returned by hdbOpen .

Value Returned

<i>t</i>	The configuration contains rules that apply to iterated instance bits.
<i>nil</i>	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")  
)
```

Related Topics

[hdbOpen](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<i>t_instName</i>	The <i>instName</i> for the instance.
<i>t_nameSpace</i>	The name space you want to use for library/cell/view and instance names when using the Hierarchy Editor interface.

Value Returned

<i>t_baseInstName</i>	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.
<i>x_leftRangeIndex</i>	The left range index of the iterated instance.
<i>x_rightRangeIndex</i>	The right range index of the iterated instance.
<i>nil</i>	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)
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

```
    printf("%s is an iterated instance with range %x : %x\n"  
          baseInstName left right)  
)
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>h_pathVectorId</i>	The path vector ID as returned by hdbCreatePathVector .
<i>t_baseInstName</i>	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.
<i>x_leftRangeIndex</i>	The left range index of the iterated instance.
<i>x_rightRangeIndex</i>	The right range index of the iterated instance.

Value Returned

<i>t</i>	The iterated instance has bit-specific configuration rules.
<i>nil</i>	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 )
```

Related Topics

[hdbCreatePathVector](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Hierarchy Database Functions

Occurrence Property Functions

Parent/Child Database Functions

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

<i>h_pathVectorId</i>	A path vector ID as returned by hdbCreatePathVector .
<i>t_baseInstName</i>	The base part of the iterated instance name. For example, if the iterated instance name is <code>I1<0:3></code> , the base instance name is <code>I1</code> .
<i>x_bit</i>	The number of the bit of the iterated instance being bound.
<i>x_leftRangeIndex</i>	The left range index of the iterated instance.
<i>x_rightRangeIndex</i>	The right range index of the iterated instance.
<i>t_libName</i>	The library name of the iterated instance bit.
<i>t_cellName</i>	The cell name of the iterated instance bit.
<i>t_viewName</i>	The view name of the iterated instance bit.
<i>h_bindWhy</i>	The ID of an object that explains why the binding was made the way it was.

Value Returned

<i>t</i>	A cell was pushed on to the specified path vector.
<i>nil</i>	A cell was not pushed on to the specified path vector.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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" )
```

Related Topics

[hdbCreatePathVector](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Occurrence Property Functions

This section describes the SKILL functions that are used to work with occurrence properties of designs.

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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) )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<code>t_name</code>	The name of the property.
<code>g_value</code>	The value of the property.
<code>nonInheritable</code>	Specifies that the property value is non-inheritable. The default value of <code>nil</code> indicates that the property value is inheritable.
<code>noEval</code>	Specifies that the property value is not subject to evaluation. The default value of <code>nil</code> indicates that the property value is subject to evaluation.

Value Returned

<code>r_occpProp</code>	The property name structure, as returned by <code>occpCreateProp</code> .
<code>nil</code>	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
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<i>o_occpHandle</i>	The properties file handle returned from a call to <code>occpOpen</code> .
<i>h_pathVectorId</i>	The occurrence path for the property as returned from a call to <code>hdbCreatePathVector</code> .
<i>t_propName</i>	The name of the property.

Value Returned

<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .
<i>nil</i>	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) ))
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Arguments

<i>o_occpHandle</i>	The properties file handle returned from a call to <code>occpOpen</code> .
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property. The default value of <code>nil</code> indicates no specific cell name.
<i>t_viewName</i>	The view name for the property. The default value of <code>nil</code> indicates no specific view name.
<i>t_baseInstName</i>	The base instance name for the iterated instance property.
<i>x_bit</i>	The integer bit value of the base instance name for the iterated instance property.
<i>t_propName</i>	The name of the property.

Value Returned

<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .
<code>nil</code>	The property information does not exist or could not be retrieved.

Example

```
when( or(
  occpGetBitInstProp(MyOcpHandle "lib1" "cell1" "view1" "I42" 2 "variant")
  occpGetBitInstProp(MyOcpHandle nil "cell1" "view1" "I42" 2 "variant")
  occpGetBitInstProp(MyOcpHandle nil nil "view1" "I42" 2 "variant")
  occpGetBitInstProp(MyOcpHandle nil nil nil "I42" 2 "variant") )
  printf("Iterated instance[2] has variant property\n") )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

occpGetCellProp

```
occpGetCellProp(  
    o_occpHandle  
    t_libName | nil  
    t_cellName  
    t_propName  
)  
=> r_occpProp / nil
```

Description

Gets the property information for a cell.

Arguments

<i>o_occpHandle</i>	The properties file handle returned from a call to <code>occpOpen</code> .
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property.
<i>t_propName</i>	The name of the property.

Value Returned

<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .
<code>nil</code>	The property information does not exist or could not be retrieved.

Example

```
when( or(  
    occpGetCellProp(MyOcpHandle "lib1" "cell1" "variant")  
    occpGetCellProp(MyOcpHandle nil "cell1" "variant")  
    printf("Cell has variant property\n") ) )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Parent/Child Database Functions

occpGetDefaultProp

```
occpGetDefaultProp(  
    o_occpHandle  
    t_propName  
)  
=> r_occpProp / nil
```

Description

Gets default property information from the specified properties file.

Arguments

<i>o_occpHandle</i>	The properties file handle returned from a call to <code>occpOpen</code> .
<i>t_propName</i>	The name of the property.

Value Returned

<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .
<i>nil</i>	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))
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>o_occpHandle</i>	The properties file handle returned from a call to <code>occpOpen</code> .
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property. The default value of <code>nil</code> indicates no specific cell name.
<i>t_viewName</i>	The view name for the property. The default value of <code>nil</code> indicates no specific view name.
<i>t_instName</i>	The instance name for the property.
<i>t_propName</i>	The name of the property.

Value Returned

<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .
<code>nil</code>	The property information does not exist or could not be retrieved.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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") )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

occpGetPathProp

```
occpGetPathProp(  
    o_occpHandle  
    h_pathVectorId  
    t_propName  
)  
=> r_occpProp / nil
```

Description

Gets property information for the specified occurrence path.

Arguments

<i>o_occpHandle</i>	The properties file handle returned from a call to <code>occpOpen</code> .
<i>h_pathVectorId</i>	The occurrence path for the property as returned from a call to <code>hdbCreatePathVector</code> .
<i>t_propName</i>	The name of the property.

Value Returned

<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .
<i>nil</i>	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) ))
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Arguments

<i>t_libName</i>	The library name for the configuration.
<i>t_cellName</i>	The cell name for the configuration.
<i>t_viewName</i>	The view name for the configuration.
<i>t_mode</i>	Specifies one of these modes: ■ <i>r</i>: Opens an existing properties file for reading. ■ <i>w</i>: Creates a new properties file and opens it for editing. If a properties file already exists, it is opened with previous content discarded. ■ <i>a</i>: Opens an existing properties file for editing. If it does not exist, creates a new file and opens it for editing.
<i>t_nameSpace</i>	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.
<i>x_iterInstSupport</i>	Indicates if the application supports iterated instance properties. The default value of 0 indicates that the application does not support iterated instances.

Value Returned

<i>o_occpHandle</i>	The handle for the opened properties file.
<i>nil</i>	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"))
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Arguments

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property. The default value of <code>nil</code> indicates no specific cell name.
<i>t_viewName</i>	The view name for the property. The default value of <code>nil</code> indicates no specific view name.
<i>t_baseInstName</i>	The base instance name for the iterated instance property.
<i>x_bit</i>	The integer bit value of the base instance name for the iterated instance property.
<i>t_propName</i>	The name of the property.

Value Returned

<code>t</code>	The property was removed.
<code>nil</code>	The property does not exist or could not be removed.

Example

```
when( occpGetBitInstProp(MyOcpHandle "lib1" "cell1" "view1" "I42" 2 "variant")
      occpRmBitInstProp(MyOcpHandle nil "cell1" "view1" "I42" 2 "variant") )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

occpRmCellProp

```
occpRmCellProp(  
    o_occpHandle  
    t_libName | nil  
    t_cellName  
    t_propName  
)  
=> t / nil
```

Description

Removes a cell property.

Arguments

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property.
<i>t_propName</i>	The name of the property.

Value Returned

<i>t</i>	The cell property was removed.
<i>nil</i>	The cell property does not exist or could not be removed.

Example

```
when( occpGetCellProp(MyOccpHandle "lib1" "cell1" "variant")  
    occpRmCellProp(MyOccpHandle nil "cell1" "variant") )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

occpRmDefaultProp

```
occpRmDefaultProp(  
    o_occpHandle  
    t_propName  
)  
=> t / nil
```

Description

Removes a default property.

Arguments

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>t_propName</i>	The name of the property.

Value Returned

<i>t</i>	The property was removed.
<i>nil</i>	The property does not exist or could not be removed.

Example

```
occpRmDefaultProp(MyOccpHandle "variant")
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

occpRmPathProp

```
occpRmPathProp(  
    o_occpHandle  
    h_pathVectorId  
    t_propName  
)  
=> t / nil
```

Description

Removes an occurrence path property.

Arguments

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>h_pathVectorId</i>	The occurrence path for the property as returned from a call to <code>hdbCreatePathVector</code> .
<i>t_propName</i>	The name of the property.

Value Returned

<i>t</i>	The property was removed.
<i>nil</i>	The property does not exist or could not be removed.

Example

```
occpRmDefaultProp(MyOcpHandle "variant")
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property. The default value of <code>nil</code> indicates no specific cell name.
<i>t_viewName</i>	The view name for the property. The default value of <code>nil</code> indicates no specific view name.
<i>t_instName</i>	The instance name of the property.
<i>t_propName</i>	The name of the property.

Value Returned

<i>t</i>	The property was removed.
<i>nil</i>	The property does not exist or could not be removed.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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") ))
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<code>o_occpHandle</code>	The properties file handle returned from a call to <code>occpOpen</code> .

Value Returned

<code>t</code>	The properties file was saved.
<code>nil</code>	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) )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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.

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Arguments

<code>o_occpHandle</code>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<code>t_libName</code>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<code>t_cellName</code>	The cell name for the property. It must be specified if a library name is specified. The default value of <code>nil</code> indicates no specific cell name.
<code>t_viewName</code>	The view name for the property. It must be specified if a cell name is specified. The default value of <code>nil</code> indicates no specific view name.
<code>t_baseInstName</code>	The base instance name for the iterated instance property.
<code>x_bit</code>	The integer bit value of the base instance name for the iterated instance property.
<code>t_propName</code>	The property name structure as returned by <code>occpCreateProp</code> .

Value Returned

<code>t</code>	The property was set.
<code>nil</code>	The property could not be set.

Example

```
MyProp = occpGetBitInstProp(MyOcpHandle "lib1" "cell1" "view1" "I42" 2 "variant")
when( MyProp
    occpSetBitInstProp(MyOcpHandle "lib1" "cell1" "view1" "I42" 3 MyProp)
    occpSetBitInstProp(MyOcpHandle "lib1" "cell1" "view1" "I42" 4 MyProp) )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property.
<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .

Value Returned

<code>t</code>	The property was set.
<code>nil</code>	The property could not be set.

Example

```
unless( or( occpGetCellProp(MyOcppHandle "lib1" "cell1" "variant")  
    occpGetCellProp(MyOcppHandle nil "cell1" "variant") )  
    occpSetCellProp(MyOcppHandle "lib1" "cell1" occpCreateProp("variant"  
        "lowPower")) )
```

Related Topics

Hierarchy Database Functions

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Occurrence Property Functions

Parent/Child Database Functions

occpSetDefaultProp

```
occpSetDefaultProp(  
    o_occpHandle  
    r_occpProp  
)  
=> t / nil
```

Description

Sets the default property information. The new setting overwrites any existing default property setting.

Arguments

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .

Value Returned

<i>t</i>	The property was set.
<i>nil</i>	The property could not be set.

Example

```
occpSetDefaultProp(MyOcpHandle occpCreateProp("variant" "lowPower"))
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Arguments

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>t_libName</i>	The library name for the property. The default value of <code>nil</code> indicates no specific library name.
<i>t_cellName</i>	The cell name for the property. It must be specified if a library name is specified. The default value of <code>nil</code> indicates no specific cell name.
<i>t_viewName</i>	The view name for the property. It must be specified if a cell name is specified. The default value of <code>nil</code> indicates no specific view name.
<i>t_instName</i>	The instance name for the instance property.
<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .

Value Returned

<code>t</code>	The property was set.
<code>nil</code>	The property could not be set.

Example

```
unless( occpGetInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42" "variant")
  occpSetInstProp(MyOccpHandle "lib1" "cell1" "view1" "I42"
    occpCreateProp("variant" "lowPower")) )
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

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

<i>o_occpHandle</i>	The properties file handle returned by <code>occpOpen</code> when used with the <code>a</code> or <code>w</code> mode.
<i>h_pathVectorId</i>	The occurrence path for the property as returned from a call to <code>hdbCreatePathVector</code> .
<i>r_occpProp</i>	The property name structure, as returned by <code>occpCreateProp</code> .

Value Returned

<i>t</i>	The property was set.
<i>nil</i>	The property could not be set.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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) ))
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Parent/Child Database Functions

You can use the parent child database SKILL functions to perform operations such as opening a parent child database handle or closing it or getting the instance master cellview details.

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Example for all pcdb Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

pcdbClose

```
pcdbClose(  
    h_pcdbHandle  
)  
=> t / nil
```

Description

Closes a pcdb handle opened with `pcdbOpen`.

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 this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbOpen](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

[Example for all pcdb Functions](#)

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

<i>h_master</i>	A master returned from a call to <code>pcdbNextInstMaster</code> .

Value Returned

<i>h_instGen</i>	An instance generator.
<i>nil</i>	Did not get an instance generator.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbNextInst](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

[Example for all pcdb Functions](#)

pcdbGetInstMasterGen

```
pcdbGetInstMasterGen(  
    h_pcdbHandle  
)  
=> h_masterGen / nil
```

Description

Creates a generator that can be used to get the masters of a view.

Arguments

<i>h_pcdbHandle</i>	A pcdb handle returned from a call to <code>pcdbOpen</code> .

Value Returned

<i>b_masterGen</i>	A generator.
<i>nil</i>	Did not get a generator.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbOpen](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

[Example for all pcdb Functions](#)

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

<i>h_pcdbHandle</i>	A pcdb handle returned from a call to <code>pcdbOpen</code> .

Value Returned

<i>b_masterGen</i>	A generator.
<i>nil</i>	Did not create a generator.

Example

An example for the use of this function can be found at

`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbOpen](#)

[Example for all pcdb Functions](#)

pcdbInstMasterCell

```
pcdbInstMasterCell(  
    h_master  
)  
=> t_cellName / nil
```

Description

Gets the cell name for a master.

Arguments

<i>h_master</i>	A master returned from a call to <code>pcdbNextInstMaster</code> .

Value Returned

<i>t_cellName</i>	A string which is the master's cell name.
<i>nil</i>	Did not get a cell name for a master.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbInstMasterLib](#)

[pcdbInstMasterView](#)

[pcdbInstMasterLibFixed](#)

[pcdbInstMasterCellFixed](#)

[pcdbInstMasterViewFixed](#)

[Example for all pcdb Functions](#)

pcdbInstMasterCellFixed

```
pcdbInstMasterCellFixed(  
    h_master  
)  
=> t / nil
```

Description

Gets whether a master cell name is fixed.

Arguments

<i>h_master</i>	A master returned from a call to <code>pcdbNextInstMaster</code> .

Value Returned

t	The master cell is fixed.
nil	The master cell is not fixed.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbInstMasterLib](#)

[pcdbInstMasterCell](#)

[pcdbInstMasterView](#)

[pcdbInstMasterLibFixed](#)

[pcdbInstMasterViewFixed](#)

[Example for all pcdb Functions](#)

pcdbInstMasterLib

```
pcdbInstMasterLib(  
    h_master  
)  
=> t_libName / nil
```

Description

Gets the library name for a master.

Arguments

<i>h_master</i>	A master returned from a call to <code>pcdbNextInstMaster</code> .

Value Returned

<i>t_libName</i>	A string which is the master's library name.
<i>nil</i>	Did not get a library name for a master.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbInstMasterCell](#)

[pcdbInstMasterView](#)

[pcdbInstMasterLibFixed](#)

[pcdbInstMasterViewFixed](#)

[Example for all pcdb Functions](#)

pcdbInstMasterLibFixed

```
pcdbInstMasterLibFixed(  
    h_master  
)  
=> t / nil
```

Description

Gets whether a master's library name is fixed.

Arguments

<i>h_master</i>	A master returned from a call to <code>pcdbNextInstMaster</code> .

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 this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbInstMasterLib](#)

[pcdbInstMasterCell](#)

[pcdbInstMasterView](#)

[pcdbInstMasterCellFixed](#)

[pcdbInstMasterViewFixed](#)

[Example for all pcdb Functions](#)

pcdbInstMasterView

```
pcdbInstMasterView(  
    h_master  
)  
=> t_viewName / nil
```

Description

Gets the view name for a master.

Arguments

<i>h_master</i>	A master returned from a call to <code>pcdbNextInstMaster</code> .

Value Returned

<i>t_viewName</i>	A string which is the master's view name.
<i>nil</i>	Did not get a view name for a master.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbInstMasterLib](#)

[pcdbInstMasterCell](#)

[pcdbInstMasterLibFixed](#)

[pcdbInstMasterCellFixed](#)

[pcdbInstMasterViewFixed](#)

[Example for all pcdb Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

pcdbInstMasterViewFixed

```
pcdbInstMasterViewFixed(  
    h_master  
)  
=> t / nil
```

Description

Gets whether a master's view name is fixed.

Arguments

<i>h_master</i>	A master returned from a call to <code>pcdbNextInstMaster</code> .

Value Returned

<i>t</i>	The master's view name is fixed.
<i>nil</i>	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

An example for the use of this function can be found at

`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInstMaster](#)

[pcdbInstMasterLib](#)

[pcdbInstMasterCell](#)

[pcdbInstMasterView](#)

[pcdbInstMasterLibFixed](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

pcdbInstMasterCellFixed

Example for all pcdb Functions

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

pcdbInstName

```
pcdbInstName(  
    h_instance  
)  
=> t_instName / nil
```

Description

Gets the instance name for an instance.

Arguments

<i>h_instance</i>	An instance returned from a call to <code>pcdbNextInst</code> .

Value Returned

<i>t_instName</i>	A string which is the instance name.
<i>nil</i>	Did not get an instance name for an instance.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInst](#)

[pcdbInstWidth](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

[Example for all pcdb Functions](#)

pcdbInstWidth

```
pcdbInstWidth(  
    h_instance  
)  
=> x_width / nil
```

Description

Gets the instance width for an instance.

Arguments

<i>h_instance</i>	An instance returned from a call to <code>pcdbNextInst</code> .

Value Returned

<i>x_width</i>	The instance width.
<i>nil</i>	Did not get an instance width for an instance.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbNextInst](#)

[pcdbInstName](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

[Example for all pcdb Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

pcdbNextInst

```
pcdbNextInst(  
    h_instGen  
)  
=> h_instance / nil
```

Description

Gets the next instance from an instance generator.

Arguments

<i>h_instGen</i>	An instance generator returned from a call to <code>pcdbNextInst</code> .

Value Returned

<i>h_instance</i>	An instance.
<i>nil</i>	Did not get an instance from an instance generator.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbGetInstGen](#)

[pcdbInstName](#)

[pcdbInstWidth](#)

[Parent/Child Database Functions](#)

[Example for all pcdb Functions](#)

pcdbNextInstMaster

```
pcdbNextInstMaster(  
    h_masterGen  
)  
=> h_master / nil
```

Description

Gets the next master from a master generator.

Arguments

<i>h_instGen</i>	A master generator returned from a call to <code>pcdbNextInst</code> .

Value Returned

<i>h_master</i>	A master.
<i>nil</i>	Did not get a master from a master generator.

Example

An example for the use of this function can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbGetInstMasterGen](#)

[pcdbInstMasterLib](#)

[pcdbInstMasterCell](#)

[pcdbInstMasterView](#)

[pcdbInstMasterLibFixed](#)

[pcdbInstMasterCellFixed](#)

[pcdbInstMasterViewFixed](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Example for all pcdB Functions

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Arguments

<i>t_libName</i>	The library name for the configuration you want to open.
<i>t_libName</i>	The cell name for the configuration you want to open.
<i>t_viewName</i>	The view name for the configuration you want to open.
<i>t_mode</i>	A string. Must be <i>r</i> for read.
<i>?nameSpace t_nameSpace</i>	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.
<i>?dontOpenNextLevel g_dontOpenNextLevel</i>	By default, or if the value is <i>nil</i>, 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 <i>nlAction</i> property set to <i>ignore</i> are excluded from the handle. When set to a value other than <i>nil</i>, 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

<i>h_pcdBHandle</i>	The pcdB handle of the configuration opened.
<i>nil</i>	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.

An example for the use of this function can also be found at:

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbGetInstMasterGen](#)

[pcdbClose](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

[Example for all pcdb Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<i>t_libName</i>	The library name.
<i>t_cellName</i>	The cell name.
<i>t_viewName</i>	The view name.
<i>t_nameSpace</i>	The name space.

Value Returned

<i>t</i>	The view has pcdb information.
<i>nil</i>	The view does not have pcdb information.

Example

```
pcdbViewHasPcdbInfo("lib1" "cell1" "view1" "Verilog")
```

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Example for all pddb Functions

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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

<i>h_pcdBHandle</i>	A pcdB handle returned from a call to <code>pcdbOpen</code> .

Value Returned

<i>t</i>	The pcdB object is saved.
<i>nil</i>	The save operation failed.

Example

An example for the use of pcdB functions can be found at
`your_install_dir/tools/hdb/examples/hdbTraverse.il`

Related Topics

[pcdbOpen](#)

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

[Example for all pcdB Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

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)
    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"
```

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

`pcdbClose (pcdb`

Related Topics

[Hierarchy Database Functions](#)

[Occurrence Property Functions](#)

[Parent/Child Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Hierarchical Design Functions

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

This topic provides a list of Cadence® SKILL functions associated with the Virtuoso MultiTech Framework or Virtuoso RF Solution database functionality.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

■ Enable Virtuoso RF Solution and MultiTech Framework Functions

- ❑ [dbIsMultiTechEnabled](#)
- ❑ [dbIsVRFEnabled](#)
- ❑ [dbIsVRFInfraEnabled](#)

Note: These functions will be deprecated in a future release.

■ [Curved Shape Functions](#)

- ❑ [dbAddPinToPinGroup](#)
- ❑ [dbCreateCurvedAreaBoundary](#)
- ❑ [dbCreateCurvedPath](#)
- ❑ [dbGetCurvedPathRealWidth](#)
- ❑ [dbSetCurvedPathRealWidth](#)
- ❑ [dbCreateCurvedPolygon](#)
- ❑ [dbCreateMultipleCurvedPolygons](#)
- ❑ [dbCreatePlaceBound](#)
- ❑ [dbGetPlaceBounds](#)
- ❑ [dbIsCurved](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

- ☐ [dbGetCurvedAreaBoundaryPoints](#)
- ☐ [dbGetCurvedPolygonPoints](#)
- ☐ [dbSetCurvedAreaBoundaryPoints](#)
- ☐ [dbCreatePinGroup](#)
- ☐ [dbGetPinGroupPlaceBoundNameAndFace](#)
- ☐ [dbGetPinGroups](#)
- ☐ [dbSetCurvedPolygonPoints](#)
- ☐ [dbGetCurvedPolygonPoints](#)
- ☐ [dbSetCurvedAreaBoundaryPoints](#)
- ☐ [dbSetCurvedPolygonPoints](#)
- ☐ [dbCreateRegionBoundary](#)
- ☐ [dbGetPlaceBoundMaxHeight](#)
- ☐ [dbSetPlaceBoundMaxHeight](#)
- ☐ [dbGetPlaceBoundMinHeight](#)
- ☐ [dbSetPlaceBoundMinHeight](#)
- ☐ [dbGetDieStack](#)
- ☐ [dbGetDieStacks](#)
- ☐ [dbGetRegionBoundaryLayer](#)
- ☐ [dbGetRegionBoundaryLayerType](#)
- ☐ [dbGetRegionBoundaryName](#)
- ☐ [dbIsRegionBoundary](#)
- ☐ [dbMapPinGroupPins](#)

■ [Virtuoso RF Solution Package Infrastructure Functions](#)

- ☐ [pkxAddFuncDefPin](#)
- ☐ [pkxCreateCompDef](#)
- ☐ [pkxCreateCompDefPin](#)
- ☐ [pkxCreateFuncDef](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

- ☐ [pkxCreateFuncDefPin](#)
- ☐ [pkxDeleteCompDef](#)
- ☐ [pkxFindCompDef](#)
- ☐ [pkxFindFuncDefPin](#)
- ☐ [pkxFindFuncDef](#)
- ☐ [pkxFindFuncDefPin](#)
- ☐ [pkxGetCompDefPins](#)
- ☐ [pkxGetCompDefs](#)
- ☐ [pkxGetFuncDefPins](#)
- ☐ [pkxGetFuncDefs](#)
- ☐ [pkxGetProp](#)
- ☐ [pkxGetProps](#)
- ☐ [pkxSetProp](#)
- ☐ [pkxPadFile](#)
- ☐ [pkxUnPadFile](#)

Enable Virtuoso RF Solution and MultiTech Framework Functions

dbIsMultiTechEnabled

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

Description

(Virtuoso MultiTech Framework) 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
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

Related Topics

[Setting Up Virtuoso MultiTech Framework](#)

dbIsVRFEnabled

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

Description

(Virtuoso MultiTech Framework) Confirms whether the Virtuoso RF Solution-related functionality is enabled in the current session.

Note: This function will be deprecated in a future release.

Arguments

None.

Value Returned

t	Virtuoso RF Solution-related functionality is enabled.
nil	Virtuoso RF Solution-related functionality is not enabled.

Example

```
when(dbIsVRFEnabled()  
    pts = dbGetCurvedPolygonPoints(css())  
)
```

Related Topics

[Setting Up Virtuoso RF Solution](#)

dbIsVRFInfraEnabled

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

Description

(Virtuoso MultiTech Framework) Confirms whether the Virtuoso RF Solution 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	Virtuoso RF Solution infrastructure is enabled.
nil	Virtuoso RF Solution infrastructure is not enabled.

Example

```
when(dbIsVRFInfraEnabled()  
    pts = dbGetCurvedPolygonPoints(css())  
)
```

Related Topics

[Allegro translators](#)

[Setting Up Virtuoso MultiTech Framework](#)

Curved Shape Functions

dbAddPinToPinGroup

```
dbAddPinToPinGroup(  
    d_diePinGroup  
    d_inst  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework Only) Adds the padstack instance to the pin group.

Arguments

<i>d_diePinGroup</i>	The pin group created as a result of dbCreatePinGroup .
<i>d_inst</i>	The padstack instance that is associated with the given pin group.

Value Returned

<i>t</i>	The padstack instance is added.
<i>nil</i>	The operation fails.

Example

In the following example, the footprint cell of a die is modified by creating placeBounds. This is required to associate a pin group with the face (top or bottom) of another specific pin group. Then, a die TILP is instantiated. Finally, `dbAddPinToPinGroup` is run to associate a `padStack` to a specific die pin group.

```
dieFootprint = (dbOpenCellViewByType "testLib" "footprint" "base" "maskLayout"  
"w")  
  
placeBound1 = (dbCreatePlaceBound dieFootprint "left" list(0:0 100:100) 0 100)  
placeBound2 = (dbCreatePlaceBound dieFootprint "right" list(100:0 200:100) 50 100)  
  
pinGroupLeftTop = (dbCreatePinGroup dieFootprint "left" "top")  
pinGroupLeftBottom = (dbCreatePinGroup dieFootprint "left" "bottom")  
pinGroupRightTop = (dbCreatePinGroup dieFootprint "right" "top")  
pinGroupRightBottom = (dbCreatePinGroup dieFootprint "right" "bottom")
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
pinInstLeftTop1 = dbCreateParamInst(dieFootprint pinSuper "PILT1" 20:20 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstLeftTop2 = dbCreateParamInst(dieFootprint pinSuper "PILT2" 20:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightTop1 = dbCreateParamInst(dieFootprint pinSuper "PIRT1" 70:20 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightTop2 = dbCreateParamInst(dieFootprint pinSuper "PIRT2" 70:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstLeftBottom1 = dbCreateParamInst(dieFootprint pinSuper "PILB1" 20:20 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstLeftBottom2 = dbCreateParamInst(dieFootprint pinSuper "PILB2" 20:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightBottom1 = dbCreateParamInst(dieFootprint pinSuper "PIRB1" 20:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightBottom2 = dbCreateParamInst(dieFootprint pinSuper "PIRB2" 70:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )

dbAddPinToPinGroup(pinGroupLeftTop pinInstLeftTop1)
dbAddPinToPinGroup(pinGroupLeftTop pinInstLeftTop2)
dbAddPinToPinGroup(pinGroupLeftBottom pinInstLeftBottom1)
dbAddPinToPinGroup(pinGroupLeftBottom pinInstLeftBottom2)
dbAddPinToPinGroup(pinGroupRightTop pinInstRightTop1)
dbAddPinToPinGroup(pinGroupRightTop pinInstRightTop2)
dbAddPinToPinGroup(pinGroupRightBottom pinInstRightBottom1)
dbAddPinToPinGroup(pinGroupRightBottom pinInstRightBottom2)
```

Related Topics

[dbCreatePlaceBound](#)

[dbCreatePinGroup](#)

[dbGetPinGroups](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[License Requirements of Virtuoso RF Solution](#)

dbCreateCurvedAreaBoundary

```
dbCreateCurvedAreaBoundary(  
    d_cellViewId  
    l_curvedPoints  
)  
=> d_curvedAreaId / nil
```

Description

(Virtuoso MultiTech Framework) Creates a curved area boundary in the specified cellview, using the specified points.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview to hold the new object.
<i>l_curvedPoints</i>	<pre>list(boundary [l_holes])</pre> List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a <code>dbClosedCurve</code>, which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular: ■ A straight edge is indicated by two end points without an arc qualifier on the beginning edge. ■ A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid. The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

```
L_curvedPoints := (elem)*
elem := {<x:float> <y:float> ([("arc" |
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"
<angle:float>)])}
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

Value Returned

<i>d_curvedAreaId</i>	Database ID of a curved area object.
<i>nil</i>	The operation fails.

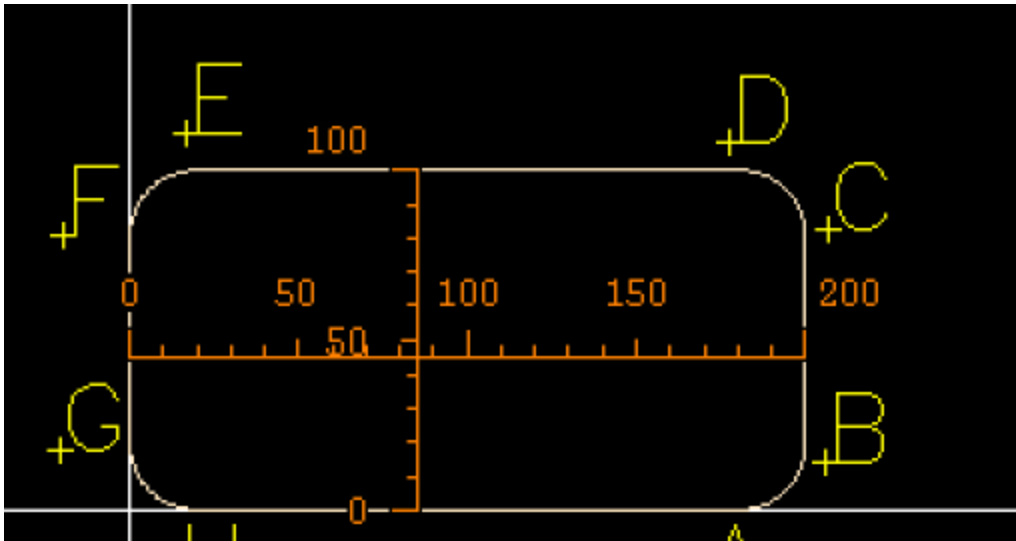
Example

Creates curved area boundary with rounded corners.

```
dbCreateCurvedAreaBoundary( cv
' (
  (180 0 "arcCentralAngle" 90)    ; A
  (200 20)                        ; B
  (200 80 "arcCentralAngle" 90)   ; C
  (180 100)                       ; D
  (20 100 "arcCentralAngle" 90)   ; E
  (0 80)                          ; F
  (0 20 "arcCentralAngle" 90)     ; G
  (20 0)                          ; H
)
)
```

Here:

- The radius of the curved corners is 20 .
- An implicit arc with 90 degrees is used on all four corners.



Creates curved area boundary with holes.

```
solid = '(
    (180 0 "arcCentralAngle" 90)    ;A
    (200 20)                        ;B
    (200 80 "arcCentralAngle" 90)  ;C
    (180 100)                       ;D
    (20 100 "arcCentralAngle" 90)  ;E
    (0 80)                          ;F
    (0 20 "arcCentralAngle" 90)    ;G
    (20 0)                          ;H
)
```

```
hole1 = '(
    (20 40)
    (40 40 "arc" "ccw")
    (60 40)
    (40 40 "arc" "ccw")
)
```

```
x=160
y=60
r=20
```

```
hole2 = list(
    list(x-r y)
```

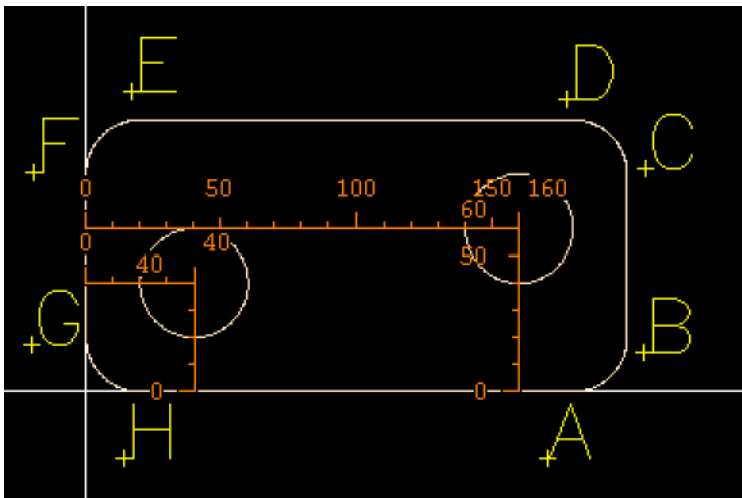
Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
list(x y "arc" "ccw")  
list(x+r y)  
list(x y "arc" "ccw")  
)
```

```
dbCreateCurvedAreaBoundary(cv list(solid list(hole1 hole2)))
```

Considers that the center of the holes is known, therefore the explicit arc form is used. Illustrates how the explicit arc points are calculated from center point and radius in `hole2`.



Related Topics

[License Requirements of Virtuoso RF Solution](#)

dbCreateCurvedPath

```
dbCreateCurvedPath(  
    d_cellViewId  
    txl_layerPurpose  
    l_curvedPoints  
    x_width  
)  
=> d_curvedPathId
```

Description

(Virtuoso MultiTech Framework) Creates a curved path that has a straight centerline and round edges at all segments.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layers or layer-purpose pairs.
<i>l_curvedPoints</i>	<code>list(boundary [l_holes])</code> List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a <code>dbClosedCurve</code> , which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular: ■ A straight edge is indicated by two end points without an arc qualifier on the beginning edge. ■ A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid. The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

```
L_curvedPoints := (elem)*  
elem := {<x:float> <y:float> ([("arc" |  
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"  
<angle:float>)])}
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

x_width Width of the curved path.

Value Returned

d_curvedPathId Database ID of the curved path that is created.

Example

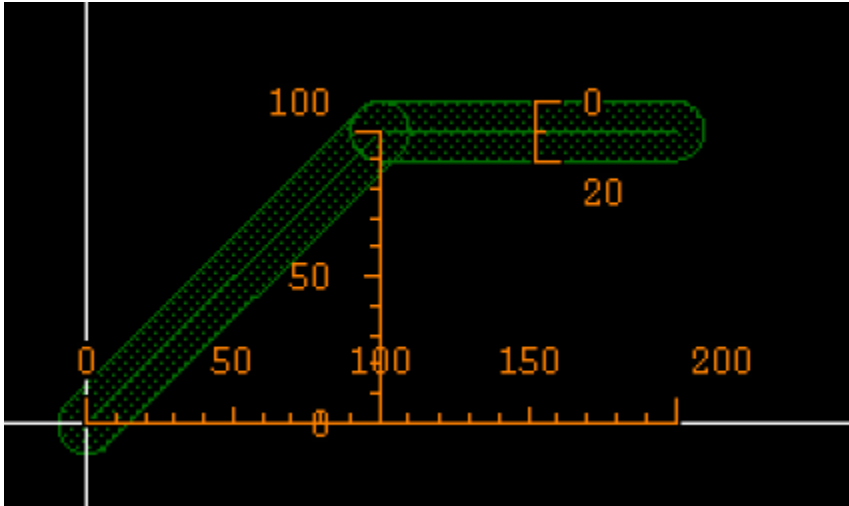
A curved path is created based on the specified list of points and layer-purpose as "DIE_PAD" "drawing".

```
dbCreateCurvedPath( cv ' ("BD4" "drawing")  
' (  
    (0 0)  
    (100 100)  
    (200 100)  
)  
  
20  
)
```


Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

The created path has three center line points, resulting in two segments and the width of the path is 20.



Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbCreateMultipleCurvedPolygons](#)

[Database Access Function Argument Types](#)

dbGetCurvedPathRealWidth

```
dbGetCurvedPathRealWidth(  
    d_curvedPath  
)  
=> l_realWidth / nil
```

Description

(Virtuoso MultiTech Framework) Returns the real width that has been set on the path by dbSetCurvedPathRealWidth.

Arguments

<i>d_curvedPath</i>	The name of the path.

Value Returned

<i>l_realWidth</i>	The value of the width set in DBUs.
<i>nil</i>	The operation fails.

Example

The following code creates a curved path of width 3 and then changes it to 2

```
cPath = dbCreateCurvedPath(cv list("DIE_PAD" "drawing") list((0:0) (0:10) (10:10)  
(10:0)) 3)  
printf("width= %L\n" dbGetCurvedPathRealWidth(cPath))  
dbSetCurvedPathRealWidth(cPath 2)  
printf("width= %L\n" dbGetCurvedPathRealWidth(cPath))
```

Related Topics

[dbCreateCurvedPath](#)

[dbSetCurvedPathRealWidth](#)

[License Requirements of Virtuoso RF Solution](#)

[dbCreateMultipleCurvedPolygons](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

Database Access Function Argument Types

dbSetCurvedPathRealWidth

```
dbSetCurvedPathRealWidth(  
    d_curvedPath  
    l_realWidth  
)  
=> d_curvedPathId
```

Description

(Virtuoso MultiTech Framework) Sets the real width of the curved path. The real width could be 1DBU or more.

Arguments

<i>d_curvedPath</i>	The name of the path.
<i>l_realWidth</i>	The value of the width to be set in DBUs.

Value Returned

<i>t</i>	The width is set successfully.
<i>nil</i>	The operation fails.

Example

The following code creates a curved path of width 3 and then sets it to 2.

```
cPath = dbCreateCurvedPath(cv list("DIE_PAD" "drawing") list((0:0) (0:10) (10:10)  
(10:0)) 3)  
printf("width= %L\n" dbGetCurvedPathRealWidth(cPath))  
dbSetCurvedPathRealWidth(cPath 2)  
printf("width= %L\n" dbGetCurvedPathRealWidth(cPath))
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbCreateCurvedPath](#)

[dbGetCurvedPathRealWidth](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

dbCreateMultipleCurvedPolygons

Database Access Function Argument Types

dbCreateCurvedPolygon

```
dbCreateCurvedPolygon(  
    d_CellViewId  
    txl_layerPurpose  
    l_curvedPoints  
)  
=> d_curvedPolygonId
```

Description

(Virtuoso MultiTech Framework) Creates a curved polygon.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layers or layer-purpose pairs.
<i>l_curvedPoints</i>	<code>list(boundary [l_holes])</code> List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a <code>dbClosedCurve</code> , which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular: ■ A straight edge is indicated by two end points without an arc qualifier on the beginning edge. ■ A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid. The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

```
L_curvedPoints := (elem)*  
elem := {<x:float> <y:float> ([("arc" |  
"arcByCenter") ("cw" | "ccw")) | [ "arcByAngle"  
<angle:float>] ) }
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

Value Returned

d_curvedPolygonId Database ID of the curved polygon that is created.

Example

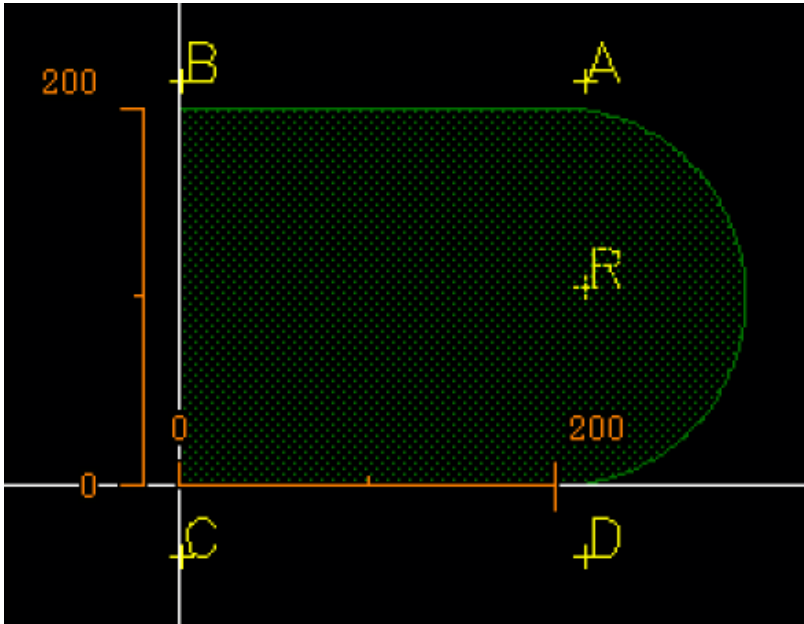
```
cp1 = list(list(0 0) list(200 0) list(200 100 "arc" "ccw") list(200 200) list(0  
200))  
dbCreateCurvedPolygon(cv list("DIE_PAD" "drawing") cp1)
```

A curved polygon is created based on the specified list of points and layer-purpose as "DIE_PAD" "drawing".

Additional Information

Creating Arcs

Consider a curved polygon with three straight edges and one circular edge:



There are three ways for creating this polygon:

```
let( (cv cp0 cp1a cp1b cp2a cp2b cp3 polygon)
  cv = geGetEditCellView()
  mapcar('dbDeleteObject cv~>shapes)
```

```
  cp0 = '(
    (200 200)           ;A
    (0 200)            ;B
    (0 0)              ;C
    (200 0)            ;D
    (200 100 "arc" "ccw") ;R
  )
```

```
  cp1a = '(
    (0 0)              ;C
    (200 0)           ;D
    (200 100 "arc" "ccw") ;R
    (200 200)         ;A
    (0 200)           ;B
  )
```


Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
cp1b = '(
    (0 0) ;C
    (200 0) ;D
    (200 100 "arcByCenter" "ccw") ;R
    (200 200) ;A
    (0 200) ;B
)

cp2a = '(
    (0 0) ;C
    (200 0 "arcCentralAngle" 180) ;D
    (200 200) ;A
    (0 200) ;B
)

cp2b = '(
    (0 0) ;C
    (200 0 "arcByAngle" 180) ;D
    (200 200) ;A
    (0 200) ;B
)

cp3 = '(
    (0 0) ;C
    (200 0) ;D
    ("arcCentralAngle" 180)
    (200 200) ;A
    (0 200) ;B
)

foreach(points list(cp0 cp1a cp1b cp2a cp2b cp3)
    polygon = dbCreateCurvedPolygon(cv '("BD4" "drawing") points)

    assert(dbGetCurvedPolygonPoints(polygon) == cp0)
)
)
```

Explicit arcs:

- cp0, cp1 and cp2 show examples of explicit arcs.

- The center point `R` is explicitly specified.
- The keywords `"arc"` and `"arcByCenter"` can be used interchangeably.
- Explicit arcs are useful for circles around a known center point.

Implicit arcs:

- `cp2a`, `cp2b` and `cp3` show examples of implicit arcs.
- No radius is specified, only points on the actual boundary are listed.
- Implicit arcs are useful when the boundary points are known.
- The keywords `"arcCentralAngle"` and `"arcByAngle"` can be used interchangeably.

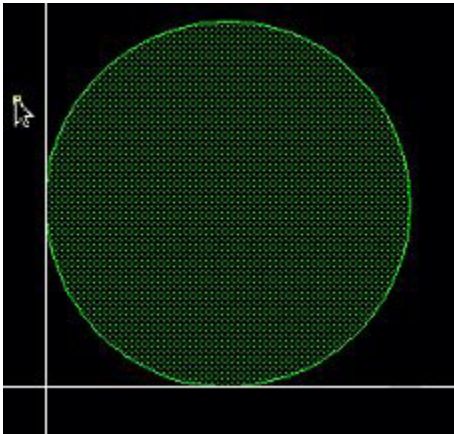
Regardless of how the curved polygon is created, `dbGetCurvedPolygonPoints` always returns the edges in `cp0`.

Creating a circle

To create a circle:

- Create a counter clockwise circular edge from `(0 100)` to `(200 100)` using `(100 100)` as the center.
- Create a counter clockwise circular edge from `(200 100)` to `(0 100)` using `(100 100)` as the center.

```
dbCreateCurvedPolygon cvId list("SURFACE" "drawing")
' (
  (0 100)
  (100 100 "arc" "ccw")
  (200 100)
  (100 100 "arc" "ccw")
)
```



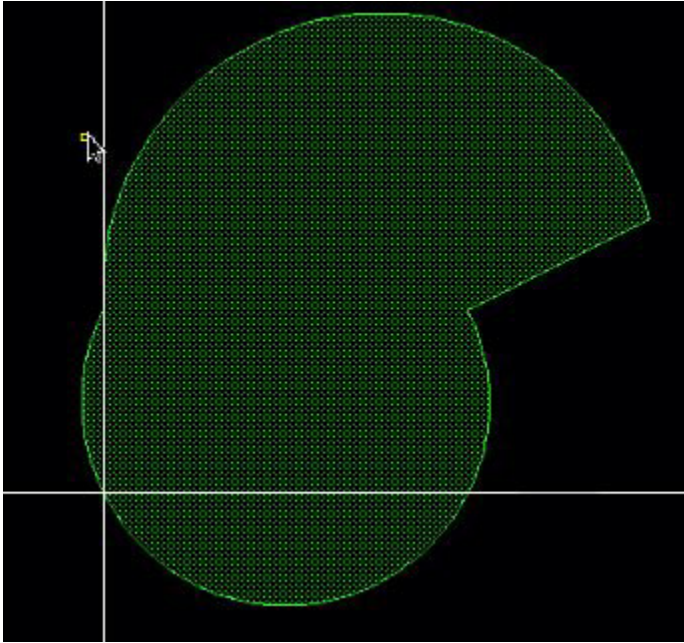
Creating a beak

To create a beak:

- Create a counter clockwise circular edge from (0 100) to (200 100) using (100 50) as the arc center
- Create a straight edge from (200 100) to (300 150)
- Create a counter clockwise circular edge from (300 150) to (0 100) using (50 100) as the arc center

```
dbCreateCurvedPolygon cv '("BD4" "drawing")
'(
    (0 100)
    (100 50 "arc" "ccw")
    (200 100)
    (300 150)
    (50 100 "arc" "ccw")
)
```

)



Creating a tombstone with eyes

To create a tombstone with eyes, you need to create the following:

1. Create a solid boundary.
 - a. Create a straight edge from (0 0) to (200 0).
 - b. Create a straight edge from (200 0) to (200 200).
 - c. Create a counter clockwise circular arc from (200 200) to (0 200) using (100 200) as the arc center.
 - d. Create a straight edge (0 200) to (0 0).

```
solid = '(  
    (0 0)  
    (200 0)  
    (200 200)  
    (100 200 "arc" "ccw")  
    (0 200)  
)
```

2. Create the first hole (hole1).
 - a. Create a counter clockwise circular arc from (50 50) to (75 75). The center is computed to create a subtended angle of 180 degree.

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

- b. Create a counter clockwise circular arc from (75 75) to (50 50). The center is computed to create a subtended angle of 180 degree.

```
hole1 = '(  
    (50 50 "arcCentralAngle" 180)  
    (75 75 "arcCentralAngle" 180)  
)
```

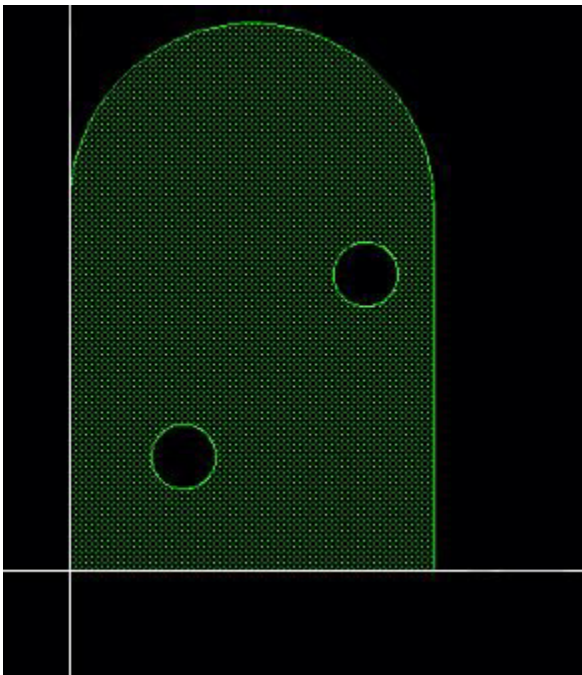
3. Create the second hole (hole2)

- a. Create a counter clockwise circular arc from (150 150) to (175 175). The center is computed to create a subtended angle of 180 degrees.
- b. Create a counter clockwise circular arc from (175 175) to (150 150). The center is computed to create a subtended angle of 180 degrees.

```
hole2 = '(  
    (150 150 "arcCentralAngle" 180)  
    (175 175 "arcCentralAngle" 180)  
)
```

4. Create the polygon itself using all the three curves.

```
dbCreateCurvedPolygon(cv "BD4" "drawing") list(solid list(hole1 hole2)))
```



Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbCreateCurvedPolygon](#)

[Database Access Function Argument Types](#)

dbCreateMultipleCurvedPolygons

```
dbCreateMultipleCurvedPolygons (
    d_cellViewId
    txl_layerPurpose
    l_curvedPoints
)
=> list(d_curvedPolygonIds) / nil
```

Description

(Virtuoso MultiTech Framework) Creates multiple curved polygons in the specified cellview on the specified layer or layer-purpose pair using the specified points.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>txl_layerPurpose</i>	Name, number, or a list of layers or layer-purpose pairs.
<i>l_curvedPoints</i>	<pre>list(boundary [l_holes])</pre> List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a <code>dbClosedCurve</code>, which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular: ■ A straight edge is indicated by two end points without an arc qualifier on the beginning edge. ■ A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid. The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

```
L_curvedPoints := (elem)*  
  
elem := {<x:float> <y:float> ([("arc" |  
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"  
<angle:float>)]}}
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw"(counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

In dbCreateMultipleCurvedPolygons, this information is used to split the boundary into multiple polygons by the holes. It is possible to choose the holes in such a way that the original boundary is no longer a single shape and is split. When it is possible to split the original boundary, dbCreateMultipleCurvedPolygons is used to create one or more polygons, depending on into how many polygons the boundary was split.

Value Returned

```
list(d_curvedPolygonIds)
```

List of IDs of created curved polygons.

```
nil
```

An error occurred.

Example

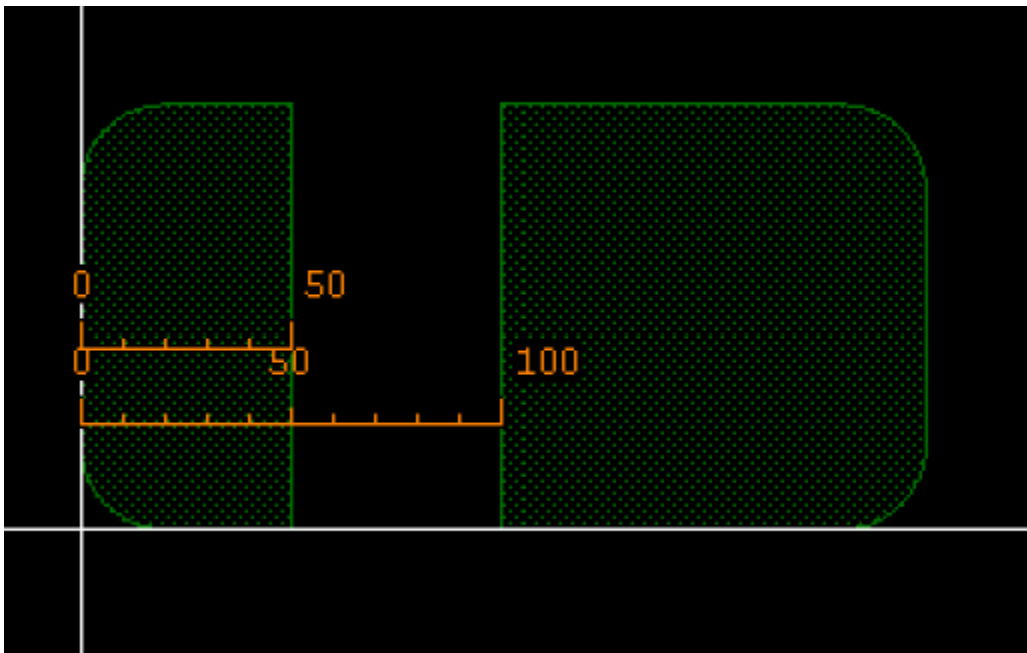
The boundary is sliced into two by a big hole shape to create two polygons.

```
solid = '(  
  (180 0 "arcCentralAngle" 90)  
  (200 20)  
  (200 80 "arcCentralAngle" 90)  
  (180 100)  
  (20 100 "arcCentralAngle" 90)  
  (0 80)  
  (0 20 "arcCentralAngle" 90)  
  (20 0)
```


Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
)  
  
hole = '(  
  (50 0)  
  (100 0)  
  (100 100)  
  (50 100)  
)  
  
dbCreateMultipleCurvedPolygons(cv '("BD4" "drawing") list(solid list(hole)))
```



dbCreatePlaceBound

```
dbCreatePlaceBound(  
    d_cellView  
    l_bBox  
    x_minHeight  
    x_maxHeight  
)  
=> d_pb / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Creates a placeBound using the values specified through the arguments. A placeBound is created within a die footprint cell and is used to represent the die thickness.

Arguments

<i>d_cellView</i>	The cellview in which the placeBound is to be created.
<i>l_bBox</i>	List of coordinates specifying the bounding box of the placeBound.
<i>x_minHeight</i>	The minimum thickness of a die.
<i>x_maxHeight</i>	The maximum thickness of a die.

Return value:

<u><i>d_pb</i></u>	The placeBound was created.
<i>nil</i>	The placeBound could not be created.

Example

```
placeBound1 = (dbCreatePlaceBound cv "pb1" list(0:0 100:100) 0 100)  
placeBound2 = (dbCreatePlaceBound cv "pb2" list(100:0 200:100) 50 100)
```

Two placeBounds, placeBound1 and placeBound2, are created.

Related Topics

[dbCreatePinGroup](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[dbGetPinGroups](#)

[License Requirements of Virtuoso RF Solution](#)

[Database Access Function Argument Types](#)

dbGetPlaceBounds

```
dbGetPlaceBounds (
    d_cellView
)
=> d_pb d_pb d_pb... / nil
```

Description

(ICADVM20.1 Virtuoso MultiTech Framework Only) Returns the list of placeBounds from the given cellview.

Arguments

<i>d_cellView</i>	The cellview in which the placeBounds are created.

Return value:

<i>d_pb</i>	The list of placeBounds in the cellview.

<i>nil</i>	No placeBounds were found.

Example

```
placeBound1 = (dbCreatePlaceBound cv "pb1" list(0:0 100:100) 0 100)
placeBound2 = (dbCreatePlaceBound cv "pb2" list(100:0 200:100) 50 100)
\i (sort (dbGetPlaceBounds cv)~>name nil)
\t ("pb1" "pb2")
\p >
```

Two placeBounds, placeBound1 and placeBound2, are created and then retrieved.

Related Topics

[dbCreatePlaceBound](#)

[dbCreatePinGroup](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[dbGetPinGroups](#)

[License Requirements of Virtuoso RF Solution](#)

Virtuoso Studio Design Environment SKILL Reference
Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

Database Access Function Argument Types

dbIsCurved

```
dbIsCurved(  
    d_shapeId  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework) Checks whether the specified shape is a curved shape.

Arguments

<i>d_shapeId</i>	Database ID of a shape.

Return value:

<i>t</i>	The specified shape is a curved shape.
<i>nil</i>	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.

Related Topics

License Requirements of Virtuoso RF Solution

Virtuoso Studio Design Environment SKILL Reference
Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

Database Access Function Argument Types

dbGetCurvedAreaBoundaryPoints

```
dbGetCurvedAreaBoundaryPoints(  
    d_curvedAreaBoundaryId  
)  
=> l_curvedPoints / nil
```

Description

(Virtuoso MultiTech Framework) 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_curvedPoints list(boundary [l_holes])

List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a `dbClosedCurve`, which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular:

- A straight edge is indicated by two end points without an arc qualifier on the beginning edge.
- A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid.

The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
L_curvedPoints := (elem)*
```

```
elem := {<x:float> <y:float> ([("arc" |  
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"  
<angle:float>)])}
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

```
nil
```

The operation failed or the function is called on a non-curved shape.

Example

```
points = dbGetCurvedAreaBoundaryPoints(areaB)
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

dbGetCurvedPolygonPoints

```
dbGetCurvedPolygonPoints(  
    d_curvedPolygonId  
    l_curvedPoints  
)  
=> l_curvedPoints / nil
```

Description

(Virtuoso MultiTech Framework) Retrieves the curved point list for the specified curved polygon.

Arguments

<i>d_curvedPolygonId</i>	Database ID of a curved polygon.

Value Returned

<i>l_curvedPoints</i>	<pre>list(boundary [l_holes])</pre> List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a <code>dbClosedCurve</code>, which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular: ■ A straight edge is indicated by two end points without an arc qualifier on the beginning edge. ■ A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid. The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

```
L_curvedPoints := (elem)*
```

```
elem := {<x:float> <y:float> ([("arc" |  
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"  
<angle:float>)])}
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

nil

The operation failed or the function is called on a non-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)  
)  
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`.

Related Topics

License Requirements of Virtuoso RF Solution

dbSetCurvedAreaBoundaryPoints

```
dbSetCurvedAreaBoundaryPoints
    d_curvedAreaBoundaryId
    l_curvedPoints
)
=> t / nil
```

Description

(Virtuoso MultiTech Framework) Sets curved points for the specified curved area boundary object.

Arguments

d_curvedAreaBoundaryId

Database ID of a curved area boundary object.

l_curvedPoints

`list(boundary [l_holes])`

List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a `dbClosedCurve`, which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular:

- A straight edge is indicated by two end points without an arc qualifier on the beginning edge.
- A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid.

The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
L_curvedPoints := (elem)*  
elem := {<x:float> <y:float> ([("arc" |  
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"  
<angle:float>)] )}
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

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)  
dbSetCurvedAreaBoundaryPoints(areaB points)
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

dbCreatePinGroup

```
dbCreatePinGroup(  
    d_cellView  
    t_name  
    t_faceType  
)  
=> d_dpg / nil
```

Description

(Virtuoso MultiTech Framework Only) Returns the pin group created with the given arguments.

Arguments

<i>d_cellView</i>	The cellview in which the pin group is created.
<i>t_name</i>	The name of a placeBound. For details, see dbCreatePlaceBound .
<i>t_faceType</i>	The top or the bottom face of a placeBound.

Value Returned

<i>d_dpg</i>	The name of the pin group that has been created.
<i>nil</i>	The operation failed.

Example

The following example modifies a die footprint cell by creating placeBounds and die pin groups.

```
dieFootprint = (dbOpenCellViewByType "testLib" "footprint" "base" "maskLayout"  
"w")  
  
placeBound1 = (dbCreatePlaceBound dieFootprint "left" list(0:0 100:100) 0 100)  
placeBound2 = (dbCreatePlaceBound dieFootprint "right" list(100:0 200:100) 50 100)  
  
pinGroupLeftTop = (dbCreatePinGroup dieFootprint "left" "top")  
pinGroupLeftBottom = (dbCreatePinGroup dieFootprint "left" "bottom")  
pinGroupRightTop = (dbCreatePinGroup dieFootprint "right" "top")
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
pinGroupRightBottom = (dbCreatePinGroup dieFootprint "right" "bottom")
```

Related Topics

[dbCreatePlaceBound](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[dbGetPinGroups](#)

[dbMapPinGroupPins](#)

[License Requirements of Virtuoso RF Solution](#)

dbGetPinGroupPlaceBoundNameAndFace

```
dbGetPinGroupPlaceBoundNameAndFace(  
    d_diePinGroup  
)  
=> list(placeBoundName, placeBoundFace) / nil
```

Description

(Virtuoso MultiTech Framework Only) Returns the placeBound name and the placeBound face of the associated pin group.

Arguments

<i>d_cellView</i>	The cellview to obtain the pin groups.

Value Returned

<i>list(placeBoundName, placeBoundFace)</i>	The name and face of the placeBound.
<i>nil</i>	The operation failed.

Example

A die footprint cell is modified by creating a placeBound and a die pin group before calling dbGetPinGroupPlaceBoundNameAndFace.

```
dieFootprint = (dbOpenCellViewByType "testLib" "footprint" "base" "maskLayout"  
"w")  
  
placeBound1 = (dbCreatePlaceBound dieFootprint "left" list(0:0 100:100) 0 100)  
placeBound2 = (dbCreatePlaceBound dieFootprint "right" list(100:0 200:100) 50 100)  
  
pinGroupLeftTop = (dbCreatePinGroup dieFootprint "left" "top")  
pinGroupLeftBottom = (dbCreatePinGroup dieFootprint "left" "bottom")  
pinGroupRightTop = (dbCreatePinGroup dieFootprint "right" "top")  
pinGroupRightBottom = (dbCreatePinGroup dieFootprint "right" "bottom")  
  
\i (dbGetPinGroupPlaceBoundNameAndFace pinGroupLeftTop)  
\t ("left" "top")
```


Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
\p >
\i (dbGetPinGroupPlaceBoundNameAndFace pinGroupLeftBottom)
\t ("left" "bottom")
\p >
\i (dbGetPinGroupPlaceBoundNameAndFace pinGroupRightTop)
\t ("right" "top")
\p >
\i (dbGetPinGroupPlaceBoundNameAndFace pinGroupRightBottom)
\t ("right" "bottom")
\p >
```

Related Topics

[dbAddPinToPinGroup](#)

[dbCreatePinGroup](#)

[dbGetPinGroups](#)

[dbMapPinGroupPins](#)

[License Requirements of Virtuoso RF Solution](#)

dbGetPinGroups

```
dbGetPinGroups (
    d_cellView
)
=> d_dpg d_dpg ... / nil
```

Description

(Virtuoso MultiTech Framework Only) Returns the pin groups defined in the specified cellview.

Arguments

<i>d_cellView</i>	The cellview to obtain the pin groups.

Value Returned

<i>d_dpg</i>	The list of pin groups in the cellview.
<i>nil</i>	The operation failed.

Example

A die footprint cell is modified by creating placeBounds and die pin groups. Subsequently, dbGetPinGroups is used to list the newly created pin groups.

```
dieFootprint = (dbOpenCellViewByType "testLib" "footprint" "base" "maskLayout"
"w")

placeBound1 = (dbCreatePlaceBound dieFootprint "left" list(0:0 100:100) 0 100)
placeBound2 = (dbCreatePlaceBound dieFootprint "right" list(100:0 200:100) 50 100)

pinGroupLeftTop = (dbCreatePinGroup dieFootprint "left" "top")
pinGroupLeftBottom = (dbCreatePinGroup dieFootprint "left" "bottom")
pinGroupRightTop = (dbCreatePinGroup dieFootprint "right" "top")
pinGroupRightBottom = (dbCreatePinGroup dieFootprint "right" "bottom")

\i (sort (dbGetPinGroups dieFootprint)~>name nil)
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
\t ("left_PLACEBOUND_FACE_BOTTOM" "left_PLACEBOUND_FACE_TOP"  
"right_PLACEBOUND_FACE_BOTTOM" "right_PLACEBOUND_FACE_TOP")  
\p >
```

Related Topics

[dbCreatePlaceBound](#)

[dbCreatePinGroup](#)

[dbAddPinToPinGroup](#)

[dbGetPinGroups](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[License Requirements of Virtuoso RF Solution](#)

dbSetCurvedPolygonPoints

```
dbSetCurvedPolygonPoints(  
    d_polygonId  
    l_curvedPoints  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework) 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(boundary [l_holes])

List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a `dbClosedCurve`, which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular:

- A straight edge is indicated by two end points without an arc qualifier on the beginning edge.
- A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid.

The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
L_curvedPoints := (elem)*  
elem := {<x:float> <y:float> ([("arc" |  
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"  
<angle:float>)] )}
```

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

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.
```

Related Topics

License Requirements of Virtuoso RF Solution

dbCreateRegionBoundary

```
dbCreateRegionBoundary(  
    d_cellViewId  
    l_curvedPoints  
    t_regionName  
    [ t_layerType ]  
)  
=> d_regionBoundaryId / nil
```

Description

(Virtuoso MultiTech Framework) 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.

Arguments

d_cellViewId Database ID of a cellview.

l_curvedPoints list(boundary [l_holes])

List of one boundary and zero or more holes forming a complex closed curve. Each boundary or hole is represented by a `dbClosedCurve`, which is a list of vertexes. In a vertex, pairs of adjacent vertexes represent edges, which can be straight or circular:

- A straight edge is indicated by two end points without an arc qualifier on the beginning edge.
- A circular edge may be indicated by explicitly specifying the center or implicitly specifying the center. The explicit specification requires the center to be on the DBU grid. The implicit specification does not require the center to be on the DBU grid.

The points list can be a mix of explicit center-based or implicit angle-based specifications. Each element in the list describes a vertex on the curved shape.

`L_curvedPoints := (elem)*`

`elem := {<x:float> <y:float> ([("arc" |
"arcByCenter") ("cw" | "ccw")) | [("arcByAngle"
<angle:float>)])}`

Here, keywords for explicit specification are "arc" and "arcByCenter" followed by "cw"(clockwise), or "ccw" (counter clockwise), to specify the direction. Keywords for implicit specification, where you do not define the center explicitly, are "arcCentralAngle" and "arcByAngle". In implicit specification, to specify a clockwise arc, specify a negative value for the angle specification.

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

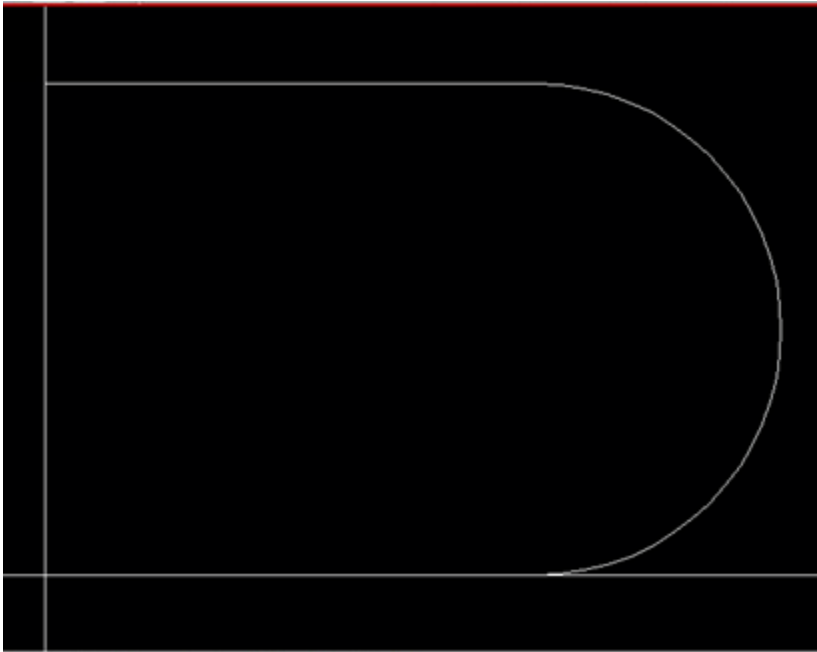
d_regionBoundaryId

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" "Metall")
```



Related Topics

[License Requirements of Virtuoso RF Solution](#)

dbGetPlaceBoundMaxHeight

```
dbGetPlaceBoundMaxHeight (
    d_placeBound
)
=> x_maxHeight / nil
```

Description

(Virtuoso MultiTech Framework Only) Returns the maximum thickness of a placeBound.

Arguments

<i>d_placeBound</i>	The placeBound pointer.

Value Returned

<i>x_maxHeight</i>	The maximum thickness of a die.
<i>nil</i>	The maximum thickness could not be obtained.

Example

```
placeBound1 = (dbCreatePlaceBound cv "pb1" list(0:0 100:100) 0 100)

\i (dbGetPlaceBoundMaxHeight placeBound1)
\t 100
\p >
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbSetPlaceBoundMaxHeight](#)

[dbCreatePinGroup](#)

[dbAddPinToPinGroup](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

dbGetPinGroups

dbGetPinGroupPlaceBoundNameAndFace

dbSetPlaceBoundMaxHeight

```
dbSetPlaceBoundMaxHeight (
    d_placeBound
)
=> x_maxHeight / nil
```

Description

(Virtuoso MultiTech Framework Only) Sets the maximum thickness of a placeBound.

Arguments

d_placeBound

The placeBound pointer.

x_maxHeight

The maximum thickness of a die to be set.

Value Returned

t

The maximum thickness has been defined successfully.

nil

The maximum thickness could not be defined.

Example

```
placeBound1 = (dbCreatePlaceBound cv "pb1" list(0:0 100:100) 0 100)
```

```
\i (dbSetPlaceBoundMaxHeight placeBound1 250)
```

```
\t t
```

```
\p >
```

```
\i (dbGetPlaceBoundMaxHeight placeBound1)
```

```
\t 250
```

```
\p >
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbGetPlaceBoundMaxHeight](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

dbCreatePinGroup

dbAddPinToPinGroup

dbGetPinGroupPlaceBoundNameAndFace

dbGetPinGroups

dbGetPinGroupPlaceBoundNameAndFace

dbGetPlaceBoundMinHeight

```
dbGetPlaceBoundMinHeight (
    d_placeBound
)
=> x_minHeight / nil
```

Description

(Virtuoso MultiTech Framework Only) Returns the minimum thickness of a placeBound.

Arguments

<i>d_placeBound</i>	The placeBound pointer.

Value Returned

<i>x_minHeight</i>	The minimum thickness of a die.
<i>nil</i>	The minimum thickness could not be obtained.

Example

```
placeBound1 = (dbCreatePlaceBound cv "pb1" list(0:0 100:100) 0 100)

\i (dbGetPlaceBoundMinHeight placeBound1)
\t 75
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbSetPlaceBoundMinHeight](#)

[dbCreatePinGroup](#)

[dbAddPinToPinGroup](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[dbGetPinGroups](#)

Virtuoso Studio Design Environment SKILL Reference
Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

dbGetPinGroupPlaceBoundNameAndFace

dbSetPlaceBoundMinHeight

```
dbSetPlaceBoundMinHeight (
    d_placeBound
)
=> x_minHeight / nil
```

Description

(Virtuoso MultiTech Framework Only) Sets the minimum thickness of a placeBound.

Arguments

d_placeBound

The placeBound pointer.

x_minHeight

The minimum thickness of a die to be set.

Value Returned

t

The minimum thickness has been defined successfully.

nil

The minimum thickness could not be set.

Example

```
placeBound1 = (dbCreatePlaceBound cv "pb1" list(0:0 100:100) 0 100)
```

```
\i (dbSetPlaceBoundMinHeight placeBound1 75)
```

```
\t t
```

```
\p >
```

```
\i (dbGetPlaceBoundMinHeight placeBound1)
```

```
\t 75
```

```
\p >
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbGetPlaceBoundMinHeight](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

dbCreatePinGroup

dbAddPinToPinGroup

dbGetPinGroupPlaceBoundNameAndFace

dbGetPinGroups

dbGetPinGroupPlaceBoundNameAndFace

dbGetDieStack

```
dbGetDieStack(  
    d_inst  
)  
=> d_ds / nil
```

Description

(Virtuoso MultiTech Framework Only) Returns a dieStack from the given instance.

Arguments

<i>d_inst</i>	The instance from which to obtain the die stack.

Value Returned

<i>d_ds</i>	The die stack of the given instance.
<i>nil</i>	No die stack has been found.

Example

```
top = (dbOpenCellViewByType libName "Top" "layout" "maskLayout" "w")  
inst1 = (dbCreateParamInst top super "I1" 0:0 "R0" 1 list(list("footprintCellName"  
"string" "LNA_PKG_100")) )  
inst2 = (dbCreateParamInst top super "I2" 0:0 "R0" 1 list(list("footprintCellName"  
"string" "LNA_PKG_100")) )  
\i (dbGetDieStack inst1) == (dbGetDieStack inst2)  
\t t  
\p >
```

Related Topics

[dbCreatePlaceBound](#)

[dbGetDieStacks](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[dbGetPlaceBoundMinHeight](#)

Virtuoso Studio Design Environment SKILL Reference
Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

License Requirements of Virtuoso RF Solution

dbGetDieStacks

```
dbGetDieStacks(  
    d_cellView  
)  
=> d_cellView / nil
```

Description

(Virtuoso MultiTech Framework Only) Returns the list of die stacks from the given cellview.

Arguments

<i>d_cellView</i>	The cellview from which to obtain the die stacks.

Value Returned

<i>tx_regionLayer</i>	The cellview that contains the die stacks.
<i>nil</i>	No die stacks have been created.

Example

There are two die stacks. One holds I1, I2, I5, and the other holds I3 and I4. The function returns two die stacks.

```
; +-----+      +-----+  
; |   I5   |      |           |  
; |         |      |           |  
; +-----+      |           |  
; +-----+      |           |  
; |   I2   |      |           |  
; |         |      |   I4    |  
; +-----+      |           |  
; +-----+      +-----+  
; |   I1   |      +-----+  
; |         |      |   I3    |  
; +-----+      +-----+  
\i ;CDSFILTER:CRI  
\i (length (dbGetDieStacks top))  
\t 2
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

\p >

Related Topics

[dbCreatePlaceBound](#)

[dbGetPinGroupPlaceBoundNameAndFace](#)

[dbGetPlaceBoundMinHeight](#)

[License Requirements of Virtuoso RF Solution](#)

dbGetRegionBoundaryLayer

```
dbGetRegionBoundaryLayer(  
    d_regionBoundaryID  
)  
=> tx_regionLayer / nil
```

Description

(Virtuoso MultiTech Framework) Returns the layer name or number of the specified region boundary.

Note: This function requires layerType to be "layer".

Arguments

<i>d_regionBoundaryID</i>	Database ID of a region boundary.

Value Returned

<i>tx_regionLayer</i>	Layer name, if found, or number of the region boundary layer.
<i>nil</i>	An error occurred or the layer type is not "layer".

Example

```
regionLayer = dbGetRegionBoundaryLayer( regbId )  
=> "METAL4"
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

[dbGetRegionBoundaryLayerType](#)

dbGetRegionBoundaryLayerType

```
dbGetRegionBoundaryLayerType (
    d_regionBoundaryId
)
=> t_regionLayerType / nil
```

Description

(Virtuoso MultiTech Framework) Returns the region boundary layer type.

Arguments

d_regionBoundaryId

Database ID of a region boundary.

Value Returned

<i>t_regionLayerType</i>	Layer type of a region boundary. Valid values: "all", "layer", "outsideLayers", "insidePlaneLayers", "insideSignalLayers"
<i>nil</i>	An error occurred.

Example

```
layerType = dbGetRegionBoundaryLayerType( regbId )
=> "outsideLayers"
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

dbGetRegionBoundaryName

```
dbGetRegionBoundaryName(  
    d_regionBoundaryId  
)  
=> t_regionName / nil
```

Description

(Virtuoso MultiTech Framework) Returns region boundary name.

Arguments

<i>d_regionBoundaryId</i>	Database ID of a region boundary.

Value Returned

<i>t_regionName</i>	Name of the region boundary.
<i>nil</i>	An error occurred.

Example

```
name = dbGetRegionBoundaryName( regbId )  
=> "regionBoundary1"
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

dbIsRegionBoundary

```
dbIsRegionBoundary(  
    d_areaBoundaryId  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework) 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
```

Related Topics

[License Requirements of Virtuoso RF Solution](#)

dbMapPinGroupPins

```
dbMapPinGroupPins(  
    d_cellView  
    t_pinMapLayer  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework Only) Maps all the pins in every pin group of the given cellview to the `pinMapLayer` parameter of a die.

ArgumentsV

<i>d_cellView</i>	The cellview in which pins are matched.
<i>t_pinMapLayer</i>	The <code>pinMapLayer</code> parameter of a die.

Value Returned

<i>t</i>	The mapping is done successfully.
<i>nil</i>	The mapping could not be done.

Example

In the following example, a footprint cell is modified by creating `placeBounds`, die pin groups, and `padStacks`. Subsequently, the `padStacks` are added to the die pin group.

```
dieFootprint = (dbOpenCellViewByType "testLib" "footprint" "base" "maskLayout"  
"w")  
  
placeBound1 = (dbCreatePlaceBound dieFootprint "left" list(0:0 100:100) 0 100)  
placeBound2 = (dbCreatePlaceBound dieFootprint "right" list(100:0 200:100) 50 100)  
  
pinGroupLeftTop = (dbCreatePinGroup dieFootprint "left" "top")  
pinGroupLeftBottom = (dbCreatePinGroup dieFootprint "left" "bottom")  
pinGroupRightTop = (dbCreatePinGroup dieFootprint "right" "top")  
pinGroupRightBottom = (dbCreatePinGroup dieFootprint "right" "bottom")
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

```
pinInstLeftTop1 = dbCreateParamInst(dieFootprint pinSuper "PILT1" 20:20 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstLeftTop2 = dbCreateParamInst(dieFootprint pinSuper "PILT2" 20:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightTop1 = dbCreateParamInst(dieFootprint pinSuper "PIRT1" 70:20 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightTop2 = dbCreateParamInst(dieFootprint pinSuper "PIRT2" 70:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstLeftBottom1 = dbCreateParamInst(dieFootprint pinSuper "PILB1" 20:20 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstLeftBottom2 = dbCreateParamInst(dieFootprint pinSuper "PILB2" 20:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightBottom1 = dbCreateParamInst(dieFootprint pinSuper "PIRB1" 20:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )
pinInstRightBottom2 = dbCreateParamInst(dieFootprint pinSuper "PIRB2" 70:70 "R0" 1
list(list("footprintLibNames" "string" "testLib") list("footprintCellName"
"string" "pinFootprint_rect") list("footprintViewName" "string" "base") ) )

dbAddPinToPinGroup(pinGroupLeftTop pinInstLeftTop1)
dbAddPinToPinGroup(pinGroupLeftTop pinInstLeftTop2)
dbAddPinToPinGroup(pinGroupLeftBottom pinInstLeftBottom1)
dbAddPinToPinGroup(pinGroupLeftBottom pinInstLeftBottom2)
dbAddPinToPinGroup(pinGroupRightTop pinInstRightTop1)
dbAddPinToPinGroup(pinGroupRightTop pinInstRightTop2)
dbAddPinToPinGroup(pinGroupRightBottom pinInstRightBottom1)
dbAddPinToPinGroup(pinGroupRightBottom pinInstRightBottom2)

dbMapPinGroupPins(dieFootprint "left:bottom M1,left:top M2,right:bottom
M3,right:top M4")
```

Related Topics

License Requirements of Virtuoso RF Solution

Virtuoso RF Solution 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 are mapped to a `funcDefPin` objects.

pkxAddFuncDefPin

```
pkxAddFuncDefPin(  
    pkx_compDefPinId  
    pkx_funcDefPinId  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework) Adds the specified `funcDefPin` object to the specified `compDefPin` object.

Arguments

<i>pkx_compDefPinId</i>	ID of a <code>compDefPin</code> object.
<i>pkx_funcDefPinId</i>	ID of a <code>funcDefPin</code> object.

Value Returned

<code>t</code>	The specified <code>funcDefPin</code> is added to <code>compDefPin</code> .
<code>nil</code>	The operation failed.

Example

Adds `funcDefPin` to `compDefPin`.

```
pkxAddFuncDefPin(compDefPin funcDefPin)
```

Related Topics

[Views in the Unified Library](#)

pkxCreateCompDef

```
pkxCreateCompDef(  
    d_bagId  
    t_deviceType  
    t_class  
    t_package  
)  
=> pkx_compDefId | nil
```

Description

(Virtuoso MultiTech Framework) Creates a `compDef` object in the specified database and returns its ID.

Arguments

<i>d_bagId</i>	ID of the database containing the object.
<i>t_deviceType</i>	Name of the device type that identifies the <code>compDef</code> object. For example, <code>dtype</code> .
<i>t_class</i>	Placement class of the <code>compDef</code> object to be created. For example, <code>IO</code> .
<i>t_package</i>	Name of the matching package symbol used by the <code>compDef</code> object.

Value Returned

<i>pkx_compDefId</i>	ID of the created <code>compDef</code> object.
<i>nil</i>	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")
```

Related Topics

Components of a Package

pkxCreateCompDefPin

```
pkxCreateCompDefPin(  
    pkx_compDefId  
    t_number  
)  
=> pkx_compDefPinId | nil
```

Description

(Virtuoso MultiTech Framework) Creates a `compDefPin` object and returns its ID.

Arguments

<i>pkx_compDefId</i>	ID of a <code>compDef</code> object.
<i>t_number</i>	Name of the pin, represented by a combination of a string and a number. For example, <code>I10</code> .

Value Returned

<i>pkx_compDefPinId</i>	ID of the created <code>compDefPin</code> object.
<i>nil</i>	The operation failed.

Example

Creates `compDefPin` on the `compDef` object with the pin number as `I1`.

```
compDefPin = pkxCreateCompDefPin(compDef "I1")
```

Related Topics

[Views in the Unified Library](#)

pkxCreateFuncDef

```
pkxCreateFuncDef(  
    pkx_compDefId  
    t_slot  
    t_type  
)  
=> pkx_funcDefId / nil
```

Description

(Virtuoso MultiTech Framework) Creates a `funcDef` object in the specified `compDef` and returns its ID.

Arguments

<i>pkx_compDefId</i>	ID of a <code>compDef</code> object.
<i>t_slot</i>	Name of the function slot. For example, G1.
<i>t_type</i>	Type of function. For example, A.

Value Returned

<i>pkx_funcDefId</i>	ID of the created <code>funcDef</code> object.
<i>nil</i>	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")
```

Related Topics

[Views in the Unified Library](#)

pkxCreateFuncDefPin

```
pkxCreateFuncDefPin(  
    pkx_funcDefId  
    t_number  
    t_use  
    x_swapCode  
)  
=> pkx_funcDefPinId / nil
```

Description

(Virtuoso MultiTech Framework) Creates a `funcDefPin` object in the specified `funcDef` object and returns its ID.

Arguments

<i>pkx_funcDefId</i>	ID of a <code>funcDef</code> object.
<i>t_number</i>	Name of the pin, represented by a combination of a string and a number. For example, I10.
<i>t_use</i>	Value indicating the logical use of the <code>funcDefPin</code> object. Valid values: IN, OUT, BI, TRI, OCA, OCL, POWER, GROUND, NC, and UNSPEC.
<i>x_swapCode</i>	Code indicating whether the <code>funcDefPin</code> object can be swapped with another pin.

Value Returned

<i>pkx_funcDefPinId</i>	ID of the created <code>funcDefPin</code> object.
<i>nil</i>	The <code>funcDefPin</code> object was not created.

Example

Creates the `funcDefPin` object in the `funcDef` object and returns its ID.

```
funcDefPin = pkxCreateFuncDefPin(funcDef "A" "IN" 1)
```


Related Topics

[Views in the Unified Library](#)

pkxDeleteCompDef

```
pkxDeleteCompDef(  
    pkx_compDefId  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework) Deletes the specified `compDef` object.

Arguments

<i>pkx_compDefId</i>	ID of a <code>compDef</code> object.

Value Returned

<code>t</code>	The specified <code>compDef</code> object was deleted.
<code>nil</code>	The specified <code>compDef</code> object was not deleted.

Example

The `compDefId` object is deleted.

```
pkxDeleteCompDef(compDefId)
```

Related Topics

[Views in the Unified Library](#)

pkxFindCompDef

```
pkxFindCompDef (
    d_bagId
    t_deviceType
)
=> pkx_compDefId / nil
```

Description

(Virtuoso MultiTech Framework) Searches for a `compDef` object in the specified database with the specified device type and returns its ID if it exists.

Arguments

<i>d_bagId</i>	ID of the database containing the object.
<i>t_deviceType</i>	Type of device that identifies the component definition. For example, <code>dtype</code> .

Value Returned

<i>pkx_compDefId</i>	ID of the <code>compDef</code> object with the specified attributes.
<i>nil</i>	The specified <code>compDef</code> object was not found.

Example

Searches for the `compDef` object in the database `bag`, with the device type `dtype`.

```
compDef = pkxFindCompDef(bag "dtype")
```

Related Topics

[Views in the Unified Library](#)

pkxFindCompDefPin

```
pkxFindCompDefPin(  
    pkx_compDefId  
    t_number  
)  
=> pkx_compDefPinId | nil
```

Description

(Virtuoso MultiTech Framework) Searches for a `compDefPin` object in the specified `compDef` with the specified attributes, and returns its ID if it exists.

Arguments

<i>pkx_compDefId</i>	ID of a <code>compDef</code> object.
<i>t_number</i>	Name of the pin, represented by a combination of a string and a number. For example, <code>I10</code> .

Value Returned

<i>pkx_compDefPinId</i>	ID of the <code>compDefPin</code> object in the specified <code>compDef</code> .
<i>nil</i>	The specified <code>compDefPin</code> object does not exist in the specified <code>compDef</code> object or the operation failed.

Example

Searches for the `compDef` object with the pin number `I1`.

```
pkxFindCompDefPin(compDef "I1")
```

Related Topics

[Views in the Unified Library](#)

pkxFindFuncDef

```
pkxFindFuncDef(  
    pkx_compDefId  
    t_slot  
    t_type  
)  
=> pkx_funcDefId / nil
```

Description

(Virtuoso MultiTech Framework) Searches for a `funcDef` object in the specified `compDef` with the specified attributes, and returns its ID if it exists.

Arguments

<i>pkx_compDefId</i>	ID of a <code>compDef</code> object.
<i>t_slot</i>	Name of the function slot. For example, G1.
<i>t_type</i>	Type of function. For example, A.

Value Returned

<i>pkx_funcDefId</i>	ID of the <code>funcDef</code> object in the specified <code>compDef</code> object.
<i>nil</i>	The specified <code>funcDef</code> object does not exist in the specified <code>compDef</code> 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")
```

Related Topics

[Views in the Unified Library](#)

pkxFindFuncDefPin

```
pkxFindFuncDefPin(  
    pkx_funcDefId | pkx_compDefPinId  
    t_name  
    t_use  
    s_swapCode  
)  
=> pkx_funcDefPinId | nil
```

Description

(Virtuoso MultiTech Framework) 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

<i>pkx_funcDefId pkx_compDefPinId</i>	ID of a <code>funcDef</code> or <code>compDefPin</code> .
<i>t_name</i>	Name of the <code>funcDefPin</code> object.
<i>t_use</i>	Value indicating the logical use of the <code>funcDefPin</code> object. Valid values: IN, OUT, BI, TRI, OCA, OCL, POWER, GROUND, NC, and UNSPEC.
<i>s_swapCode</i>	Code indicating whether the <code>funcDefPin</code> object can be swapped with another pin.

Value Returned

<i>pkx_funcDefPinId</i>	ID of the <code>funcDefPin</code> pin object that was found in the <code>funcDef</code> or <code>compDefPin</code> object.
<i>nil</i>	The <code>funcDefPin</code> 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)
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

Searches for the `compDefPin` object `pin3` with name `A`, use `IN`, and swapcode `1` in the `funcDef` object.

```
pin3 = pkxFindFuncDefPin(compDefPin "A" "IN" 1)
```

Related Topics

[Views in the Unified Library](#)

pkxGetCompDefPins

```
pkxGetCompDefPins(  
    d_bagId  
)  
=> list(pkx_compDefPinId) | nil
```

Description

(Virtuoso MultiTech Framework) Returns a list of `compDefPin` object IDs in the specified database.

Arguments

<code>d_bagId</code>	ID of the database containing the <code>compDefPin</code> objects.

Value Returned

<code>list(pkx_compDefPinId)</code>	List of <code>compDefPin</code> objects.
<code>nil</code>	Cannot find any <code>compDefPin</code> objects or the operation failed.

Example

Returns a list of `compDefPin` objects in the database `bag`.

```
pins = pkxGetCompDefPins(bag)
```

Related Topics

[Views in the Unified Library](#)

pkxGetCompDefs

```
pkxGetCompDefs (  
    d_bagId  
)  
=> list(pkx_compDefId) / nil
```

Description

(Virtuoso MultiTech Framework) Returns a list of `compDef` object IDs in the specified database.

Arguments

<code>d_bagId</code>	ID of the database containing the <code>compDef</code> objects.

Value Returned

<code>list(pkx_compDefId)</code>	List of <code>compDef</code> IDs in the specified database.
<code>nil</code>	Cannot find any <code>compDef</code> objects or the operation failed.

Example

Returns a list of `compDef` objects in the database `bag`.

```
pkxGetCompDefs (bag)
```

Related Topics

[Views in the Unified Library](#)

pkxGetFuncDefPins

```
pkxGetFuncDefPins(  
    d_bagId  
)  
=> list(pkx_funcDefPinId) | nil
```

Description

(Virtuoso MultiTech Framework) Returns a list of `funcDefPin` object IDs in the specified database.

Arguments

<code>d_bagId</code>	ID of the database containing the objects.

Value Returned

<code>list(pkx_funcDefPinId)</code>	List of <code>funcDefPinIds</code> found in the specified database.
<code>nil</code>	Cannot find any <code>funcDefPin</code> objects or the operation failed.

Example

Returns a list of `funcDefPin` objects in the database `bag`.

```
pkxGetFuncDefPins(bag)
```

Related Topics

[Views in the Unified Library](#)

pkxGetFuncDefs

```
pkxGetFuncDefs (
    d_bagId
)
=> list(pkx_funcDefId) / nil
```

Description

(Virtuoso MultiTech Framework) Returns a list of `funcDef` object IDs in the specified database.

Arguments

<code>d_bagId</code>	ID of the database containing the <code>funcDef</code> object.

Value Returned

<code>list(pkx_funcDefId)</code>	List of <code>funcDef</code> objects found in the specified database.
<code>nil</code>	Cannot find any <code>funcDef</code> objects or the operation failed.

Example

Returns a list of `funcDef` objects in the database `bag`.

```
pkxGetFuncDefs(bag)
```

Related Topics

[Views in the Unified Library](#)

pkxGetProp

```
pkxGetProp(  
    pkx_Obj  
    t_propName  
)  
=> pkx_Prop / nil
```

Description

(Virtuoso MultiTech Framework) Returns the property of the specified package object and name.

Arguments

<i>pkx_Obj</i>	ID of the specified package object.
<i>t_propName</i>	Name of the property.

Value Returned

<i>pkx_Prop</i>	Property of the specified package object.
<i>nil</i>	The operation failed.

Example

Retrieves the property for the package object `compDef` with the property name `propList`.

```
prop = pkxGetProp(compDefPin "propList")
```

Related Topics

[Views in the Unified Library](#)

pkxGetProps

```
pkxGetProps (  
    pkx_Obj  
    t_propName  
)  
=> list(pkx_properties) / nil
```

Description

(Virtuoso MultiTech Framework) Returns a list of properties for the specified package object.

Arguments

<i>pkx_Obj</i>	ID of a package object.
<i>t_propName</i>	Name of the property.

Value Returned

<i>list(pkx_properties)</i>	List of properties.
<i>nil</i>	The operation failed.

Example

Retrieves the list of properties for the package object `compDef`.

```
compDefProps = pkxGetProps(compDef)
```

Related Topics

[Views in the Unified Library](#)

pkxSetProp

```
pkxSetProp(  
    pkx_Obj  
    t_propName  
    S_propType  
    g_propValue  
)  
=> pkx_property / nil
```

Description

(Virtuoso MultiTech Framework) Sets the specified properties on the specified package object.

Arguments

<i>pkx_Obj</i>	ID of a package object.
<i>t_propName</i>	Name of the property.
<i>S_propType</i>	Type of the property.
<i>g_propValue</i>	Value of the property.

Value Returned

<i>pkx_prop</i>	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")
```

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

Related Topics

[Views in the Unified Library](#)

pkxPadFile

```
pkxPadFile(  
    t_fileName  
    t_newFileName  
    g_magicBytes  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework Only) Pads the specified SiP file with so-called magic bytes up front. Magic bytes are a set of bytes that can be added to a SiP file to encrypt it. The import assisted flow takes a SiP file as an input to generate a Virtuoso RF Solution 5x hierarchy. In the process, the original SiP file can be modified incrementally in the assisted export flow. However, to ensure that the original SiP file can be retrieved, it is padded before any modification is done.

Arguments

<i>t_fileName</i>	Name of the SiP file to be padded.
<i>t_newFileName</i>	New file name after padding.
<i>g_magicBytes</i>	A set of bytes to be added to the SiP file.

Value Returned

<i>t</i>	The specified file was padded.
<i>nil</i>	The operation failed.

Example

```
pkxPadFile("RF_pkg_pa_sim.sip" "new.sip" "Magic Bytes")
```

Related Topics

[pkxUnPadFile](#)

pkxUnPadFile

```
pkxUnPadFile(  
    t_fileName  
    t_newFileName  
    g_magicBytes  
)  
=> t / nil
```

Description

(Virtuoso MultiTech Framework Only) Removes the padding from the specified SiP file using magic bytes.

Arguments

<i>t_fileName</i>	Name of the padded file.
<i>t_newFileName</i>	New file name after removing the padding.
<i>g_magicBytes</i>	A set of bytes that is added to a SiP file.

Value Returned

<i>t</i>	The padding was removed from the specified SiP file.
<i>nil</i>	The operation failed.

Example

```
pkxUnPadFile("newFile.log" "theFile.log" "Magic Bytes")
```

Related Topics

[pkxPadFile](#)

Virtuoso Studio Design Environment SKILL Reference

Virtuoso MultiTech Framework Functions (Virtuoso MultiTech Framework)

Photonics Functions (Virtuoso Photonics Option)

This topic provides a list of Cadence® SKILL functions associated with the Virtuoso Photonics flow. You need the `Virtuoso_Photonics_Option` license with `Virtuoso_MultiTech_Framework` (95022) to use these SKILL functions.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[cclsEnabled](#)

[Curvy Core Mathematical Object Creation Functions](#)

[Curvy Core Shape Creation Functions](#)

[Curvy Core Shape Geometry Functions](#)

[Curvy Core Mathematical Object Functions](#)

[Curvy Core Mathematical Object Attribute Query Functions](#)

[Curvy Core Shape Object Attribute Query Functions](#)

[Curvy Core Transformations Functions](#)

[Curvy Core Connector Functions](#)

[Curvy Core Optical Rule Engine Functions](#)

[Photonics Database Functions](#)

[Photonics Flow Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

[Photonics Symbol Generator Functions](#)

[License Checkout for Virtuoso Photonics Platform](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

ccIsEnabled

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

Description

(Virtuoso Photonics Option) 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.
```

Curvy Core Mathematical Object Creation Functions

The curvy core engine supports some curvy core mathematical objects such as curves, facets, line segments, and poly curves. You can use SKILL functions to create these mathematical objects.

Related Topics

[ccCreateCurve](#)

[ccCreateFacet](#)

[ccCreateLineSegment](#)

[ccCreatePolyCurve](#)

ccCreateCurve

```
ccCreateCurve (  
    t_x  
    t_y  
    l_interval  
)  
=> o_curve / nil
```

Description

(Virtuoso Photonics Option) 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

<i>t_x</i>	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.
<i>t_y</i>	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.
<i>l_interval</i>	<code>list(n_begin:n_end)</code> An interval for the input variable determined by its start <i>n_begin</i> and end <i>n_end</i> .

Value Returned

<i>o_curve</i>	ID of the resulting ccCurve.
<i>nil</i>	An error occurred.

Examples

```
ccCreateCurve("t" "1" 0:1)
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
=> 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
```

Related Topics

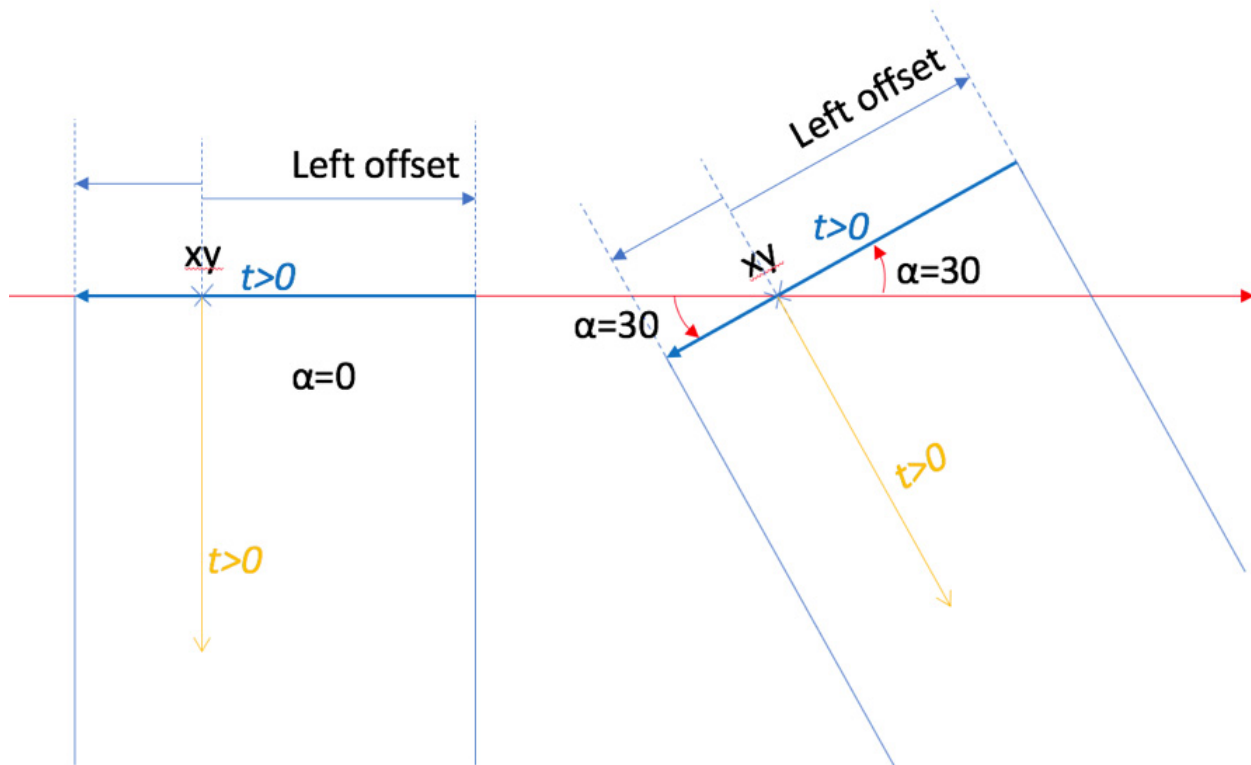
[Curvy Core Mathematical Object Creation Functions](#)

ccCreateFacet

```
ccCreateFacet(  
    l_origin  
    [ ?angle n_angle ]  
    [ ?radius n_radius ]  
    [ ?offsets l_offsets ]  
    [ ?name t_name ]  
)  
=> o_facet / nil
```

Description

(Virtuoso Photonics Option) 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.



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Arguments

<code>l_origin</code>	<code>list(n_x n_y)</code> Origin points of the facet specified by <code>n_x</code> and <code>n_y</code> on the X-axis and Y-axis, respectively.
<code>?angle n_angle</code>	Angle between the facet and the positive X-axis. The default is 0.
<code>?radius n_radius</code>	Curvature radius of the origin point. The default is 0.
<code>?offsets l_offsets</code>	<code>list(n_lhs n_rhs)</code> Offsets from the origin point to the endpoints of the facet. The default is (0:0).
<code>?name t_name</code>	Name of the facet. The default is " ".

Value Returned

<code>o_facet</code>	ID of the resulting ccFacet.
<code>nil</code>	An error occurred.

Example

```
ccCreateFacet(-2:0 ?offsets 2:2 ?angle 90)
> cc@0x269c83e0
```

Related Topics

[Curvy Core Mathematical Object Creation Functions](#)

ccCreateLineSegment

```
ccCreateLineSegment(  
    l_pt1  
    l_pt2  
    ) => o_curve / nil
```

Description

(Virtuoso Photonics Option) 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

<i>l_pt1</i>	<code>list(l_x l_y)</code> The start point of the line segment.
<i>l_pt2</i>	<code>list(l_x l_y)</code> The end point of the line segment.

Value Returned

<i>o_curve</i>	ID of the resulting ccCurve.
<i>nil</i>	An error occurred.

Example

```
ccCreateLineSegment(-1:-1 1:1)  
> cc@0x269c83f0
```

Related Topics

[Curvy Core Mathematical Object Creation Functions](#)

ccCreatePolyCurve

```
ccCreatePolyCurve(  
    l_segments  
    [ ?close { t | nil } ]  
    [ ?smoothness S_smoothness ]  
)  
=> o_polyCurve / nil
```

Description

(Virtuoso Photonics Option) 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*.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?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).
```

Related Topics

[Curvy Core Mathematical Object Creation Functions](#)

ccGetFacetCorners

```
ccGetFacetCorners(  
    o_shape  
    g_facet  
)  
=> l_corners / nil
```

Description

Retrieves the corner points of the facet snapped to the manufacturing grid of the shape containing the facet.

Arguments

<i>o_shape</i>	ID of a ccShape object.
<i>g_facet</i>	<i>o_facet</i> <i>t_facetName</i> where: ■ <i>o_facet</i> is ID of a ccFacet object. ■ <i>t_facetName</i> is the name of the facet.

Examples

```
c = ccCreateCurve("20*cos(t)" "10*sin(t)" -pi/4:pi/3)  
=> cc@0xb3f1648e  
path = ccCreatePath(cv "waveguide" list(c) ?defaultWidth 1)  
=> cc@0x343b3b00  
ccGetFacetCorners(path path->beginFacet)  
=> ((13.95 -6.6) (14.35 -7.5))  
ccGetFacetCorners(path "begin")  
=> ((13.95 -6.6) (14.35 -7.5))  
)
```

ccGetFacetOrigin

```
ccGetFacetOrigin(  
    o_shape  
    g_facet  
)  
=> l_origin / nil
```

Description

Retrieves the origin point of the facet snapped to the manufacturing grid of the shape containing the facet.

Arguments

<i>o_shape</i>	ID of a ccShape object.
<i>g_facet</i>	<i>o_facet</i> <i>t_facetName</i> <i>l_facetOrigin</i>

where:

- *o_facet* is ID of a ccFacet object.
- *t_facetName* is the name of the facet.
- *l_facetOrigin* is the origin point of the facet represented by (*n_x* *n_y*).

Value Returned

<i>l_origin</i>	The snapped origin point of the facet.
nil	An error occurred.

Examples

```
curve = ccCreateCurve("20*cos(t)" "10*sin(t)" -pi/4:pi/3)  
> cc@0xb3f1648e  
path = ccCreatePath(cv "waveguide" list(curve) ?defaultWidth 1)  
> cc@0x343b3b00  
ccGetFacetOrigin(path path->beginFacet)  
> (14.15 -7.05)  
ccGetFacetOrigin(path "begin")  
> (14.15 -7.05)
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
ccGetFacetOrigin(path '(14.14214 -7.071068))  
> (14.15 -7.05)
```


Curvy Core Shape Creation Functions

The curvy core engine supports several curvy core shapes such as ellipses, lines, paths, pies, polygons, rectangles, and surfaces. You can use SKILL functions to create these curvy core shapes.

Related Topics

[ccCreateEllipse](#)

[ccCreateLine](#)

[ccCreatePath](#)

[ccCreatePie](#)

[ccCreatePolygon](#)

[ccCreateRect](#)

[ccCreateSurface](#)

[ccCreateSurfaceFromPath](#)

ccCreateEllipse

```
ccCreateEllipse(  
    d_cellViewId  
    l_lpp  
    l_llPoint  
    l_urPoint  
    ?offset l_offset  
    ?rotation n_rotation  
    ?genFigs g_genFigs  
)  
=> o_surface / nil
```

Description

(Virtuoso Photonics Option) Creates an ellipse with the specified coordinates as a ccSurface object in the cellview with specified layer-purpose pair. The ellipse is then transformed according to the specified offset and rotation. Optionally, OpenAccess figures (polygons) are created.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>l_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) A layer or a layer-purpose pair.
<i>l_llPoint</i>	<i>list(n_x n_y)</i> Lower-left point of the rectangle enclosing the ellipse.
<i>l_urPoint</i>	<i>list(n_x n_y)</i> Upper-right point of the rectangle enclosing the ellipse.
<i>?offset l_offset</i>	<i>list(n_lhs n_rhs)</i> Offset value for the rectangle transformation. The default is (0,0).
<i>?rotation n_rotation</i>	Rotation angle for the ellipse transformation. The default is (0,0).

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?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.

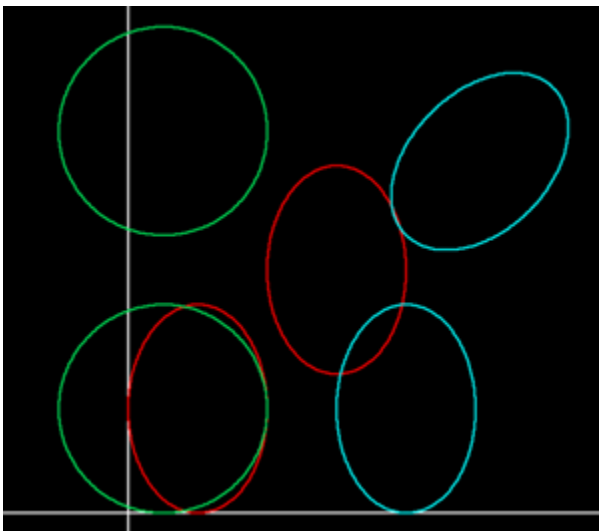
Example

```
cv = geGetEditCellView()

; the red ellipses
ccCreateEllipse( cv list("y1" "drawing") 0:0 10:15 ?offset 10:10)
ccCreateEllipse( cv list("y1" "drawing") 0:0 10:15 )

; the green ellipses (circles)
ccCreateEllipse( cv list("y2" "drawing") -5:0 10:15 )
ccCreateEllipse( cv list("y2" "drawing") -5:0 10:15 ?offset 0:20)

; the light blue ellipses
ccCreateEllipse( cv list("y4" "drawing") -5:0 5:15 ?offset 20:0)
ccCreateEllipse( cv list("y4" "drawing") -5:0 5:15 ?offset 20:20 ?rotation -45)
```



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Curvy Core Shape Creation Functions](#)

ccCreateLine

```
ccCreateLine(  
    d_cellViewId  
    l_lpp  
    g_polyCurveSegments  
)  
=> o_line / nil
```

Description

(Virtuoso Photonics Option) 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

<i>d_cellViewId</i>	Database ID of cellview.
<i>l_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) A layer or a layer-purpose pair.
<i>g_polyCurveSegmen</i>	<i>o_polyCurve</i> (<i>l_polyCurveSegment...</i>) ID of an originating ccPolyCurve (<i>o_polyCurve</i>) or a list of polyCurve segments (<i>l_polyCurveSegment</i>) as specified in <u>ccCreatePolyCurve</u> .

Value Returned

<i>o_line</i>	ID of the resulting ccLine.
<i>nil</i>	An error occurred.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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).
```

Related Topics

[ccCreatePolyCurve](#)

[Curvy Core Shape Creation Functions](#)

ccCreatePath

```
ccCreatePath(  
    d_cellviewId  
    l_lpp  
    l_segments  
    [?defaultWidth g_defWidth]  
    [?smoothness S_smoothness]  
)  
=> o_path / nil
```

Description

(Virtuoso Photonics Option) Creates a ccPath object on the specified layer-purpose pair from one or a list of path segments.

Arguments

<i>d_cellviewId</i>	Database ID of a cellview.
<i>l_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) A layer or a layer-purpose pair.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?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.
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Curvy Core Shape Creation Functions](#)

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

(Virtuoso Photonics Option) 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.

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, capping 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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Arguments

<code>d_cellViewId</code>	Database ID of a cellview.
<code>l_lpp</code>	<code>tx_layer (tx_layer tx_purpose)</code> A layer or a layer-purpose pair.
<code>?angle n_angle</code>	Angle between the starting radius and the positive X-axis. The ending radius will have the angle <code>n_angle</code> 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.
<code>?startRadius n_startRadius</code>	Radius of the inner circle. The default is <code>nil</code> , which means there is no inner circle.
<code>?length n_length</code>	Difference between the radiuses of the outer circle and the inner circle representing donut thickness, or a circle radius when <code>g_startRadius</code> is <code>nil</code> .
<code>?endWidth n_endWidth</code>	Width of the capping rectangle. The default is <code>nil</code> , which means that the width is not capped.
<code>?endLength n_endLength</code>	Length of the capping rectangle. The default is <code>nil</code> , which means length is not capped.
<code>?offset l_offset</code>	<code>list(n_lhs n_rhs)</code> Offset of the resulting figure. The default is <code>(0,0)</code> .
<code>?rotation n_rotation</code>	Rotation of the resulting figure. The default is <code>(0,0)</code> .

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?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 occurred.

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.
```

Related Topics

[Curvy Core Shape Creation Functions](#)

ccCreatePolygon

```
ccCreatePolygon(  
    d_cellview  
    l_lpp  
    l_points  
    ?offset l_offset  
    ?rotation n_rotation  
    ?genFigs g_genFigs  
    ?tolerance n_tolerance  
)  
=> o_surface | nil
```

Description

(Virtuoso Photonics Option) 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

<i>d_cellviewId</i>	Database ID of a cellview.
<i>l_lpp</i>	A layer or a layer-purpose pair. <i>tx_layer</i> (<i>tx_layer tx_purpose</i>)
<i>l_points</i>	List of points. <i>list(n_x n_y)</i> Note: If the list of points do not close the polygon, it is closed automatically.
<i>?offset l_offset</i>	Offset value for the surface. <i>list(n_lhs n_rhs)</i> The default is (0,0).
<i>?rotation n_rotation</i>	Rotation angle for the surface. The default is (0,0).

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?genFigs g_genFigs

Specifies whether OpenAccess figures are generated.

The default is `t`, which means that figures are generated.

?tolerance n_tolerance

Tolerance for the removal of input points. Point will be removed if its distance from the line connecting its two neighboring points is less than `n_tolerance`. The default is 0.

Value Returned

o_surface

ID of the resulting `ccSurface`. The polygon is the boundary of the surface.

nil

An error occurred.

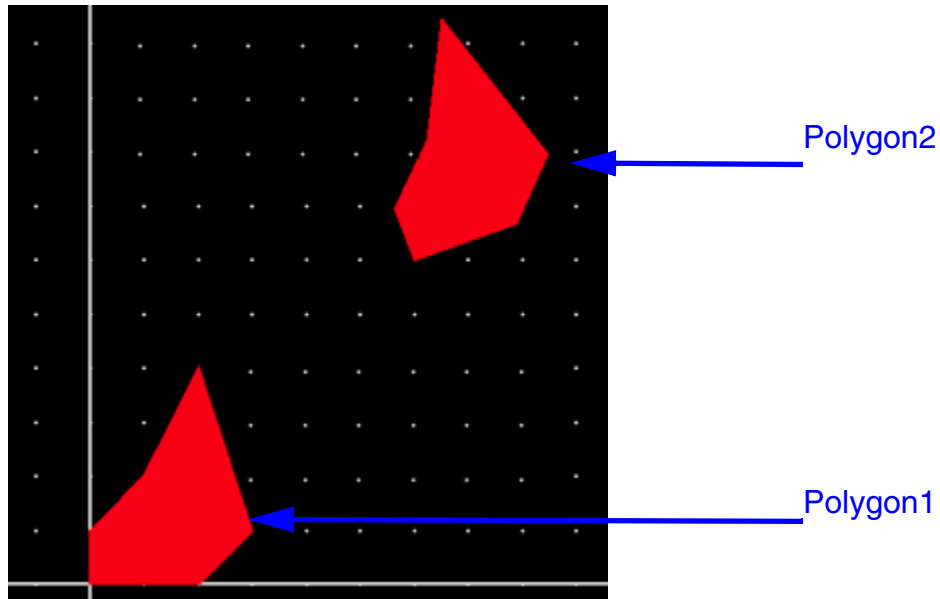
Example

```
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
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Two polygons are created as shown below:



Related Topics

[Curvy Core Shape Creation Functions](#)

ccCreateRect

```
ccCreateRect(  
    d_cellview  
    l_lpp  
    l_llPoint  
    l_urPoint  
    ?offset l_offset  
    ?rotation n_rotation  
    ?genFigs g_genFigs  
)  
=> o_surface | nil
```

Description

(Virtuoso Photonics Option) 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

<code>d_cellViewId</code>	Database ID of a cellview.
<code>l_lpp</code>	A layer or a layer-purpose pair. <code>tx_layer</code> (<code>tx_layer tx_purpose</code>)
<code>l_llPoint</code>	Lower-left point of the rectangle. <code>list(n_x n_y)</code>
<code>l_urPoint</code>	Upper-right point of the rectangle. <code>list(n_x n_y)</code>
<code>?offset l_offset</code>	Offset value for the rectangle transformation. <code>list(n_lhs n_rhs)</code> The default is (0,0).
<code>?rotation n_rotation</code>	Rotation angle for the rectangle transformation. The default is (0,0).

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?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
```

Related Topics

[Curvy Core Shape Creation Functions](#)

ccCreateSurface

```
ccCreateSurface(  
    d_cellViewId  
    l_lpp  
    o_boundary  
    [ l_holes ]  
)  
=> o_surface / nil
```

Description

(Virtuoso Photonics Option) 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

<i>d_cellViewId</i>	Database ID of a cellview.
<i>l_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) A layer or a layer-purpose pair.
<i>o_boundary</i>	A ccPolyCurve object with positive area.
<i>l_holes</i>	(<i>o_holes...</i>) A list of ccPolyCurves with negative areas.

Value Returned

<i>o_surface</i>	ID of the resulting ccSurface.
<i>nil</i>	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
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Curvy Core Shape Creation Functions](#)

ccCreateSurfaceFromPath

```
ccCreateSurfaceFromPath(  
    o_path  
    [ ?facetNames l_facetNames ]  
)  
=> o_surface | nil
```

Description

(Virtuoso Photonics Option) 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

<i>o_path</i>	ID of a ccPath object.
<i>?facetNames l_facetNames</i>	<pre>list(t_beginFacet t_endFacet) nil</pre> List of new names for begin (<i>t_beginFacet</i>) and end facets (<i>t_endFacet</i>) of the ccPath object.

Value Returned

<i>o_surface</i>	ID of the resulting ccSurface
<i>nil</i>	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"
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
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"
```

Related Topics

Curvy Core Shape Creation Functions

Curvy Core Shape Geometry Functions

You can use the curvy core shape geometry functions to copy specified curvy core shape objects to the specified cellview and layer-purpose pair (LPP), associate a database figure with a new curvy core shape, set offsets, remove database figures, and perform other curvy core shape geometry related operations.

Related Topics

[ccCopyFig](#)

[ccConvertIntoCurvy](#)

[ccGenFigs](#)

[ccGenOffsetFig](#)

[ccGetFromFig](#)

[ccMoveFig](#)

[ccRemoveFigs](#)

ccCopyFig

```
ccCopyFig(  
    o_shape  
    [ ?cv d_cellViewID ]  
    [ ?lpp g_lpp ]  
    [ ?transform l_transform ]  
)  
=> o_shape2 / nil
```

Description

(Virtuoso Photonics Option) 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

<i>o_shape</i>	ID of a ccShape object.
<i>?cv d_cellViewId</i>	ID of a cellview.
<i>?lpp g_lpp</i>	A layer or a layer-purpose pair in the specified cellview in the following format: <i>tx_layer</i> (<i>tx_layer tx_purpose</i>)
<i>?transform l_transform</i>	A transformation object in the following format: (<i>(n_offsetX n_offsetY) n_angle</i>) where: ■ <i>n_angle</i> is the rotation angle. ■ <i>n_offsetX</i> and <i>n_offsetY</i> determine the offsets of the X-axis and Y-axis in this format (0,0).

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

<code>o_shape2</code>	ID of the destination ccShape.
<code>nil</code>	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)
```

Related Topics

Curvy Core Shape Geometry Functions

ccConvertIntoCurvy

```
ccConvertIntoCurvy(  
    d_fig  
)  
=> o_owner / nil
```

Description

(Virtuoso Photonics Option) Returns a new ccShape to which the specified database figure is attached.

Note: This function supports rectangles and truncated pathSegs.

Argument

<i>d_fig</i>	Database ID of a figure.

Value Returned

<i>o_owner</i>	ID of the new ccShape to which the specified figure is attached.
<i>nil</i>	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.
```

Related Topics

[Curvy Core Shape Geometry Functions](#)

ccGenFigs

```
ccGenFigs (
    o_shape
    [ ?attach { t | nil } ]
    [ ?snapToMFG { t | nil } ]
    [ ?accuracy { n_accuracy | nil } ]
)
=> l_figs / nil
```

Description

(Virtuoso Photonics Option) 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

<code>o_shape</code>	ID of a ccShape object.
<code>?attach { t nil }</code>	Specifies whether to attach the resulting figures to the originating ccShape. The default value <code>t</code> specifies that the resulting figures are attached to the originating ccShape.
<code>?snapToMFG { t nil }</code>	Specifies whether to snap the output figures to the manufacturing grid. The default value <code>t</code> specifies that the output figures are snapped to the manufacturing grid.
<code>?accuracy { n_accuracy nil }</code>	Discretization accuracy for retrieving the accuracy from the manufacturing grid. The default value <code>nil</code> specifies that using the step of the manufacturing grid as the discretization accuracy value.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

<i>l_figs</i>	List of database figure IDs.
<i>nil</i>	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
```

Related Topics

[Curvy Core Shape Geometry Functions](#)

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

(Virtuoso Photonics Option) 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

<i>o_path</i>	ID of a ccPath object. The edges and facets of this path are used to generate an offset ccShape.
<i>g_lpp</i>	A layer or a layer-purpose pair in the specified cellview in the following format: <i>tx_layer</i> (<i>tx_layer tx_purpose</i>)
<i>?side t_side</i>	Defines the edge and offset direction. Valid values: "left", "right"

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?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

<code>o_shape</code>	ID of the created ccShape.
<code>nil</code>	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))
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Curvy Core Shape Geometry Functions](#)

ccGetFromFig

```
ccGetFromFig(  
    d_fig  
)  
=> o_owner / nil
```

Description

(Virtuoso Photonics Option) 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

<i>d_fig</i>	Database ID of a figure.

Value Returned

<i>o_owner</i>	ID of the ccShape object to which the specified figure is attached.
<i>nil</i>	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
```

Related Topics

[Curvy Core Shape Geometry Functions](#)

ccMoveFig

```
ccMoveFig(  
    o_shape  
    [ ?cv d_cellViewId ]  
    [ ?lpp g_lpp ]  
    [ ?transform l_transform ]  
)  
=> o_shape2 / nil
```

Description

(Virtuoso Photonics Option) 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

<code>o_shape</code>	ID of a ccShape object.
<code>?cv d_cellViewId</code>	ID of a cellview.
<code>?lpp g_lpp</code>	A layer or a layer purpose pair in the specified cellview in the following format <code>tx_layer (tx_layer tx_purpose)</code>
<code>?transform l_transform</code>	

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)`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

<code>o_shape2</code>	ID of the destination ccShape.
<code>nil</code>	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.
```

Related Topics

Curvy Core Shape Geometry Functions

ccRemoveFigs

```
ccRemoveFigs (  
    o_shape  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Removes the database figures attached to the given ccShape.

Argument

<i>o_shape</i>	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
```

Related Topics

[Curvy Core Shape Geometry Functions](#)

Curvy Core Mathematical Object Functions

The curvy core mathematical object functions can be used to set layer offsets, poly curve offsets, and rename facets.

Related Topics

[ccLayerOffset](#)

[ccOffsetPolyCurve](#)

[ccRenameFacet](#)

ccLayerOffset

```
ccLayerOffset(  
    g_shapes  
    n_value  
    [ ?cellView d_cellView ]  
    [ ?lpp g_lpp ]  
    [ ?facetsToCurves { t | nil } ]  
)  
=> l_surfaces
```

Description

(Virtuoso Photonics Option) Returns a list of surfaces derived by merging the areas of the input shapes and offsetting them by the specified offset value. Resulting surfaces can optionally be created in the specified cellview and with the specified layer-purpose pair different from the default layer-purpose pair of the first input shape.

Arguments

<i>g_shapes</i>	<i>o_shape</i> (<i>o_shape</i> ...) An input shape or a list of input shapes.
<i>n_value</i>	Offset value.
<i>?cellView d_cellView</i>	Database ID of a cellview.
<i>?lpp g_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) Layer or a layer-purpose pair.
<i>?facetsToCurves t</i> <i>nil</i>	Specifies whether all input facets are converted to curves before the actual operation. The default is <i>nil</i> , which means that the input facets are not converted to curves.

Virtuoso Studio Design Environment SKILL Reference

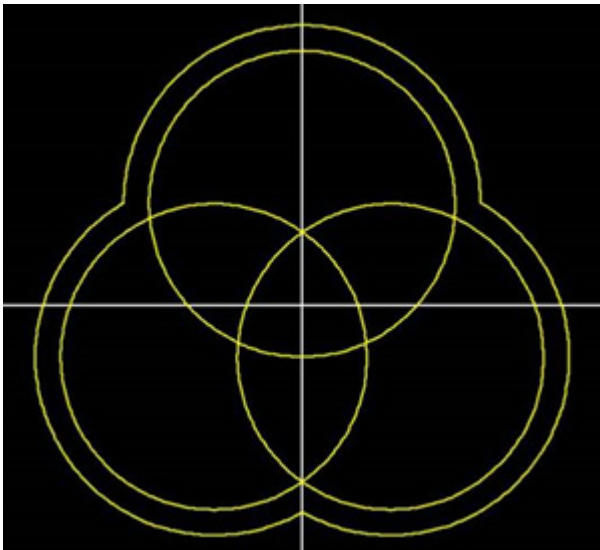
Photonics Functions (Virtuoso Photonics Option)

Value Returned

l_surfaces List of resulting surfaces.

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=ccLayerOffset(list(s1 s2 s3) 0.5)
> (cc@0x2b01d870)
```



Related Topics

[Curvy Core Mathematical Object Functions](#)

ccOffsetPolyCurve

```
ccOffsetPolyCurve(  
    o_polyCurve  
    n_value  
    [ ?key g_key ]  
)  
=> o_polyCurve2
```

Description

(Virtuoso Photonics Option) Returns a copy of the specified polyCurve that is expanded by the specified offset value.

Arguments

<i>o_polyCurve</i>	ID of a ccPolyCurve object.
<i>n_value</i>	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.
<i>?key g_key</i>	A flag that outputs a polyCurve with "raw" offset when set to a value that is not <code>nil</code> . A "raw" offset polyCurve has cusps, which can be useful for debugging. The default is <code>nil</code> .

Value Returned

<i>o_polyCurve2</i>	ID of the resulting ccPolyCurve.
<code>nil</code>	An error occurred.

Example

```
arc = ccCreatePolyCurve(list(list(ccCreateCurve("cos(t)" "sin(t)" 0:3.14159265))))  
> cc@0xc70cee6c  
  
arc2 = ccOffsetPolyCurve(arc 1)  
> cc@0x86ca13ca  
; A circle arc of radius 1 gets expanded to an arc of radius 2.  
arc2~>beginPt
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

> (2.0 0.0)

arc2~>endPt

> (-2.0 0.0)

Related Topics

Curvy Core Mathematical Object Functions

ccOrientCurve

```
ccOrientCurve(  
    o_curve  
    S_orient  
)  
=> o_outCurve | nil
```

Description

(Virtuoso Photonics Option) Returns a copy of the specified `ccCurve` or `ccPolyCurve` rotated counterclockwise around the origin by a multiple of 90 degrees after optionally mirroring it across the X-axis or Y-axis. Mirroring also reverses the orientation of the curve.

Arguments

<code>o_curve</code>	ID of a <code>ccCurve</code> or <code>ccPolyCurve</code> object.
<code>S_orient</code>	Transformation specified by rotation angle around (0,0) and optional mirroring across the X-axis or Y-axis. Valid values are "R0", "R90", "R180", "R270", "MY", "MYR90", "MX", "MXR90"

Examples

In this example a straight line curve with equation $x(t)=2*t$, $y(t)=3*t$ is mirrored and then rotated by 90 degrees counterclockwise around the origin, resulting in a straight line with equation $x(t)=-2*t$, $y(t)=3*t$ and transformation angle of 90 degrees.

```
pc = ccCreatePolyCurve(ccCreateCurve("2*t" "3*t" 0:1))  
> cc@0x68c6b149  
tpc = ccOrientCurve(pc "MYR90")  
> cc@0xf6fd767d  
tpc->segments  
> ((cc@0x9321b1c5  
>   ((0.0 0.0) 90.0)  
>   )  
> )  
caar(tpc->segments)~>??  
> (cc@0x9321b1c5 objType "ccCurve" xExpr "((-2)*t)"  
>   yExpr "(3*t)" range  
>   (1.0 0.0))
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

>)

ccRenameFacet

```
ccRenameFacet (
    o_facet
    t_name
)
=> o_facet2 / nil
```

Description

(Virtuoso Photonics Option) Returns a copy of the specified facet with a different name.

Arguments

<i>o_facet</i>	ID of a ccFacet object.
<i>t_name</i>	A new name for the destination facet.

Value Returned

<i>o_facet2</i>	ID of the resulting ccFacet.
<i>nil</i>	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"
```

Related Topics

[Curvy Core Mathematical Object Functions](#)

Curvy Core Mathematical Object Attribute Query Functions

Curvy core objects such as connectors and curves can be queried using curvy core mathematical object attribute query functions. You can use these functions to check if the given curvy core curve is linear or to return the approximate length of a curve or a polycurve. You can also use these functions to return a point on a curve and to perform other attribute query operations on curvy core mathematical objects.

Related Topics

[ccCubicSplineConnector](#)

[ccGetCurveLength](#)

[ccGetCurvePoint](#)

[ccGetCurveDerivative](#)

[ccGetPolyCurveMinRadius](#)

[cclsLinearCurve](#)

ccCubicSplineConnector

```
ccCubicSplineConnector(  
    l_points  
    [ ?d1 n_d1 ]  
    [ ?d2 n_d2 ]  
)  
=> o_polyCurve / nil
```

Description

(Virtuoso Photonics Option) 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 $n\_x1 < \dots < n\_xN$. <code>List(list(n_x1 n_y1)...list (n_xN n_yN))</code>
$?d1$ 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.

Value Returned

$o\_polyCurve$	ID of the resulting ccPolyCurve.
<code>nil</code>	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:

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
ccCubicSplineConnector(list(2:4 5:2 7:3 10:1) ?d1 0)  
> cc@0x1b6a029
```

Related Topics

[Curvy Core Mathematical Object Attribute Query Functions](#)

ccGetCurveLength

```
ccGetCurveLength(  
    o_curve  
    [ n_discAccuracy ]  
)  
=> n_length / nil
```

Description

Returns the approximate length of a curve or a polyCurve by discretizing it with the specified accuracy and summing up the lengths of the linear segments between the discretization points corrected by additional terms due to curvature.

The additional terms due to curvature are $ds * (ds * (k1 + k2))^2 / 96$, where ds is the length of the linear segment and $k1$ and $k2$ are the curvatures at the ends of the segment.

Arguments

<i>o_curve</i>	ID of a ccCurve or a ccPolyCurve
<i>n_discAccuracy</i>	Discretization accuracy specifying the maximum deviation of the linear segments between the discretization points from the actual curve. The default value is 0.01.

Value Returned

<i>n_length</i>	The approximate length of a curve or a polyCurve.
<i>nil</i>	An error occurred.

Example

```
c=ccCreateCurve("10*C(t/10)" "10*S(t/10)" 0.2:1.2)  
cc@0x6db5618c  
printf("%.16g\n" ccGetCurveLength(c 0.0001))  
0.99999999996324054
```

Related Topics

[Curvy Core Mathematical Object Attribute Query Functions](#)

ccGetCurvePoint

```
ccGetCurvePoint(  
    g_curve  
    n_t  
)  
=> l_xy
```

Description

(Virtuoso Photonics Option) 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  
ccGetCurvePoint(c 0.5)
```


Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
> (0.5 0.25)
ccGetCurvePoint(list(c list(1:1 180)) 0.5)
> (0.5 0.75)
```

Related Topics

[Curvy Core Mathematical Object Attribute Query Functions](#)

ccGetCurveDerivative

```
ccGetCurveDerivative(  
    g_curve  
    n_t  
    [ x_order ]  
)  
=> l_xy
```

Description

(Virtuoso Photonics Option) 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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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)
```

Related Topics

[Curvy Core Mathematical Object Attribute Query Functions](#)

ccGetPolyCurveMinRadius

```
ccGetPolyCurveMinRadius(  
    o_polyCurve  
)  
=> n_minRadius
```

Description

(Virtuoso Photonics Option) Returns the minimum curvature radius of a polyCurve, ignoring any tangent discontinuity points.

Arguments

<i>o_polyCurve</i>	ID of a ccPolyCurve object.

Value Returned

<i>n_minRadius</i>	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
```

Related Topics

[Curvy Core Mathematical Object Attribute Query Functions](#)

ccIsLinearCurve

```
ccIsLinearCurve(  
    o_curve  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Checks if the given ccCurve is linear.

Arguments

<i>o_curve</i>	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
```

Related Topics

[Curvy Core Mathematical Object Attribute Query Functions](#)

Curvy Core Shape Object Attribute Query Functions

The curvy core shape object attribute query functions can be used to get all persistent curvy core shapes in a cellview, get path attributes, or to search for a specific facet.

Related Topics

[ccGetAll](#)

[ccGetPathBoundary](#)

[ccGetPathCenterLine](#)

[ccGetPathEdge](#)

[ccFindSurfaceFacet](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

ccGetAll

```
ccGetAll(  
    d_cellViewId  
)  
=> l_shapes / nil
```

Description

(Virtuoso Photonics Option) Returns all persistent ccShapes in the specified cellview.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.

Value Returned

<i>l_shapes</i>	List of ccShape IDs.
<i>nil</i>	An error occurred.

Example

```
ccGetAll(cv)  
=> (cc@0x269c83f0)
```

Related Topics

[Curvy Core Shape Object Attribute Query Functions](#)

ccGetPathBoundary

```
ccGetPathBoundary(  
    o_path  
)  
=> o_boundary / nil
```

Description

(Virtuoso Photonics Option) Returns the ccBoundary object of the specified ccPath.

Arguments

<i>o_path</i>	ID of a ccPath object.

Value Returned

<i>o_boundary</i>	ID of a ccBoundary object.
<i>nil</i>	An error occurred.

Example

```
ccGetPathBoundary(ccConvertIntoCurvy(dbCreatePathSeg(cv "waveguide" -1:-1 0.5:0.5  
1 "truncate" "truncate")))  
=> cc@0x15a2201f
```

Related Topics

[Curvy Core Shape Object Attribute Query Functions](#)

ccGetPathCenterLine

```
ccGetPathCenterLine(  
    o_path  
)  
=> o_centerLine / nil
```

Description

(Virtuoso Photonics Option) Returns the center line of the path as an open ccPolyCurve.

Arguments

<i>o_path</i>	ID of a ccPath object.

Value Returned

<i>o_centerLine</i>	ID of a ccPolyCurve object forming the center line.
<i>nil</i>	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
```

Related Topics

[Curvy Core Shape Object Attribute Query Functions](#)

ccGetPathEdge

```
ccGetPathEdge(  
    o_path  
    S_side  
)  
=> o_edge / nil
```

Description

(Virtuoso Photonics Option) Creates a new ccPolyCurve that is either the left or the right edge of the boundary of the specified path.

Arguments

<i>o_path</i>	ID of a ccPath object.
<i>S_side</i>	Side of the boundary. Valid values: 'left and 'right

Value Returned

<i>o_edge</i>	ID of a ccPolyCurve object forming the specified edge of the boundary.
<i>nil</i>	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))
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Curvy Core Shape Object Attribute Query Functions](#)

ccFindSurfaceFacet

```
ccFindSurfaceFacet(  
    o_surface  
    t_name  
)  
=> o_facet / nil
```

Description

(Virtuoso Photonics Option) Searches for a facet with the specified name on the specified surface.

Arguments

<i>o_surface</i>	ID of a ccSurface object.
<i>t_name</i>	Name of a facet.

Value Returned

<i>o_facet</i>	ID of the found facet.
<i>nil</i>	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
```

Related Topics

[Curvy Core Shape Object Attribute Query Functions](#)

Curvy Core Transformations Functions

The transform functions can be used to create derived shapes from curvy core shapes.

Related Topics

[ccLayerAnd](#)

[ccLayerAndNot](#)

[ccLayerOr](#)

[ccReversePolyCurve](#)

[ccSmoothenFig](#)

[ccSmoothenPolyCurve](#)

[ccTransformFacet](#)

[ccTransformPolyCurve](#)

ccLayerAnd

```
ccLayerAnd(  
    g_shapes1  
    g_shapes2  
    [ ?cellView d_cellViewId ]  
    [ ?lpp l_lpp ]  
    [ ?facetsToCurves { t | nil } ]  
)  
=> l_surface / nil
```

Description

(Virtuoso Photonics Option) 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

<i>g_shapes1</i>	<i>o_shape</i> (<i>l_shapes</i> ...) A shape or a list of shapes of the first input argument.
<i>g_shapes2</i>	<i>o_shape</i> (<i>l_shapes</i> ...) A shape or a list of shapes of the second input argument.
?cellView <i>d_cellViewId</i>	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 <i>g_shapes1</i> .
?lpp <i>l_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) 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 <i>g_shapes1</i> .

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?facetsToCurves t | nil`

Specifies whether all input facets are converted to curves before the actual operation.

The default is `nil`, which means that the input facets are not converted to curves.

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@0x27eeeea10)
```

Related Topics

Curvy Core Transformations Functions

ccLayerAndNot

```
ccLayerAndNot(  
    g_shapes1  
    g_shapes2  
    [ ?cellView d_cellViewId ]  
    [ ?lpp l_lpp ]  
    [ ?facetsToCurves { t | nil } ]  
)  
=> l_surface / nil
```

Description

(Virtuoso Photonics Option) 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

<i>g_shapes1</i>	<i>o_shape</i> (<i>l_shapes...</i>) A shape or a list of shapes of the first input argument.
<i>g_shapes2</i>	<i>o_shape</i> (<i>l_shapes...</i>) A shape or a list of shapes of the second input argument.
?cellView <i>d_cellViewId</i>	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 <i>g_shapes1</i> .
?lpp <i>l_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) 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 <i>g_shapes1</i> .

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?facetsToCurves t | nil`

Specifies whether all input facets are converted to curves before the actual operation.

The default is `nil`, which means that the input facets are not converted to curves.

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@0x27eeeea10)
```

Related Topics

Curvy Core Transformations Functions

ccLayerOr

```
ccLayerOr(  
    g_shapes1  
    g_shapes2  
    [ ?cellView d_cellViewId ]  
    [ ?lpp l_lpp ]  
    [ ?facetsToCurves { t | nil } ]  
    [ ?mergeFacets { t | nil } ]  
)  
=> l_surface / nil
```

Description

(Virtuoso Photonics Option) 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

<i>g_shapes1</i>	<i>o_shape</i> (<i>l_shapes</i> ...) A shape or a list of shapes of the first input argument.
<i>g_shapes2</i>	<i>o_shape</i> (<i>l_shapes</i> ...) A shape or a list of shapes of the second input argument.
?cellView <i>d_cellViewId</i>	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 <i>g_shapes1</i> .
?lpp <i>l_lpp</i>	<i>tx_layer</i> (<i>tx_layer tx_purpose</i>) 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 <i>g_shapes1</i> .

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?facetsToCurves t | nil`

Specifies whether all input facets are converted to curves before the actual operation.

The default is `nil`, which means that the input facets are not converted to curves.

`?mergeFacets { t | nil }`

Specifies whether adjacent facets should be merged into a single facet if their corners deviate from the resulting facet by less than 1/2 manufacturing grid. The corners of the resulting facet equal the outermost corners of the merged facets and its enter and its angle equal the middle point and the angle between them, respectively.

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=ccLayerOr(list(s1 s2) list(s3))
=> (cc@0x27eeeea10)
```

Related Topics

Curvy Core Transformations Functions

ccMirrorFacet

```
ccMirrorFacet(  
    o_facet  
)  
=> o_facet2 / nil
```

Description

(Virtuoso Photonics Option) 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

<i>o_facet</i>	ID of a ccFacet object.

Value Returned

<i>o_facet2</i>	ID of the mirrored facet.
<i>nil</i>	The operation failed.

Example

```
cMirrorFacet(lfacet)  
=> cc@0x15a2201f
```

Related Topics

[Curvy Core Transformations Functions](#)

ccReversePolyCurve

```
ccReversePolyCurve(  
    o_polyCurve  
)  
=> o_polyCurve2 / nil
```

Description

(Virtuoso Photonics Option) 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

<i>o_polyCurve</i>	ID of a ccPolyCurve object.

Value Returned

<i>o_polyCurve2</i>	ID of the reversed ccPolyCurve object.
<i>nil</i>	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))
```

Related Topics

[Curvy Core Transformations Functions](#)

ccSmoothenFig

```
ccSmoothenFig(  
    o_shape  
    n_radius  
)  
=> o_surface / nil
```

Description

(Virtuoso Photonics Option) Returns the result of smoothening of the specified ccShape as a new transient ccSurface.

Arguments

<i>o_shape</i>	ID of a ccShape object.
<i>n_radius</i>	Radius of the turn made by the bend. With positive radius, this function smoothenes concave cavities of the ccShape object. With negative radius, it smoothenes convex bumps.

Value Returned

<i>o_surface</i>	ID of the resulting ccSurface object.
<i>nil</i>	An error occurred.

Example

```
ccSmoothenFig(path 1)  
=> cc@0x15a2820b
```

Related Topics

[Curvy Core Transformations Functions](#)

ccSmoothenPolyCurve

```
ccSmoothenPolyCurve(  
    o_polyCurve  
    n_radius  
)  
=> o_polyCurve2 | nil
```

Description

(Virtuoso Photonics Option) Returns a smoothed copy of the given ccPolyCurve.

Arguments

<i>o_polyCurve</i>	ID of a ccPolyCurve object.
<i>n_radius</i>	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

<i>o_polyCurve2</i>	ID of the smoothed ccPolyCurve object.
<i>nil</i>	An error occurred.

Example

```
ccSmoothenPolyCurve(pc)  
=> cc@0x269c83f0
```

Related Topics

[Curvy Core Transformations Functions](#)

ccTransformFacet

```
ccTransformFacet(  
    o_facet  
    l_transform  
)  
=> o_facet2 | nil
```

Description

(Virtuoso Photonics Option) Returns a transformed copy of the given ccFacet object.

Arguments

<i>o_facet</i>	ID of a ccFacet object.
<i>l_transform</i>	$((n\_offsetX\ n\_offsetY)\ n\_angle)$ A transformation object specified by an offset specified by <i>n_offsetX</i> and <i>n_offsetY</i> and a rotation angle <i>n_angle</i> around (0,0).

Value Returned

<i>o_facet2</i>	ID of the transformed ccFacet object.
<i>nil</i>	The operation failed.

Example

```
ccTransformFacet(lfacet list(0:0 90))  
=> cc@0x15a2201f
```

Related Topics

[Curvy Core Transformations Functions](#)

ccTransformPolyCurve

```
ccTransformPolyCurve(  
    o_polyCurve  
    l_transform  
)  
=> o_polycurve2 | nil
```

Description

(Virtuoso Photonics Option) Returns a transformed copy of the given ccPolyCurve.

Arguments

<i>o_polycurve</i>	ID of a ccPolyCurve object.
<i>l_transform</i>	$((n\_offsetX\ n\_offsetY)\ n\_angle)$ A transformation object specified by an offset specified by <i>n_offsetX</i> and <i>n_offsetY</i> and a rotation angle <i>n_angle</i> around (0,0).

Value Returned

<i>o_polycurve2</i>	ID of the transformed polyCurve object.
<i>nil</i>	The operation failed.

Example

```
pc = ccCreatePolyCurve(ccCreateCurve("t" "0" 0:1))  
=> cc@0x1d42cee  
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)))
```

Related Topics

Curvy Core Transformations Functions

Curvy Core Connector Functions

The curvy core engine supports various types of connectors. You can use functions to create curvy core connectors such as bend, curve, spiral, waypoint, and wide connectors.

Related Topics

[ccBendConnector](#)

[ccCreateConnectorWidthExpressionList](#)

[ccCurveConnector](#)

[ccSpiralConnector](#)

[ccTromboneConnector](#)

[ccWaypointConnector](#)

[ccWideConnector](#)

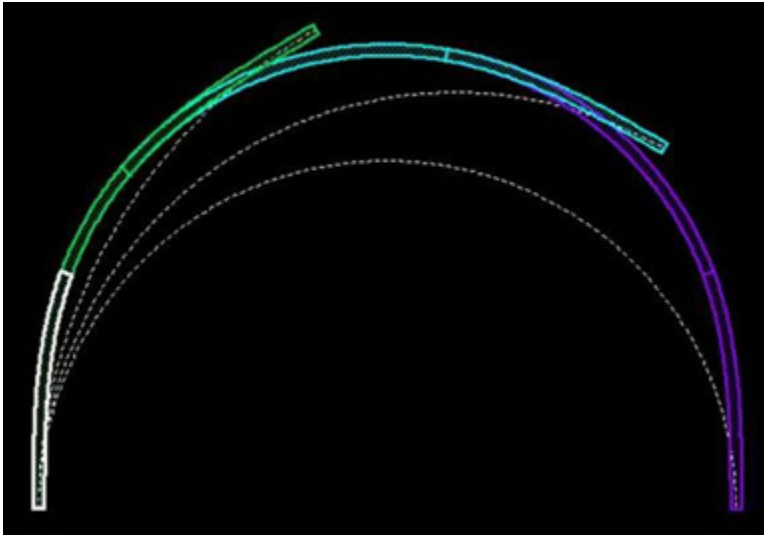
ccBendConnector

```
ccBendConnector(  
    l_port  
    n_radius  
    n_bangle  
    n_rmin  
    n_gamma  
)  
=> l_connector / nil
```

Description

(Virtuoso Photonics Option) 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 $1 - (\text{cangle} / \text{bangle})$ bend connectors are created containing fixed clothoid sections with angle varying from 0 to `cangle` and radius varying from 0 to `rmin`.



Constraints

- $0 < |\text{bangle}| < 2 * \text{PI}$
- $0 \leq \text{gamma} \leq 1$

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

- $\gamma = 0$ or $\gamma = 1$ when $\text{bangle} > \text{PI}$

Arguments

<i>l_port</i>	<code>list(nil 'xy l_point 'facet l_facet) nil</code> The starting port. where, ■ <i>l_point</i> <code>list(n_x n_y)</code> A list of two-dimensional point represented by <i>n_x</i> and <i>n_y</i>. ■ <i>l_facet</i> <code>list(nil 'angle n_angle ['radius n_radius] ['width n_width])</code> A facet is specified by the angle <i>n_angle</i>. Values for <i>n_radius</i> and <i>n_width</i> are not required and are ignored if specified.
<i>n_radius</i>	Radius of the turn made by the bend.
<i>n_bangle</i>	Angle of the turn made by the bend. Valid values: Greater than 0 and less than the product of 2 and Pi.
<i>n_rmin</i>	Minimum allowed curvature radius.
<i>n_gamma</i>	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.

Value Returned

<code>l_connector</code>	<pre>list(nil 'connector o_polyCurve 'message t_message)</pre> The resulting connector (ccPolyCurve ID) bundled with debug or error information. where, ■ <code>o_polyCurve</code> specifies the resulting ccPolyCurve connector. If <code>o_polyCurve = nil</code>, the connector cannot be implemented. ■ <code>t_message</code> is a string holding the debug information or an error message describing the reason why the connector calculation was not successfully completed.
<code>nil</code>	An error occurred, the connector cannot be implemented.

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")
```

Related Topics

[ccWaypointConnector](#)

[Curvy Core Connector Functions](#)

ccCreateConnectorWidthExpressionList

```
ccCreateConnectorWidthExpressionList(  
    n_startWidth  
    n_endWidth  
    S_taperStyle  
    o_polyCurve  
    [ ?startClamp n_startClampLength ]  
    [ ?endClamp n_endClampLength ]  
)  
=> l_segmentsWidths
```

Description

(Virtuoso Photonics Option) 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:

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

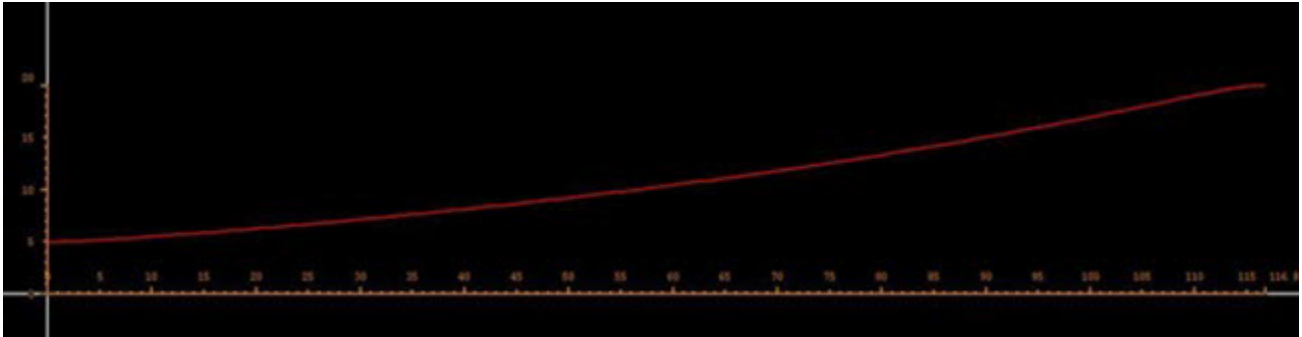
At each end of the connector:

$$w(t) = f(t)$$

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

<i>n_startWidth</i>	The start width of the connector.
<i>n_endWidth</i>	The end width of the connector.
<i>S_taperStyle</i>	The taper style of the connector. Valid values: 'linear', 'exp', 'parabolic, and 'log.
<i>o_polyCurve</i>	A polyCurve specifying the centerline of the connector with the segments parametrized by the arc length.
<i>n_startClampLength</i>	Length of the clamping region at the start of the connector.
<i>n_endClampLength</i>	Length of the clamping region at the end of the connector.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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")

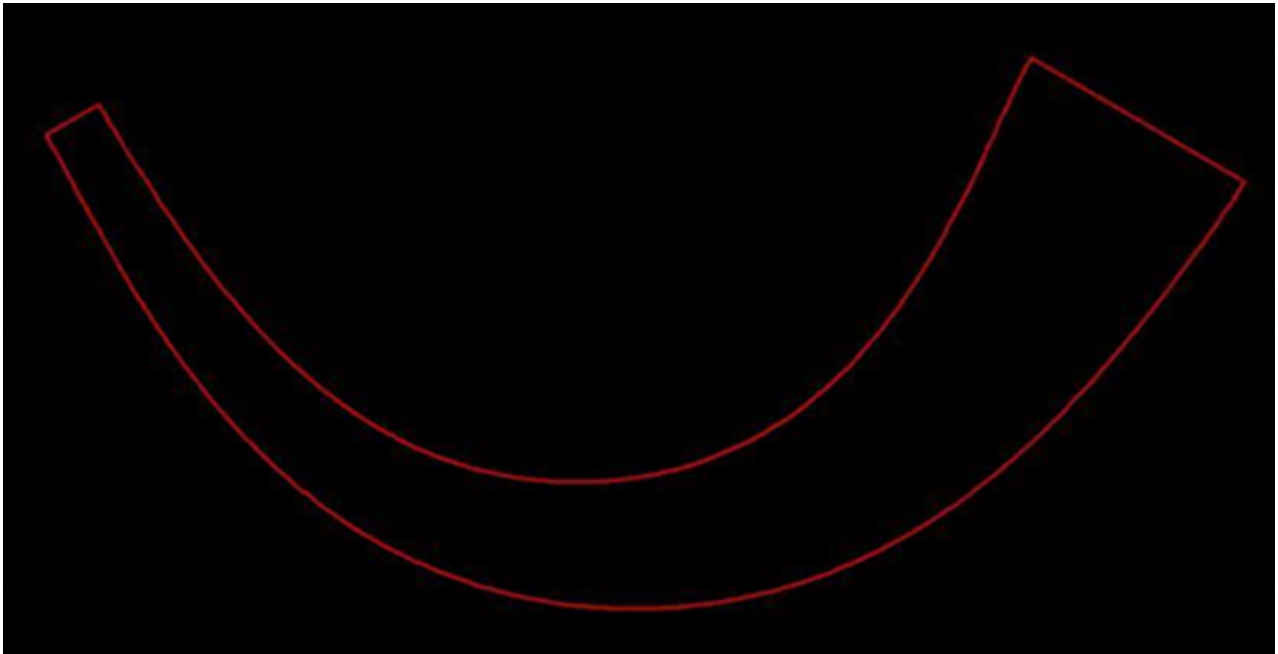
s = ccCreateConnectorWidthExpressionList(5 20 'exp c->connector ?startClamp 3.0
?endClamp 3.0)
> ((cc@0xb1fb573d
  ((-7.105427e-15 -3.552714e-15) -60.0)
  "5*exp((0.01186848385793852)*(1*t+(0)))+(0.2021274047854202)*exp(-t/(3))+(-
  9.0738813184265e-18)*exp(t/(3))+(-0.2021274047854202)+(0.008033382305447485)*t"
  )
  (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"
```


Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
)  
)
```

```
ccGenFigs(ccCreatePath(cv "y1" s))  
> (db:0x21d0d49b)
```



Related Topics

[ccBendConnector](#)

[ccCurveConnector](#)

[ccWaypointConnector](#)

[Curvy Core Connector Functions](#)

ccCurveConnector

```
ccCurveConnector(  
    l_port1  
    l_port2  
    n_rmin  
    [ ?method x_method ]  
    [ ?parameters l_parameters ]  
)  
=> l_connector | nil
```

Description

(Virtuoso Photonics Option) 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

<i>l_port1</i>	<pre>list(nil 'xy l_point 'facet l_facet)</pre> Starting port for the connector where: ■ <i>l_point</i> <pre>list(n_x n_y)</pre> A two-dimensional point represented by <i>n_x</i> and <i>n_y</i>. ■ <i>l_facet</i> <pre>list(nil 'angle n_angle ['radius n_radius] ['width n_width])</pre> A facet is specified by the angle <i>n_angle</i>. Values for <i>n_radius</i> and <i>n_width</i> are not required and are ignored if specified.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

<i>l_port2</i>	<pre>list(nil 'xy l_point 'facet l_facet)</pre> Ending port for the connector. ■ <i>l_point</i> <pre>list(n_x n_y)</pre> A two-dimensional point represented by <i>n_x</i> and <i>n_y</i>. ■ <i>l_facet</i> <pre>list(nil 'angle n_angle ['radius n_radius] ['width n_width])</pre> A facet is specified by the angle <i>n_angle</i>. Values for <i>n_radius</i> and <i>n_width</i> are not required and are ignored if specified.
<i>n_rmin</i>	Minimally allowed curvature radius.
<i>?method x_method</i>	Method number from 0 to 9. The default is 0. For more information see Additional Information.
<i>?parameters l_parameters</i>	<pre>list(n_parameters...)</pre> List of additional numeric parameters passed to the connector calculation methods. The default is <code>nil</code>.

Value Returned

<i>l_connector</i>	<pre>list(nil 'connector o_polyCurve 'message t_message ['segments l_segments])</pre> The resulting connector (ccPolyCurve ID) bundled with debug or error logs, where, ■ <i>o_polyCurve</i> is the resulting ccPolyCurve connector. If <i>o_polyCurve</i> = nil, the connector cannot be implemented. ■ <i>t_message</i> is a string holding the debug printouts and error messages. ■ <i>l_segments</i> is a list of segment lengths and minimum or maximum curvatures rather than the segments themselves (<i>o_polyCurve</i>~>segments) and is only set if <i>l_method</i>>0. If <i>l_method</i>=0 or in case of error, <i>l_segments</i> is set to nil
nil	An error occurred, the connector cannot be implemented.

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")
```

Additional Information

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

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

2	SmallCircleLine
3	LargeCircleLine with varying clothoid length
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.

Related Topics

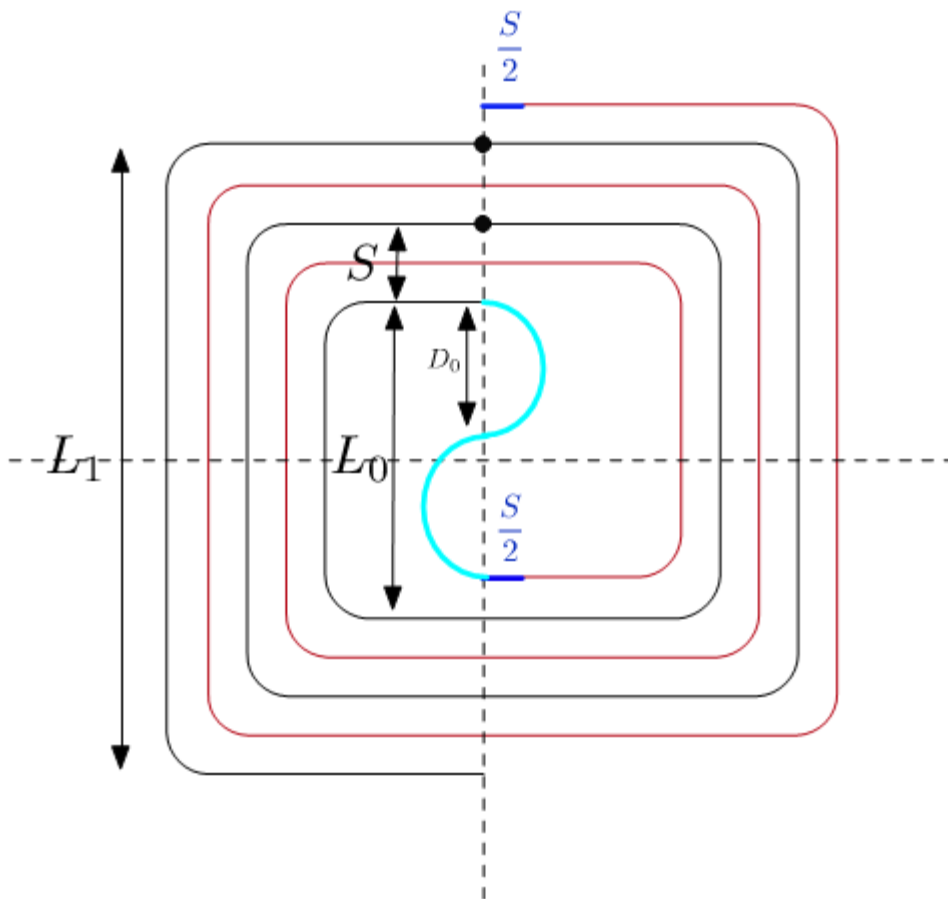
Curvy Core Connector Functions

ccSpiralConnector

```
ccSpiralConnector(  
    n_lengthInnerSpiral0  
    n_lengthOuterSpiral1  
    x_numberOfTurns  
    n_minRadius  
    n_gamma  
    [ ?portLocation t_location]  
)  
=> l_connector / nil
```

Description

(Virtuoso Photonics Option) 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.



Arguments

n_lengthInnerSpiral0

Size of the innermost turn of the spiral, $L0$.

n_lengthOuterSpiral1

Size of the outermost turn of the spiral $L1$.

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.

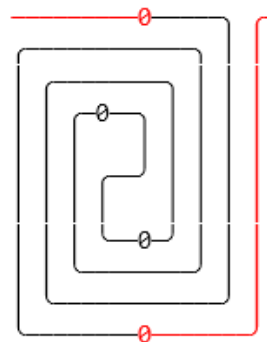
?portLocation *t_location*

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.



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

<i>l_connector</i>	<pre>list(nil 'connector o_polyCurve 'message t_message)</pre> The resulting connector and a message string where, ■ <i>o_polyCurve</i> is the ID of the resulting ccPolyCurve connector. If <i>o_polyCurve</i> = <i>nil</i>, the connector cannot be implemented. ■ <i>t_message</i> is a string holding the debug printouts and error messages.
<i>nil</i>	An error occurred, the connector cannot be implemented.

Example

```
ccSpiralConnector(50 100 3 6 0)  
=> (nil connector cc@0xc8c2798a message "Length=2162.349132 MinRadius=6.000000")
```

Related Topics

[Curvy Core Connector Functions](#)

ccTromboneConnector

```
ccTromboneConnector(  
    l_legs  
    n_rmin  
    n_gamma  
)  
=> l_connector / nil
```

Description

(Virtuoso Photonics Option) 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

<i>l_legs</i>	(<i>n_l1 n_l2 n_l3 n_l4 n_l5</i>) List of five length values for the legs of the trombone.
<i>n_rmin</i>	Minimally allowed curvature radius.
<i>n_gamma</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

<i>l_connector</i>	<pre>list(nil 'connector o_polyCurve 'message t_message)</pre> The resulting connector (ccPolyCurve ID) and a message string. where, ■ <i>o_polyCurve</i> is the resulting ccPolyCurve connector. If <i>o_polyCurve</i> = nil, the connector cannot be implemented. ■ <i>t_message</i> is a string holding the debug printouts and error messages.
<i>nil</i>	An error occurred, the connector cannot be implemented.

Example

```
ccTromboneConnector('(50 50 50 50 50) 12 0)  
=> (nil connector cc@0xc7e73ea2 message "Length=221.267246 MinRadius=12.000000")
```

Related Topics

[Curvy Core Connector Functions](#)

ccWaypointConnector

```
ccWaypointConnector(  
    l_port1  
    l_port2  
    l_points  
    n_rmin  
    n_gamma  
)  
=> l_connector / nil
```

Description

(Virtuoso Photonics Option) 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`, they specify the initial and the final points, tangents, and radiuses of the connector.

If no waypoints are specified, the start and the end ports are connected by a curve connector.

If a single waypoint is specified, it is connected to the start and the end ports 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 ports 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 ports 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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Arguments

<i>l_port1</i>	<code>list(nil 'xy l_point 'facet l_facet) nil</code> Starting port for the connector. Unlike <code>ccCurveConnector</code> , ports in waypoint connector are allowed to be simultaneously set to <code>nil</code> . However, a <code>nil</code> port specified with a non- <code>nil</code> port will result in an error.
<i>l_port2</i>	<code>list(nil 'xy l_point 'facet l_facet) nil</code> Ending port for the connector. Unlike in <code>ccCurveConnector</code> , ports in waypoint connector are allowed to be simultaneously set to <code>nil</code> . However, a <code>nil</code> port + non- <code>nil</code> port will result in an error.
<i>l_point</i>	<code>list(n_x n_y)</code> A two-dimensional point.
<i>l_facet</i>	<code>list(nil 'angle n_angle ['radius n_radius] ['width n_width])</code> A facet is specified by the angle <i>n_angle</i> . Values for <i>n_radius</i> and <i>n_width</i> are not required and are ignored if specified.
<i>l_points</i>	<code>(l_point...)</code> A list of two-dimensional points.
<i>n_rmin</i>	Minimally allowed curvature radius.
<i>n_gamma</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

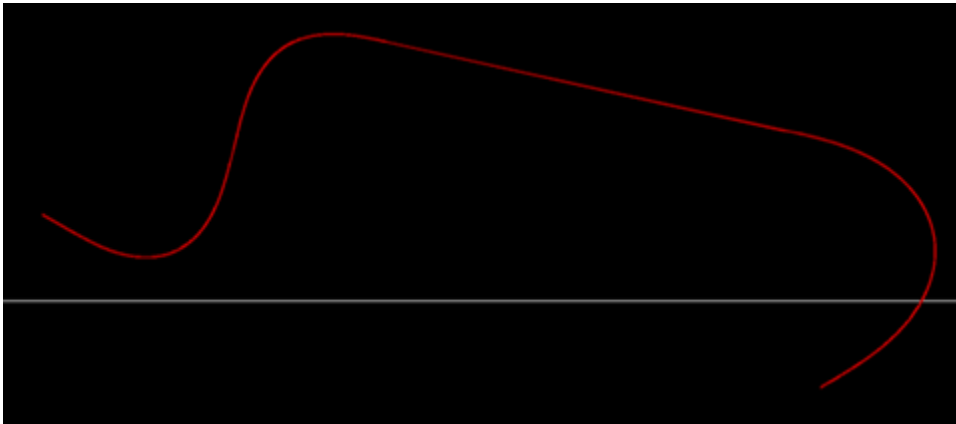
<i>l_connector</i>	<pre>list(nil 'connector o_polyCurve 'message t_message ['incomplete t_incomplete])</pre> The resulting connector (ccPolyCurve ID) bundled with debug or error logs. where, ■ <i>o_polyCurve</i> is the resulting ccPolyCurve connector. If <i>o_polyCurve</i> = nil, the connector cannot be implemented. ■ If <i>t_incomplete</i> is not nil, then the connector only contains segments between the starting point and the point where the calculation failed. ■ <i>t_message</i> is a string holding the debug printouts and error messages. ■ <i>t_incomplete</i> is a string describing why connector can only be partially calculated. If the complete connector could be calculated, the value is nil.
<i>nil</i>	An error occurred, the connector cannot be implemented.

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(getEditCellView() list("y1" "drawing") wp->connector->segments  
?defaultWidth 0.5)  
ccGenFigs(p)
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)



Related Topics

[Curvy Core Connector Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

ccWideConnector

```
ccWideConnector(  
    g_connector  
    l_lengthWidths  
    n_clamp1  
    n_clamp2  
)  
=> l_segmentWidths
```

Description

(Virtuoso Photonics Option) 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...*)

where,

- *o_polyCurve* is the ID of a ccPolyCurve object.
- *l_segment* is a list of polyCurve segments 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*).

Each polyCurve segment is assumed to be parametrized by the ascending or descending arc length and an arbitrary initial value.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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.

When set to `nil`, 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.

When set to `nil`, the spline function has zero second derivative at the last connector point.

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.

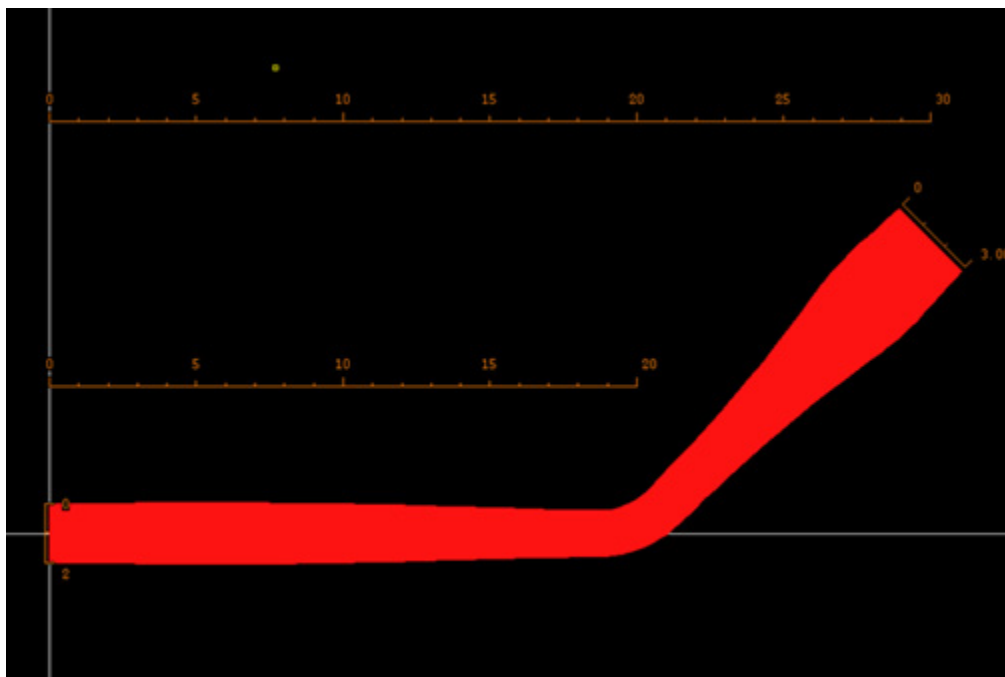
Example

```
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
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
((0.0 0.0) 0.0) "((3+(0.1*t))+(0.03576862324*(t^2)))+((-  
0.01512250106)*(t^3)))"  
)  
)
```



Related Topics

[Curvy Core Connector Functions](#)

Curvy Core Optical Rule Engine Functions

The SKILL functions associated with the optical rule engine (ORE) can be used to evaluate ORE expressions, compile the ORE expressions specified as a file or a list, or retrieve a list of ORE context names. You can also use the ORE SKILL functions to get ORE contexts or to purge specific or all contexts.

Related Topics

[ccORE](#)

[ccORECompile](#)

[ccOREGetContext](#)

[ccOREGetContextObject](#)

[ccOREGetContexts](#)

[ccOREPurgeContext](#)

[ccOREPurgeContexts](#)

ccORE

```
ccORE(  
    [ ?rule g_rule ]  
    [ ?sourceCellView o_sourceCellView ]  
    [ ?destinationCellView o_destinationCellView ]  
    [ ?filterLPP l_filterLPP ]  
    [ ?stopLevel x_stopLevel ]  
    [ ?parameters l_parameters ]  
    [ ?loadContext t_loadContext ]  
    [ ?saveContext t_saveContext ]  
    [ ?saveResult s_saveResult ]  
    [ ?trace g_trace ]  
)  
=> t
```

Description

Evaluates Optical Rule Engine (ORE) expressions specified in a file or a SKILL list. Expressions are evaluated on the source cellview and the resulting markers and shapes are stored in the source cellview or optionally in the destination cellview. The evaluation can be limited only to shapes on the specified layer-purpose pairs and down to the specified level of hierarchy.

The default values defined in the expressions can be overridden by the specified parameter values. The ORE context can be saved at the end of the run under a specified name and loaded at the beginning of the next run by referring to the same name. The evaluation of ORE expressions can optionally be traced and the list of ORE objects resulting from them saved in a SKILL variable.

Arguments

?rule g_rule *t_fileName* | (*g_expression...*)

Name of the file containing the ORE expressions or a list of ORE expressions, where each *g_expression* can be a string, a number, a variable, or a list containing the name of an ORE function and expressions representing the values of its arguments:

```
g_expression: t_string | n_number |  
s_variable | (s_function [ g_expression...  
])
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?sourceCellView *o_sourceCellView*

ID of the source cellview defining the ORE relations 'design', 'shape', and 'cell'. The source cellview must be readable.

?destinationCellView *o_destinationCellView*

ID of the destination cellview in which ORE markers and shapes are created. The destination cellview must be writable.

The default destination cellview is the source cellview.

?filterLPP *l_filterLPP*

((t_layer t_purpose) ...)

List of layer-purpose pairs of shapes to be included in the ORE relation 'shape'.

The default is to include shapes on all layer-purpose pairs.

?stopLevel *x_stopLevel*

Number of hierarchy levels of shapes and cells to be included in the ORE relations 'shape' and 'cell'. The value 0 corresponds to the shapes only in the source cellview without the cell instances inside.

The default is to include shapes and cells on all hierarchy levels.

?parameters *l_parameters*

((s_parameter g_value) ...)

List of parameter values overriding the default values defined in *g_rule*, where *s_parameter* is the parameter name and *g_value* can be a number, a string, a Boolean, or a list containing the name of an ORE function and expressions representing the values of its arguments:

*g_value: n_number | t_string | b_boolean |
(s_function [g_expression...])*

?loadContext *t_loadContext*

Name of the ORE context that is loaded at the beginning of the ORE run.

?saveContext *t_saveContext*

Name under which the ORE context is saved at the end of the ORE run.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?saveResult s_saveResult`

Name of the SKILL variable in which the list of ORE objects resulting from the evaluation of ORE expressions is saved at the end of the ORE run. Each ORE object is represented by a read-only SKILL defstruct.

`?trace g_trace`

A boolean specifying whether the trace of ORE expression evaluations is printed to CIW.

Value Returned

`t`

The specified ORE expressions are evaluated successfully.
An error is displayed if the operation fails.

Example

Derive a relation containing numbers from 1 to N in tuples of order 1, overriding the default parameter value N=1 by N=3 and storing the result in the variable `ore`.

```
ccORE(?rule '((expand ns (ns) (transform (d) design ((sequence N)))) ?parameters
'((N 3)) ?saveResult 'ore)
=> t
```

Print variable `ore` containing the derived relation

```
ore
=> (ccORERelation@0x278f7b00)
```

Print tuples contained in the derived relation

```
ore~>tuples~>components
=> (((ccORENumber@0x278f7a70) (ccORENumber@0x278f7aa0) (ccORENumber@0x278f7ad0)))
```

Print values of the components of the tuples in the derived relation.

```
ore~>tuples~>components~>value
=> (((3.0) (2.0) (1.0)))
```

Related Topics

[SKILL ORE Object Representation](#)

[Curvy Core Optical Rule Engine Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Optical Rule Engine

ccORECompile

```
ccORECompile(  
    g_rule  
    t_outputFileName  
    [ ?header t_header ]  
)  
=> t
```

Description

Compiles the Optical Rule Engine (ORE) expressions specified as a file or a list and stores them in an encrypted form including an optional unencrypted header in a file with the specified name. The encrypted output file can be specified as the input *g_rule* key argument of the `ccORE` function.

Arguments

<i>g_rule</i>	<i>t_fileName</i> (<i>g_expression</i> ...) Name of the file containing the ORE expressions or a list of ORE expressions, where each <i>g_expression</i> can be a string, a number, a variable, or a list containing the name of an ORE function and expressions representing the values of its arguments: <i>g_expression</i> : <i>t_string</i> <i>n_number</i> <i>s_variable</i> (<i>s_function</i> [<i>g_expression</i> ...])
<i>t_outputFileName</i>	Name of the output file containing compiled rule expressions in encrypted form.
<i>t_header</i>	Unencrypted header to be included in the output file.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

t The specified ORE expressions are compiled successfully.
 An error is displayed if the operation fails.

Example

Compile a rule defining a relation containing numbers from 1 to N in tuples of order 1 in a file with the name `encrypted.ore` and the header "Numbers from 1 to 3".

```
ccORECompile('((define numbers (expand ns (ns) (transform (d) design ((sequence 3)))))) "encrypted.ore" ?header "Numbers from 1 to 3")  
=>t
```

Run the rule defined in the file `encrypted.ore` and store the result in the variable `ore`.

```
ccORE(?rule "encrypted.ore" ?saveResult 'ore)  
=>t
```

Print values of the components of the tuples in the derived relation.

```
ore~>tuples~>components~>value  
=> (((3.0) (2.0) (1.0)))
```

Related Topics

[Curvy Core Optical Rule Engine Functions](#)

[Optical Rule Engine](#)

ccOREGetContext

```
ccOREGetContext(  
    t_contextName  
)  
=> l_context
```

Description

Retrieves the Optical Rule Engine (ORE) context with the specified name.

Argument

<i>t_contextName</i>	Name of the ORE context.

Value Returned

<i>l_context</i>	<code>list(nil 's_objectName r_object ...)</code>

Disembodied property list representing the ORE context, where:

- *s_objectName* is the name of the ORE object.
- *r_object* is the defstruct representing the ORE object.

The operation fails and an error is displayed if the context with the specified name does not exist.

Example

Define variable `PI_2` and save the resulting ORE context under the name `ore_cxt`.

```
ccORE(?rule '((define PI_2 PI/2)) ?saveContext "ore_cxt")  
=> t
```

Get the ORE context with the name `ore_cxt`.

```
ccOREGetContext("ore_cxt")  
=> (nil PI_2 ccORENumber@0x27fd3b48 context ccOREString@0x27fd3b60 cell  
ccORERelation@0x27fd3b78 design  
ccORERelation@0x27fd3bd8 PI ccORENumber@0x27fd3bf0 shape ccORERelation@0x27fd3c08  
)
```

Get the ORE object stored in the variable `PI_2` in the ORE context with the name `ore_cxt`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
ccOREGetContext("ore_cxt")->PI_2  
=> ccORENumber@0x27fd3c20
```

Get the value of the ORE object stored in the variable `PI_2` in the ORE context with the name `ore_cxt`.

```
ccOREGetContext("ore_cxt")->PI_2->value  
=> 1.570796
```

Related Topics

[Curvy Core Optical Rule Engine Functions](#)

[Optical Rule Engine](#)

ccOREGetContextObject

```
ccOREGetContextObject(  
    t_contextName  
    t_objectName  
)  
=> r_object
```

Description

Retrieves the defstruct representing the specified Optical Rule Engine (ORE) object in the specified ORE context.

Arguments

<i>t_contextName</i>	Name of the ORE context.
<i>t_objectName</i>	Name of the ORE object.

Value Returned

<i>r_object</i>	Defstruct representing the ORE object. The operation fails and an error is displayed if the context with the given name does not exist.

Example

Define the variable `PI_2` and save the resulting ORE context under the name `ore_cxt`.

```
ccORE(?rule '((define PI_2 PI/2)) ?saveContext "ore_cxt")  
=> t
```

Get the ORE object stored in the variable `PI_2` in the ORE context with the name `ore_cxt`.

```
ccOREGetContextObject("ore_cxt" "PI_2")  
=> ccORENumber@0x27fd3dd0
```

Get the value of the ORE object stored in variable `PI_2` in the ORE context with the name `ore_cxt`.

```
ccOREGetContextObject("ore_cxt" "PI_2")->value  
=> 1.570796
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Curvy Core Optical Rule Engine Functions](#)

[Optical Rule Engine](#)

ccOREGetContexts

```
ccOREGetContexts (  
    )  
=> l_contextNames
```

Description

Retrieves a list of Optical Rule Engine (ORE) context names.

Arguments

None

Value Returned

```
l_contextNames      (t_contextName...)  
A list of ORE context names.
```

Example

Define variable `PI_2` and save the resulting ORE context under the name `ore_cxt`.

```
ccORE(?rule '((define PI_2 PI/2)) ?saveContext "ore_cxt")  
=> t
```

Get the list of the names of the ORE contexts.

```
ccOREGetContexts()=> ("ore_cxt")
```

Related Topics

[Curvy Core Optical Rule Engine Functions](#)

[Optical Rule Engine](#)

ccOREPurgeContext

```
ccOREPurgeContext(  
    t_contextName  
)  
=> t / nil
```

Description

Purges the optical rule engine (ORE) context with the specified name.

Arguments

<i>t_contextName</i>	Name of the ORE context to be pruned.

Value Returned

<i>t</i>	ORE context with the specified name existed and is purged.
<i>nil</i>	ORE context with the specified name does not exist and cannot be purged or if the operation fails.

Example

Define variable `PI_2` and save the resulting ORE context under the name `ore_cxt`.

```
ccORE(?rule '((define PI_2 PI/2)) ?saveContext "ore_cxt")  
=> t
```

Get the list of the names of the ORE contexts.

```
ccOREGetContexts()  
=> ("ore_cxt")
```

Purge ORE context with the name `ore_cxt`.

```
ccOREPurgeContext("ore_cxt")  
=> t
```

Get the list of the names of the ORE contexts.

```
ccOREGetContexts()=> nil
```


Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Curvy Core Optical Rule Engine Functions](#)

[Optical Rule Engine](#)

ccOREPurgeContexts

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

Description

Purges all Optical Rule Engine (ORE) contexts.

Arguments

None

Value Returned

t	All ORE contexts are purged.
nil	The operation failed.

Example

Define variable `PI_2` and save the resulting ORE context under the name `ore_cxt`.

```
ccORE(?rule '((define PI_2 PI/2)) ?saveContext "ore_cxt")  
=> t
```

Get the list of the names of the ORE contexts.

```
ccOREGetContext()  
=> ("ore_cxt")
```

Purge all ORE contexts.

```
ccOREPurgeContexts()  
=> t
```

Get the list of the names of the ORE contexts.

```
ccOREGetContexts()  
=> nil
```

Related Topics

[Curvy Core Optical Rule Engine Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Optical Rule Engine

Photonics Database Functions

You can use the photonic database functions to work with photonic pinFigs. From checking if a cellview has photonic pinFigs to getting the attributes of a photonic pinFig, such as its angle, radius, or width, to setting or unsetting these attributes, all such pinFig database related tasks can be performed using the photonic database functions.

Related Topics

- [dbCellViewHasPhotonicPinFig](#)
- [dbGetPhotonicPinFigAngle](#)
- [dbGetPhotonicPinFigRadius](#)
- [dbGetPhotonicPinFigWidth](#)
- [dbIsPhotonicsEnabled](#)
- [dbSetPhotonicPinFigAngle](#)
- [dbSetPhotonicPinFigRadius](#)
- [dbSetPhotonicPinFigWidth](#)
- [dbUnsetPhotonicPinFigAngle](#)
- [dbUnsetPhotonicPinFigRadius](#)
- [dbUnsetPhotonicPinFigWidth](#)

dbCellViewHasPhotonicPinFig

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

Description

(Virtuoso Photonics Option) Checks whether the specified cellview has a photonic pinFig.

Arguments

<i>d_cvId</i>	Database ID of a cellview.

Return Value

<i>t</i>	Specified cellview has a photonic pinFig.
<i>nil</i>	Specified cellview does not have a photonic pinFig.

Example

The specified cellview contains a photonic pinFig.

```
dbCellViewHasPhotonicPinFig(cvId)  
=> t
```

Related Topics

[Photonics Database Functions](#)

dbGetPhotonicPinFigAngle

```
dbGetPhotonicPinFigAngle(  
    d_pinFigId  
)  
=> n_angle / nil
```

Description

(Virtuoso Photonics Option) Returns the attribute value of the angle for the specified photonics pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.

Value Returned

<i>n_angle</i>	Attribute value of the angle for the specified photonics pinFig.
<i>nil</i>	The pinFig is not a photonic pin or the attribute is not set.

Example

Attribute value of the angle on a photonic pinFig pinFigId is 15.5.

```
dbGetPhotonicPinFigAngle(pinFigId)  
=> 15.5
```

Related Topics

[Photonics Database Functions](#)

dbGetPhotonicPinFigRadius

```
dbGetPhotonicPinFigRadius(  
    d_pinFigId  
)  
=> n_radius / nil
```

Description

(Virtuoso Photonics Option) Returns the attribute value of the radius for the specified photonics pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.

Value Returned

<i>n_radius</i>	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

Attribute value of the radius on a photonic pinFig `pinFigId` is 5.

```
dbGetPhotonicPinFigRadius(pinFigId)  
=> 5
```

Related Topics

[Photonics Database Functions](#)

dbGetPhotonicPinFigWidth

```
dbGetPhotonicPinFigWidth(  
    d_pinFigId  
)  
=> n_width / nil
```

Description

(Virtuoso Photonics Option) Returns the attribute value of the width for the specified photonics pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.

Value Returned

<i>n_width</i>	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

Attribute value of the width on a photonic pinFig `pinFigId` is 1.

```
dbGetPhotonicPinFigWidth(pinFigId)  
=> 1
```

Related Topics

[Photonics Database Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

dbIsPhotonicsEnabled

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

Description

(Virtuoso Photonics Option) 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
```

Related Topics

[Photonics Database Functions](#)

dbSetPhotonicPinFigAngle

```
dbSetPhotonicPinFigAngle(  
    d_pinFigId  
    n_value  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Sets the given attribute value as the angle of the specified photonic pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.
<i>n_value</i>	Angle attribute value to be set.

Value Returned

<i>t</i>	Specified attribute value is set as the angle of the photonics pinFig.
<i>nil</i>	Angle attribute value is not set.

Example

Set 15.5 as the angle attribute value on the photonic pinFig *pinFigId*.

```
dbSetPhotonicPinFigAngle(pinFigId 15.5)  
=> t
```

Related Topics

[Photonics Database Functions](#)

dbSetPhotonicPinFigRadius

```
dbSetPhotonicPinFigRadius(  
    d_pinFigId  
    n_value  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Sets the given attribute value as the radius of the specified photonic pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.
<i>n_value</i>	Radius attribute value to be set.

Value Returned

<i>t</i>	Specified attribute value is set as the radius of the photonics pinFig.
<i>nil</i>	Radius attribute value is not set.

Example

Set 5 as the radius attribute value on the photonic pinFig *pinFigId*.

```
dbSetPhotonicPinFigRadius(pinFigId 5)  
=> t
```

Related Topics

[Photonics Database Functions](#)

dbSetPhotonicPinFigWidth

```
dbSetPhotonicPinFigWidth(  
    d_pinFigId  
    n_value  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Sets the given attribute value as the width of the specified photonic pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.
<i>n_value</i>	Width attribute value to be set.

Value Returned

<i>t</i>	Specified attribute value is set as width of the photonics pinFig.
<i>nil</i>	Width attribute value is not set.

Example

Set 5 as the width attribute value on the photonic pinFig *pinFigId*.

```
dbSetPhotonicPinFigWidth(pinFigId 5)  
=> t
```

Related Topics

[Photonics Database Functions](#)

dbUnsetPhotonicPinFigAngle

```
dbUnsetPhotonicPinFigAngle(  
    pinFigId  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Resets the attribute value of the angle for the specified photonics pinFig.

Arguments

<i>pinFigId</i>	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

Angle attribute value of the photonic pinFig `pinFigId` is reset.

```
dbUnsetPhotonicPinFigAngle(pinFigId)  
=> t
```

Related Topics

[Photonics Database Functions](#)

dbUnsetPhotonicPinFigRadius

```
dbUnsetPhotonicPinFigRadius(  
    pinFigId  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Resets the attribute value of the radius for the specified photonics pinFig.

Arguments

<i>pinFigId</i>	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

Radius attribute value of the photonic pinFig *pinFigId* is reset.

```
dbUnsetPhotonicPinFigRadius(pinFigId)  
=> t
```

Related Topics

[Photonics Database Functions](#)

dbUnsetPhotonicPinFigWidth

```
dbUnsetPhotonicPinFigWidth(  
    pinFigId  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Resets the attribute value of the width for the specified photonics pinFig.

Arguments

<i>pinFigId</i>	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

Width attribute value of the photonic pinFig *pinFigId* is reset.

```
dbUnsetPhotonicPinFigWidth(pinFigId)  
=> t
```

Related Topics

[Photonics Database Functions](#)

Photonics Flow Functions

The photonics flow heavily relies on the use of composite waveguides, physical connection points, and the modeProp table. The SKILL functions that are associated with the photonics flow can be used to create composite waveguides and set the specified composite waveguides into a composite of the specified master. You can also use these functions to edit, and decompose waveguides, create ports, connector anchors, and work with the modeProp table.

Related Topics

- [phoCompose](#)
- [phoComputeModeProps](#)
- [phoHiCompose](#)
- [phoCreatePort](#)
- [phoGetConnectorAnchors](#)
- [phoGenWaveguide](#)
- [phoTechHasModeProp](#)

phoAddWaveguidePorts

```
phoAddWaveguidePorts (
    ccPath
    t_startPortName
    t_startPortDirection
    t_endPortName
    t_endPortDirection
    [ ?n_portPinWidth n_portPinWidth ]
    [ ?startWidth f_width ]
    [ ?endWidth f_width ]
)
=> l_port / nil
```

Description

(Virtuoso Photonics Option) 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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Arguments

<i>o_path</i>	ID of a ccPath object.
<i>t_startPortName</i>	The name of the start port. For example, "opt0".
<i>t_startPortDirection</i>	The direction of the start port. The default is "inputOutput".
<i>t_endPortName</i>	The name of the end port. For example, "opt1". The default is "end".
<i>t_endPortDirection</i>	The direction of the end port. The default is "inputOutput".
<i>?n_portPinWidth n_portPinWidth</i>	Attribute value of the width for the specified photonics pinFig in user units.
<i>?startWidth n_width</i>	When specified, this value allows the user to override the default value that is normally taken from the ccPath corresponding to the start width. It represents the expected optical width for the optical port in contrast to the <code>portPinWidth</code>. The default value is <code>nil</code>, which means that the corresponding width will be taken by the start width of the ccPath.
<i>?endWidth f_width</i>	When specified, this value allows the user to override the default value that is normally taken from the ccPath corresponding to the end width. It represents the expected optical width for the optical port in contrast to the <code>portPinWidth</code>. The default value is <code>nil</code>, which means that the corresponding width will be taken by the end width of the ccPath.

Value Returned

<i>l_port</i>	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.	

Virtuoso Studio Design Environment SKILL Reference

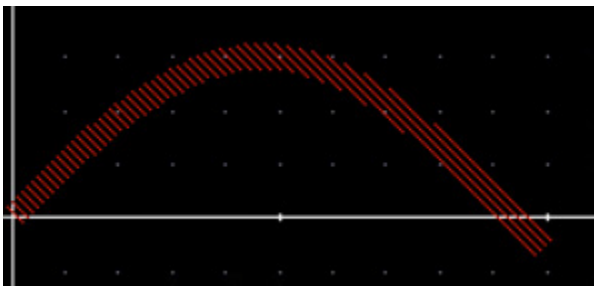
Photonics Functions (Virtuoso Photonics Option)

`nil` The ports are not created, or an error occurs.

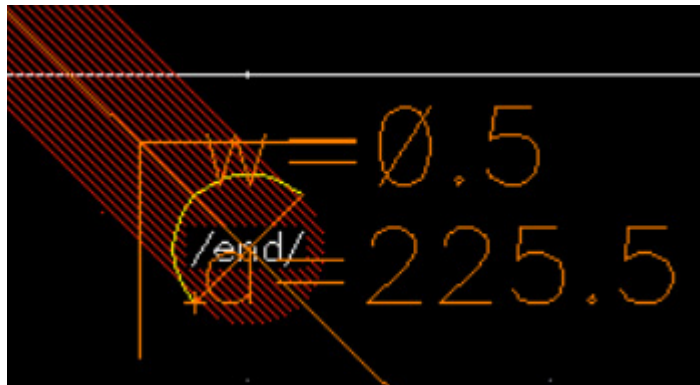
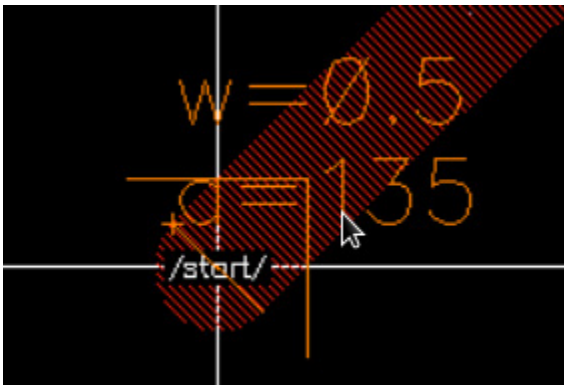
Example

```
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)
```



```
phoAddWaveguidePorts(ccPath "start" "inputOutput" "end" "inputOutput")
(
  (nil xy
    (0.0 0.0) name "start" direction "inputOutput" pinFigId db:0x211fbelb facet
    (nil width 0.5 angle 135.0 radius 0.0)
  )
  (nil xy
    (10.0 -0.572) name "end" direction "inputOutput" pinFigId db:0x211fbelc facet
    (nil width 0.5 angle 225.53 radius -43.31903
    )
  )
)
```



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

Photonics Flow Functions

phoCompose

```
phoCompose (
    l_dbInstID
    l_libCellView
)
=> t_dbInstID / nil
```

Description

(Virtuoso Photonics Option) 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
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
    composite = phoCompose(list(wg1 wg2) list("gopdk" "waveguideComposite"  
"layout"))  
  )
```

Related Topics

[Photonics Flow Functions](#)

phoComputeModeProps

```
phoComputeModeProps (
    o_occPath
    n_waveLength
    n_temperature
    t_mode
    [ valueFlags l_valueFlags ]
    [ alpha n_alpha ]
)
=> l_values / nil
```

Description

(Virtuoso Photonics Option) Computes the path length and average values for the selected mode properties along the center line of the given path

Arguments

<i>o_occPath</i>	A curvy path for modeProp computation.
<i>n_waveLength</i>	Input wavelength parameter. This value should be present in the modeProp table.
<i>n_temperature</i>	Input temperature parameter. This value should be present in the modeProp table.
<i>t_mode</i>	Input modeProp parameters. This value should be present in the modeProp table.
<i>l_valueFlags</i>	List of modeProp values to compute. The default is <code>nil</code> , which means all modeProp values are computed.
<i>alpha n_alpha</i>	The required computation precision. The default is <code>0.1</code> , which is the maximum allowed value. Specifying a value greater than <code>0.1</code> results in an error.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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)
```

Related Topics

[Photonics Flow Functions](#)

phoHiCompose

```
phoHiCompose (
    l_dbInstID
    [ ?compositeMaster l_compositeMaster ]
)
=> d_InstID / nil
```

Description

(Virtuoso Photonics Option) 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

<i>l_dbInstID</i>	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.

<i>?compositeMaster l_compositeMaster</i>	The library cell view that specifies the master of the composite waveguide to be created.

Value Returned

<i>d_InstID</i>	Database instance ID of the composite waveguide created.
<i>nil</i>	The composite waveguide was not created.

Example

Composes three abutted instances into a composite waveguide instance.

```
when(win = hiGetCurrentWindow())
    when(lcv = geGetEditCellView(win)
        i0 = dbFindAnyInstByName(lcv "I0")
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
i1 = dbFindAnyInstByName(lcv "I1")
i2 = dbFindAnyInstByName(lcv "I2")
composite = phoHiCompose(list(i0 i1 i2))
)
)
```

Related Topics

[Photonics Flow Functions](#)

[Composite Waveguide Editor Form](#)

[Composite Waveguides](#)

phoCreatePort

```
phoCreatePort (
    d_cellViewId
    l_lpp
    t_x
    t_y
    t_portDirection
    t_netName
    t_termName
    t_pinName
    n_width
    n_angle
    n_radius
    [ n_portWidth ]
    [ n_portHeight ]
    => d_portShape / nil
```

Description

(Virtuoso Photonics Option) 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.

This port must be on the layer manufacturing grid to enable proper inter-device abutment. You should not use the `xy` attribute of a facet, which is a mathematical point to set these values as you would be using the database rounding and not the CC routing. Pick the start and end point of the `ccPath` corresponding to the facet to set the `x` and `y` coordinates of the port.

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

<code>d_cellViewId</code>	Database ID of a cellview.
<code>l_lpp</code>	Layer-purpose pair for used to create the port. Valid value for layer is <code>waveguide</code> . Valid value for purpose is <code>drawing</code> .
<code>t_x</code>	X-coordinate for the center of the port. This is the start point of the <code>ccPath</code> corresponding to the facet.

Note: A port is drawn as a circle.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

<i>t_y</i>	Y-coordinate for the center of the port. This is the end point of the <code>ccPath</code> corresponding to the facet.
<i>t_portDirection</i>	The direction of the port. The default is "inputOutput".
<i>t_netName</i>	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.
<i>t_termName</i>	Name of the terminal.
<i>t_pinName</i>	Name of the pin. If a pin name is not specified, it is created.
<i>n_width</i>	Width of the optical facet.
<i>n_angle</i>	Angle of the optical facet.
<i>n_radius</i>	Radius of the optical facet.
<i>n_portWidth</i>	Width to draw the physical port (width of the ellipse). Note: If no default width is specified, <code>phoCreatePort</code> uses the <code>minWidth</code> for the specified layer-purpose pair. If the <code>minWidth</code> for the layer-purpose pair is not defined in the technology database, a port of minimum size (2DBU) is created.
<i>n_portHeight</i>	Height to draw the physical port (height of the ellipse). Note: If no default width is specified, <code>phoCreatePort</code> uses the <code>minWidth</code> for the specified layer-purpose pair. If the <code>minWidth</code> for the layer-purpose pair is not defined in the technology database, a port of minimum size (2DBU) is created.

Value Returned

<i>d_portShape</i>	Database ID of the port that is created.
<i>nil</i>	Port was not created.

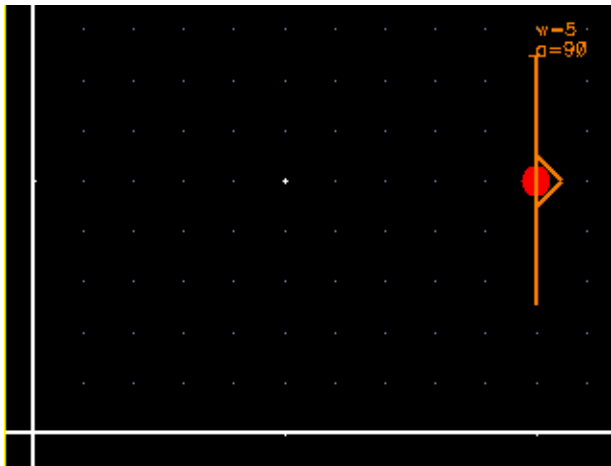
Example

```
phoCreatePort(cv list("waveguide" "drawing") 10.0 5.0 "inputOutput" "Opt1" "Opt1"  
nil 5.0 90.0 0.0)
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

=> db:0x215db89c



Related Topics

[Photonics Flow Functions](#)

phoGetConnectorAnchors

```
phoGetConnectorAnchors(  
    dbInstID  
)  
=> list( l_anchorStartDescription l_anchorEndDescription)
```

Description

(Virtuoso Photonics Option) 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

<i>dbInstID</i>	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

Gets a database pointer to a connector waveguide instance selected in the layout canvas.

```
instId = car(getSelSet())
```

Gets the anchor information for the selected connector instance.

```
phoGetConnectorAnchors(instId)  
(("waveguide" "drawing")
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
(-44.534 12.882) 0.0 0.5 nil
)
(("waveguide" "drawing")
(95.568 7.032) 0.0 0.5 nil
)
)
```

Related Topics

Photonics Flow Functions

phoGenWaveguide

```
phoGenWaveguide (
    o_path
    [ ?excludeLayers l_excludeLayers ]
    [ ?derivedShapeSpecs l_derivedShapeSpecs ]
)
=> l_paths / nil
```

Description

(Virtuoso Photonics Option) 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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Arguments

o_path ID of a ccPath.

l_excludeLayers List of waveguide layer names to exclude.

`list(s_layer...)`

l_derivedShapeSpecs

The object shape to be derived from the master shape. A derived shape definition list has the following format:

```
list((tx_layer | (tx_layer tx_purpose) [['enclosure
    tn_enclosure] | ['offset ['side "both" | "right" |
    "left"] ['inner tn_inner] ['outer tn_outer]]}...)
```

Here:

- *tx_layer*: The layer of the derived shape.
- *tx_purpose*: The corresponding purpose for the layer. If not specified, the value is set as `drawing`.
- The type of the derived shape can be one of the following:
 - *enclosure*: Centers the shape around the master shape based on the *tn_enclosure* value specified. By default, the value is `nil`.
 - *offset*: Offsets the shape from the master shape. The specification includes one or more of these attributes: *side* (direction as `both`, `right`, or `left`), *inner* (offset of the inner edge of the derived shape from the centerline of the master shape), and *outer* (offset of the outer edge of the derived shape from the edge of the master shape). The default values are `both`, `nil`, and `nil`.

Value Returned

l_shapes List of the generated ccShape IDs.

`list(o_shape...)`

`nil` An error occurred.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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
```

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](#)

[Photonics Flow Functions](#)

phoHiDecompose

```
phoHiDecompose (
    l_dbCompWaveInstID
)
=> l_dbInstID / nil
```

Description

(Virtuoso Photonics Option) 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

<i>l_dbCompWaveInstID</i>	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

<i>l_dbInstID</i>	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.
<i>nil</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
car(insts)~>w = "2u"  
phoHiCompose(insts)  
)  
)
```

Related Topics

[Composite Waveguide Editor Form](#)

[Composite Waveguides](#)

[Photonics Flow Functions](#)

phoHiEditCompositeWG

```
phoHiEditCompositeWG(  
    )
```

Description

(Virtuoso Photonics Option) 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 Form](#)

[Composite Waveguides](#)

[Photonics Flow Functions](#)

phoIsWaveguide

```
phoIsWaveguide(  
    [ ?masterID d_masterID ]  
    [ ?lcv l_lcv ]  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) 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 specified 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](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Photonics Flow Functions

phoTechDeleteModePropDataTable

```
phoTechDeleteModePropDataTable(  
    d_techDB  
    t_waveguideDefName  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Deletes the modeProp table for the specified waveguideDef from the given technology database.

Arguments

<i>d_techDB</i>	Database ID of the technology database in which the modeProp table is searched.

<i>t_waveguideDefName</i>	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
```

Related Topics

Photonics Flow Functions

phoTechGetModePropDataTableParamValues

```
phoTechGetModePropDataTableParamValues (  
    d_techDB  
    t_waveguideDefName  
    S_param  
)  
=> l_values / nil
```

Description

(Virtuoso Photonics Option) Returns the list of values specified in the modeProp table for the given parameter.

Arguments

<i>d_techDB</i>	ID of the technology database storing the modeProp table.
<i>t_waveguideDefName</i>	Name of the waveguideDef.
<i>S_param</i>	Name of the parameter. Valid values are 'wavelength, 'width, 'curvature, 'temperature and 'mode, or the value of corresponding string.

Value Returned

<i>l_values</i>	$(n\_value...) \mid (t\_value...)$ A list of numeric values (<i>n_value</i>) that are valid for all parameters other than 'mode or a list of strings (<i>t_value</i>) that are valid for 'mode.
<i>nil</i>	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")
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

Photonics Flow Functions

phoTechHasModeProp

```
phoTechHasModeProp(  
    d_techDB  
    t_waveguideDefName  
    s_param  
    g_value  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) 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.

Arguments

<i>d_techDB</i>	ID of the technology database in which the search is performed.
<i>t_waveguideDefName</i>	Name of the waveguideDef.
<i>s_param</i>	modeProp parameter name for which the technology database is searched.
<i>g_value</i>	modeProp parameter value for which the technology database is searched.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

<code>t</code>	The specified parameter value was found in the modeProp table for the specified waveguide.
<code>nil</code>	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
```

Related Topics

[Photonics Flow Functions](#)

phoTechHasModePropDataTable

```
phoTechHasModePropDataTable(  
    d_techDB  
    t_waveguideDefName  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) 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

<i>d_techDB</i>	ID of the technology database in which search is performed.
<i>t_waveguideDefName</i>	Name of the waveguideDef.

Value Returned

<i>t</i>	The modeProp table for the given waveguideDef name was found.
<i>nil</i>	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
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

Photonics Flow Functions

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

<i>d_techDB</i>	ID of the technology database in which search is performed.
<i>t_waveguideDefName</i>	Name of the <code>waveguideDef</code> .

Value Returned

<i>t</i>	The <code>localOnly</code> flag is set.
<i>nil</i>	The <code>localOnly</code> flag is not set, the <code>modeProp</code> 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
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
; 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")
```

Related Topics

Photonics Flow Functions

phoTechLoadModePropDataTable

```
phoTechLoadModePropDataTable(  
    d_techID  
    t_path(  
        [ ?waveguideDefName t_waveguideDefName ]  
        [ ?localOnly g_localOnly ]  
        [ ?overwrite g_overwrite ]  
    )  
=> t / nil
```

Description

(Virtuoso Photonics Option) 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

<i>d_techID</i>	ID of the technology database file in which the specified waveguideDef is stored.
<i>t_path</i>	Path to the data file.
<i>?localOnly g_localOnly</i>	A boolean parameter indicating how modeProp table is retrieved. ■ When set to <code>t</code>, the contents of the specified modeProp table is retrieved only from the local technology database. This is the default. ■ When set to <code>nil</code>, 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 <code>localOnly</code> flag. Once set, this flag permanently affects the modeProp table contents exposed to the modeProp APIs. Note: This default value <code>t</code> is in effect for the modeProp tables created by the older versions of this API.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?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.

`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

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

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

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 not succeed.

Related Topics

[Photonics Flow Functions](#)

phoRegisterInstTermOpticalMonitorMenuItem

```
phoRegisterInstTermOpticalMonitorMenuItem(  
    t_menuItemText  
    s_callback  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Registers the callback function to add an optical monitor to the Virtuoso Schematic Editor contextual menu. Optical monitors are used to setup a simulation and plot for complex mathematical optical signals. This function can be called at any time in the code. However, calling it several times with the same name string overrides the previously stored callback for that menu item and causes a warning to be displayed in the CIW.

Arguments

<i>t_menuItemText</i>	Name of the menu option for the optical monitor.
<i>s_callback</i>	Name of the callback function to register the optical monitor menu item.

Value Returned

<i>t</i>	Optical monitor callback function is registered.
<i>nil</i>	Optical monitor callback is not registered.

Example

The example below registers optical monitor for instTerm.

```
phoRegisterInstTermOpticalMonitorMenuItem("OM Incident and Phase"  
'OMExpressionBuilderINCB )  
  
phoRegisterInstTermOpticalMonitorMenuItem("OM Incident and Phase"  
'OMExpressionBuilderOUTCB )
```

Related Topics

[Photonics Flow Functions](#)

phoOMGetObject

```
phoOMGetObject(  
    )  
=> l_list / nil
```

Description

(Virtuoso Photonics Option) Retrieves the list of information required to register an optical monitor menu item in the Virtuoso Schematic Editor contextual menu. The function call must be made inside the optical monitor, otherwise the information returned is not reliable.

Arguments

None

Value Returned

l_list

Returns a list of the following elements:

- Database ID of the instance from which the port was selected.
Note: This is the local ID. Adjust it for the hierarchy before using it to create the expression.
- Database ID of the instance pin.
- Information regarding the ADE session connected to the current schematic.
Note: This information is used to locate the test to which the optical monitor callback expressions will be added.
- Name of the current test to which the expressions will be added.

nil

Virtuoso Photonics option is not available.

Example

```
phoOMGetObject()
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

Photonics Flow Functions

Photonics Symbol Generator Functions

The optional SKILL generators speed up the generation of consistent symbols for photonics PDK. They can be used to generate a regular symbol cellview with a single call instead of drawing the symbol. They can also be wrapped into a SKILL Pcell shell, which allows support for magnification.

These generators check whether the Virtuoso Photonics Option is enabled. For more information on starting Virtuoso with photonics enabled, refer to

Ports for these symbols are typed "optical" to enable optical signal handling in Virtuoso Schematic Editor.

All the functions allow the PDK developer to specify the name of the optical (and electrical, if relevant) ports and direction.

Related Topics

[Launching Virtuoso in Photonics Mode](#)

[Symbol Pcell Example](#)

Symbol Pcell Example

Description

This is an example of wrapping the basic generator to create a parameterized symbol. The only parameter supported here is magnification.

```
pcDefinePCell(

    list( ddGetObj("gopdk")
        "wgWaypointConnector"
        "symbol"
        "schematicSymbol"
    )

    ; define the PCell parameters
    (
        (mag float 1.0)
    )

    prog( ()

        unless( isCallable('phoDrawWayPointConnectorSymbol)
            return()
        )

        ; Call the symbol generator
        phoDrawWayPointConnectorSymbol( pcCellView mag
                                         ?startPortName "optS"
                                         ?endPortName  "optE"
                                         )

        return(t)
    )
)
```

Related Topics

[Launching Virtuoso in Photonics Mode](#)

[Photonics Symbol Generator Functions](#)

phoDrawBendConnectorSymbol

```
phoDrawBendConnectorSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a bend connector symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the start port. The default is "start".
<i>?startPortDirection s_direction</i>	Direction of the start port. The default is "inputOutput".
<i>?endPortName s_PortName</i>	Name of the end port. The default is "end".
<i>?endPortDirection s_direction</i>	Direction of the end port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

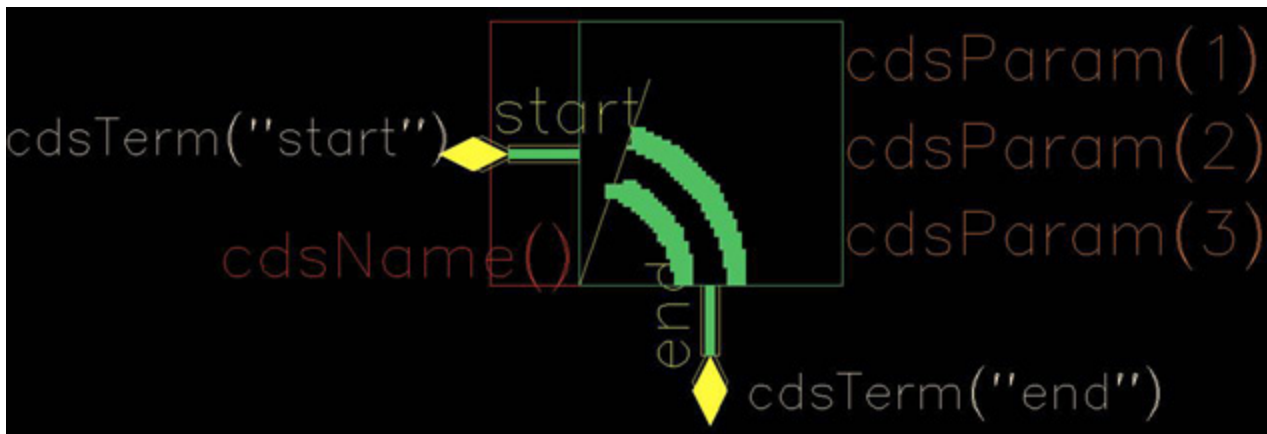
Photonics Functions (Virtuoso Photonics Option)

Return Value

<code>t</code>	Symbol for the bend connector is generated.
<code>nil</code>	Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawBendConnectorSymbol(cv 1.0)
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawCWLaserSymbol

```
phoDrawCWLaserSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a laser symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
?startPortName <i>s_PortName</i>	Name of the start port. The default is "opt0".
?startPortDirection <i>s_direction</i>	Direction of the start port. The default is "inputOutput".

Return Value

t	Laser symbol is generated.
nil	Symbol is not generated.

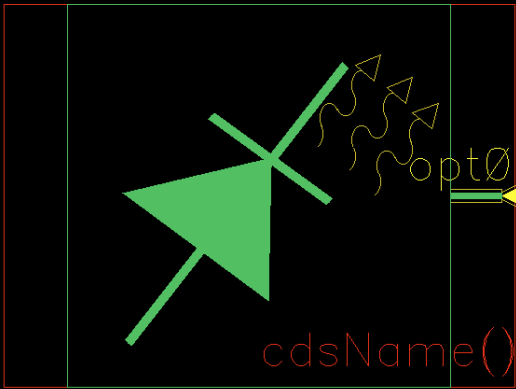
Example

```
cv = geGetEditCellView()  
phoDrawCWLaserSymbol(cv 1.0)
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
cdsParam(1)
cdsParam(2)
cdsParam(3)
cdsParam(4)
```



```
cdsName()
```

```
cdsTerm("opt0")
```

Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawCompWaveguideSymbol

```
phoDrawCompWaveguideSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a composite waveguide symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the start port. The default is "in0".
<i>?startPortDirection s_direction</i>	Direction of the start port. The default is "inputOutput".
<i>?endPortName s_PortName</i>	Name of the end port. The default is "out0".
<i>?endPortDirection s_direction</i>	Direction of the end port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

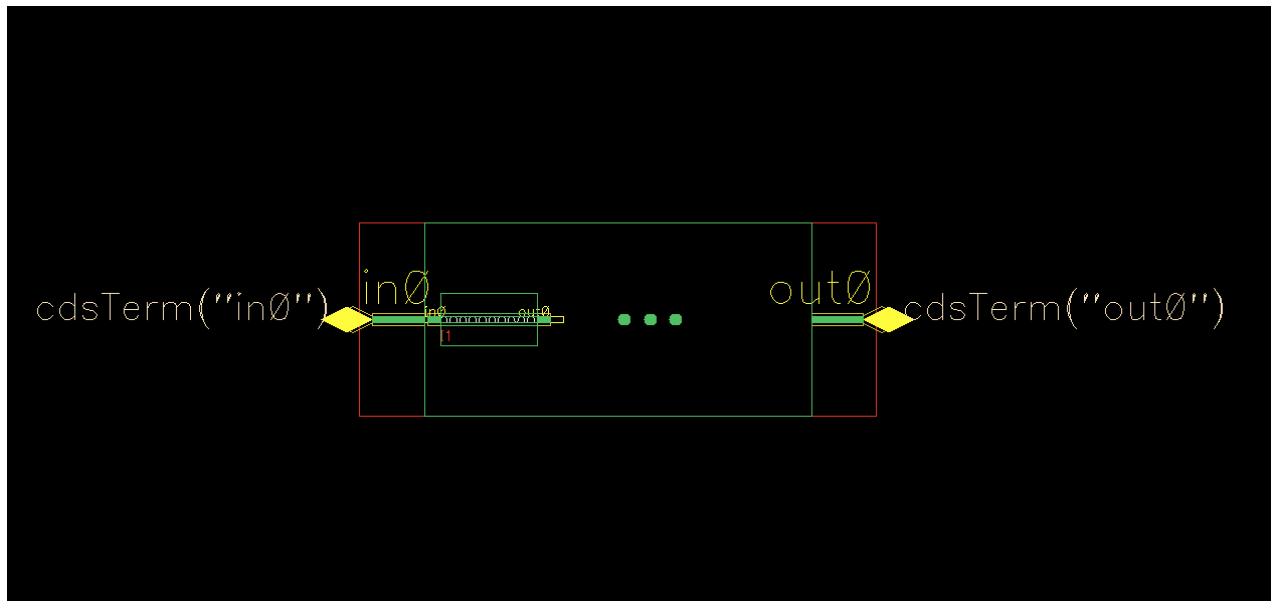
Photonics Functions (Virtuoso Photonics Option)

Return Value

t	Symbol for the composite waveguide is generated.
nil	Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawCompWaveguideSymbol(cv 1.0)
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawCrossBBSymbol

```
phoDrawCrossBBSymbol(  
    d_cellViewId  
    n_magnification  
    ?northPortName s_PortName  
    ?northPortDirection s_direction  
    ?southPortName s_PortName  
    ?southPortDirection s_direction  
    ?eastPortName s_PortName  
    ?eastPortDirection s_direction  
    ?westPortName s_PortName  
    ?westPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a cross building block symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?northPortName s_PortName</i>	Name of the north port. The default is "opt0".
<i>?northPortDirection s_direction</i>	Direction of the north port. The default is "inputOutput".
<i>?southPortName s_PortName</i>	Name of the south port. The default is "opt2".
<i>?southPortDirection s_direction</i>	Direction of the south port. The default is "inputOutput".
<i>?eastPortName s_PortName</i>	Name of the east port. The default is "opt1".
<i>?eastPortDirection s_direction</i>	Direction of the east port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?westPortName *s_PortName*

Name of the west port. The default is "opt3".

?westPortDirection *s_direction*

Direction of the west port. The default is "inputOutput".

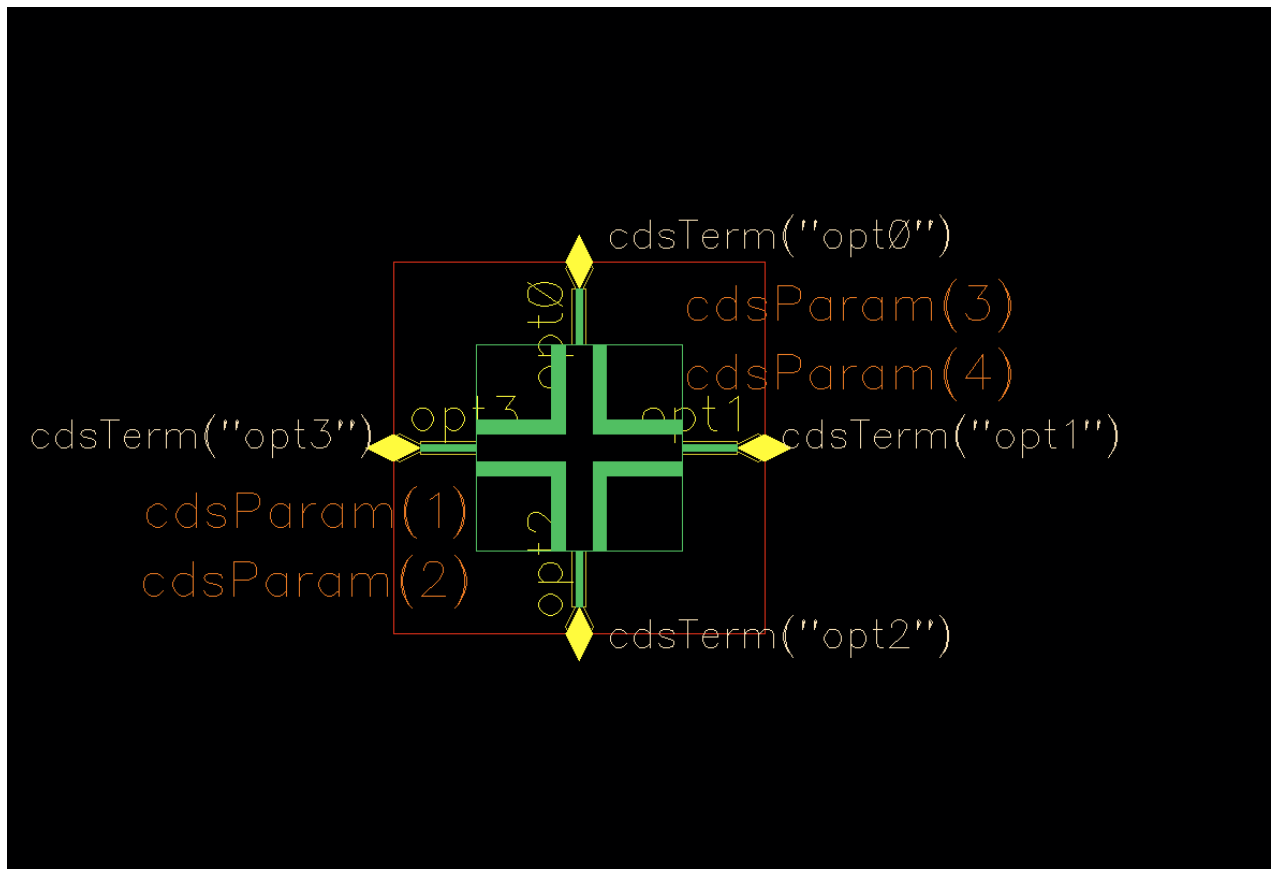
Return Value

t Symbol for the cross building block is generated.

nil Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawCrossBBSymbol(cv 1.0)
```



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawCurveDeltaConnectorSymbol

```
phoDrawCurveDeltaConnectorSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a curved delta connector symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
?startPortName <i>s_PortName</i>	Name of the start port. The default is "start".
?startPortDirection <i>s_direction</i>	Direction of the start port. The default is "inputOutput".
?endPortName <i>s_PortName</i>	Name of the end port. The default is "end".
?endPortDirection <i>s_direction</i>	Direction of the end port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

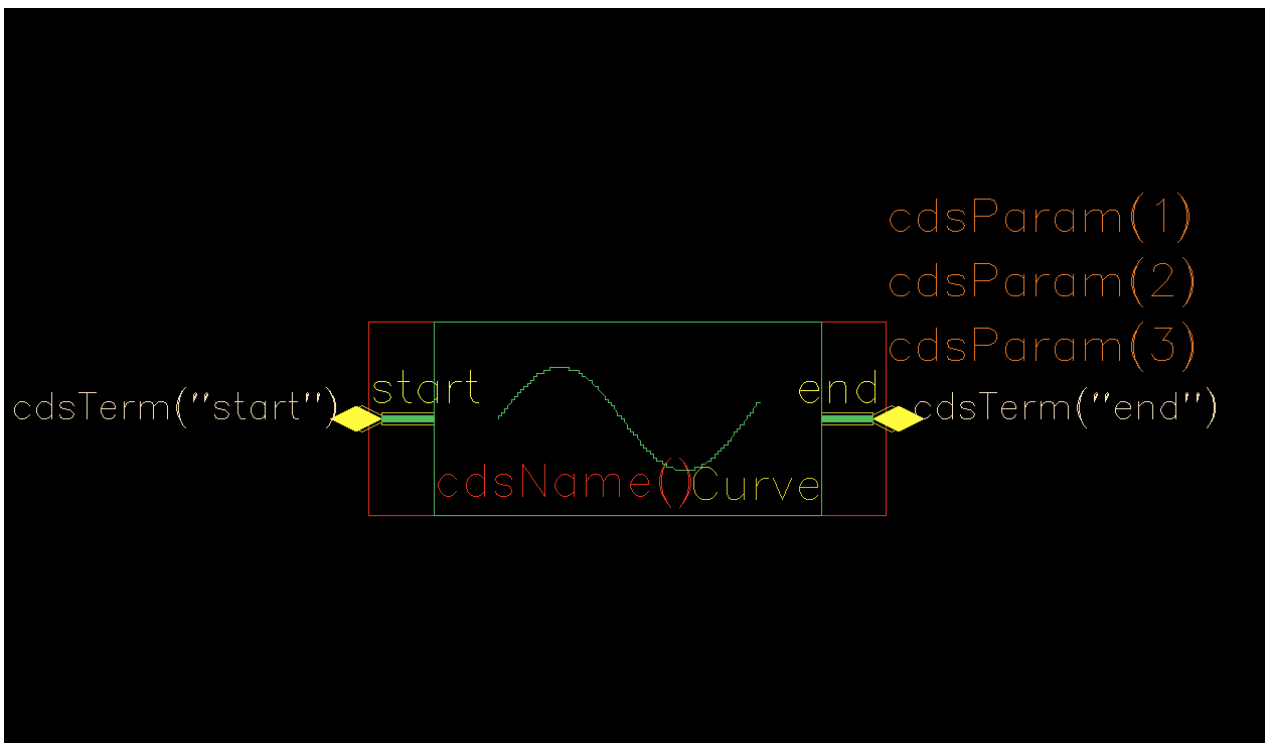
Photonics Functions (Virtuoso Photonics Option)

Return Value

<code>t</code>	Symbol for the curved delta connector is generated.
<code>nil</code>	Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawCurveDeltaConnectorSymbol(cv 1.0)
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawDCSymbol

```
phoDrawDCSymbol(  
    d_cellViewId  
    n_magnification  
    ?addPortName s_PortName  
    ?addPortDirection s_direction  
    ?dropPortName s_PortName  
    ?dropPortDirection s_direction  
    ?inPortName s_PortName  
    ?inPortDirection s_direction  
    ?thruPortName s_PortName  
    ?thruPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a directional coupler symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?addPortName s_PortName</i>	Name of the add port. The default is "add".
<i>?addPortDirection s_direction</i>	Direction of the add port. The default is "inputOutput".
<i>?dropPortName s_PortName</i>	Name of the drop port. The default is "drop".
<i>?dropPortDirection s_direction</i>	Direction of the drop port. The default is "inputOutput".
<i>?inPortName s_PortName</i>	Name of the in port. The default is "in".
<i>?inPortDirection s_direction</i>	Direction of the in port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?thruPortName *s_PortName*

Name of the thru port. The default is "thru".

?thruPortDirection *s_direction*

Direction of the thru port. The default is "inputOutput".

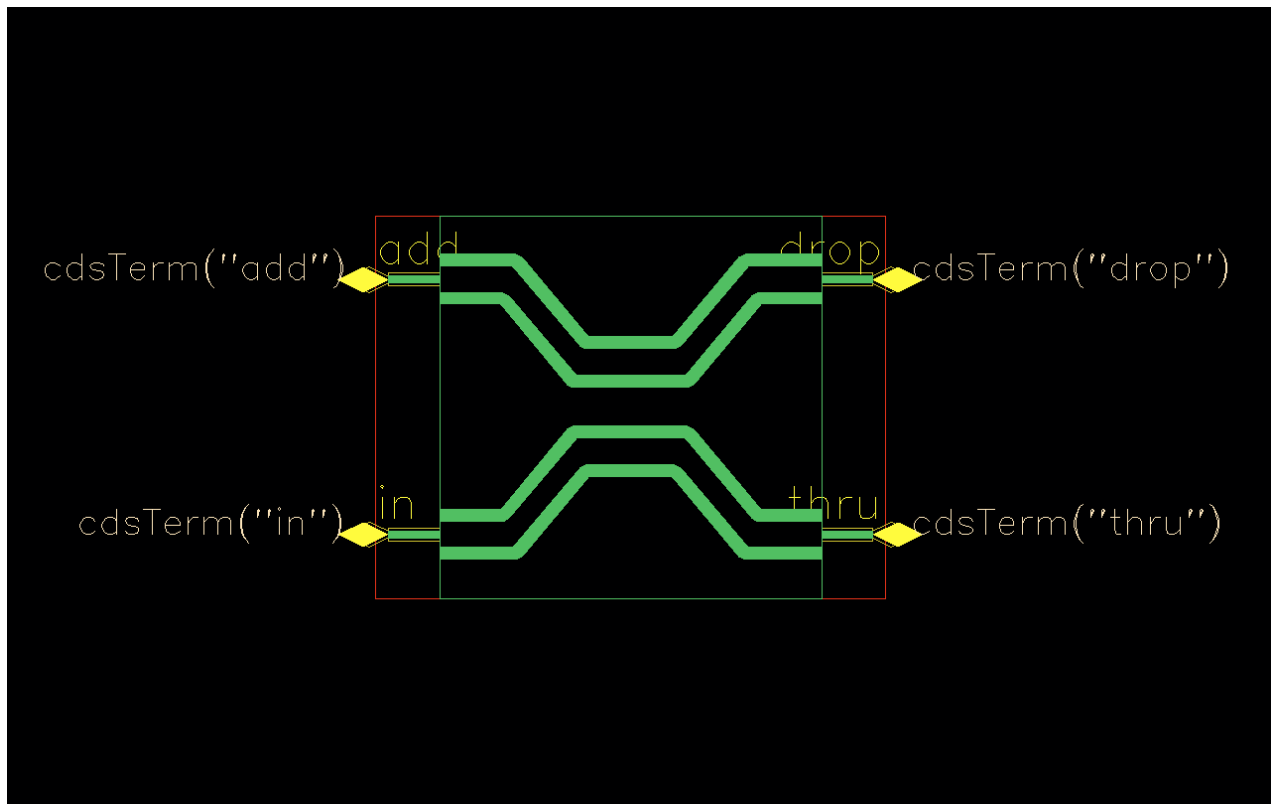
Return Value

t Symbol for the directional coupler is generated.

nil Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawDCSymbol(cv 1.0)
```



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawEdgeCouplerBBSymbol

```
phoDrawEdgeCouplerBBSymbol(  
    d_cellViewId  
    n_magnification  
    ?edgePortName s_PortName  
    ?edgePortDirection s_direction  
    ?outPortName s_PortName  
    ?outPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws an optical edge coupler building block symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a writable schematic symbol cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?edgePortName s_PortName</i>	Name of the edge port. The default is "fiber".
<i>?edgePortDirection s_direction</i>	Direction of the edge port. The default is "inputOutput".
<i>?outPortName s_PortName</i>	Name of the drop port. The default is "out".
<i>?outPortDirection s_direction</i>	Direction of the out port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

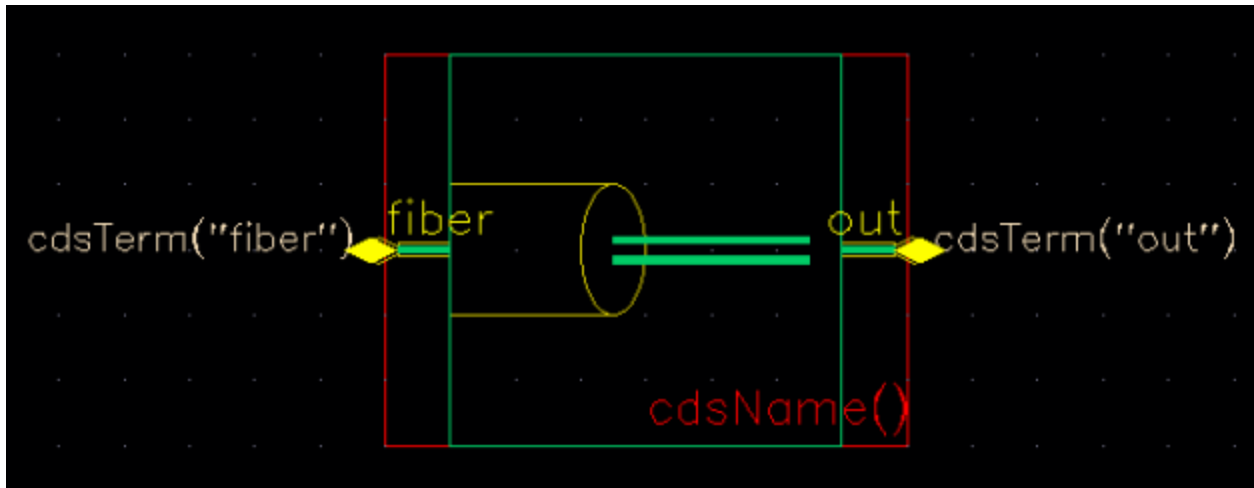
Photonics Functions (Virtuoso Photonics Option)

Value Returned

t	Symbol for the optical edge coupler building block is generated.
nil	Symbol is not generated.

Example

```
cv = dbOpenCellViewByType("gopdkX" "edgeCouplerExample" "symbol" "schematicSymbol"  
"w")  
phoDrawEdgeCouplerBBSymbol(cv 1.0 )
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawMMISymbol

```
phoDrawMMISymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?startPortNum i_num  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
    ?endPortNum  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a multimode interferometer symbol. You can define the number of ports on both sides.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the starting port. The default is "optS".
<i>?startPortDirection s_direction</i>	Direction of the starting port. The default is "inputOutput".
<i>?startPortNum i_num</i>	Number of the starting port. The default is "1".
<i>?endPortName s_PortName</i>	Name of the ending port. The default is "optE".
<i>?endPortDirection s_direction</i>	Direction of the ending port. The default is "inputOutput".
<i>?endPortNum i_num</i>	Number of the ending port. The default is "2".

Virtuoso Studio Design Environment SKILL Reference

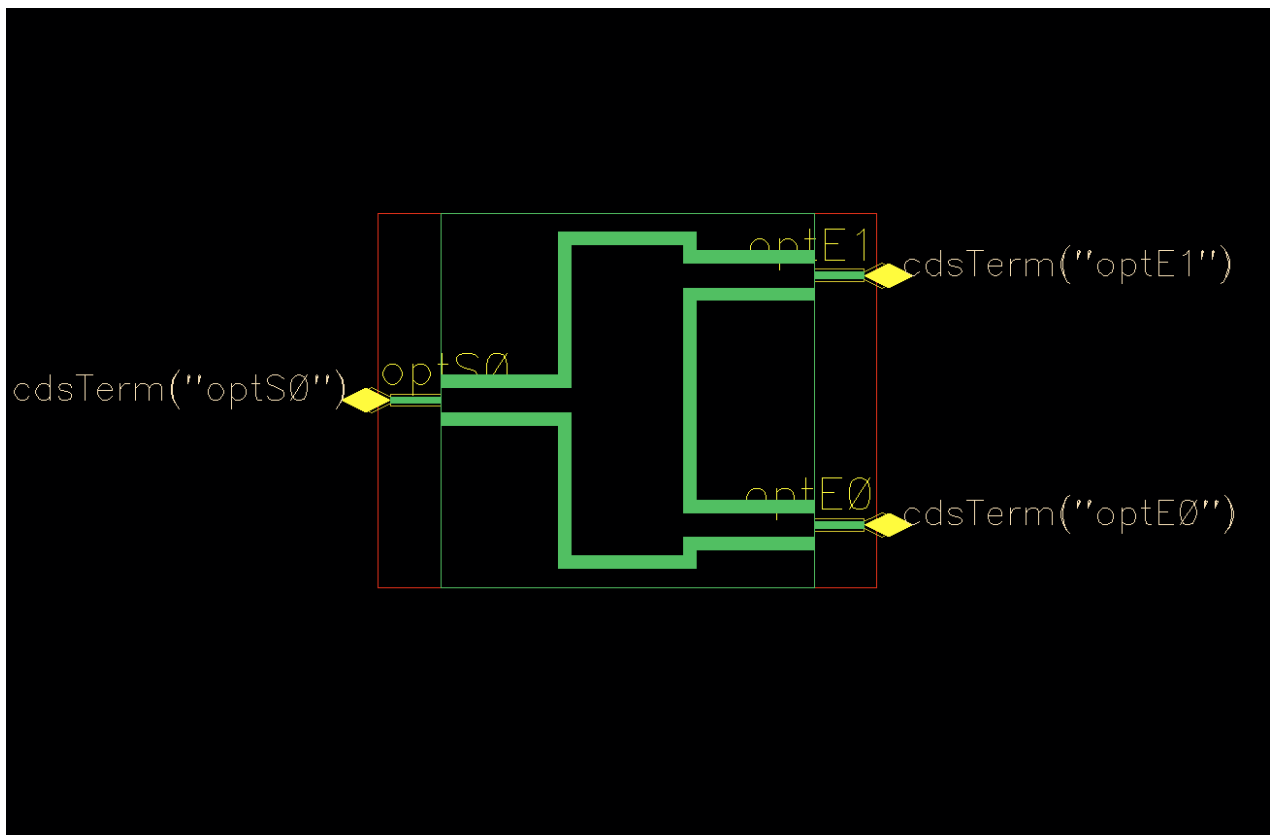
Photonics Functions (Virtuoso Photonics Option)

Return Value

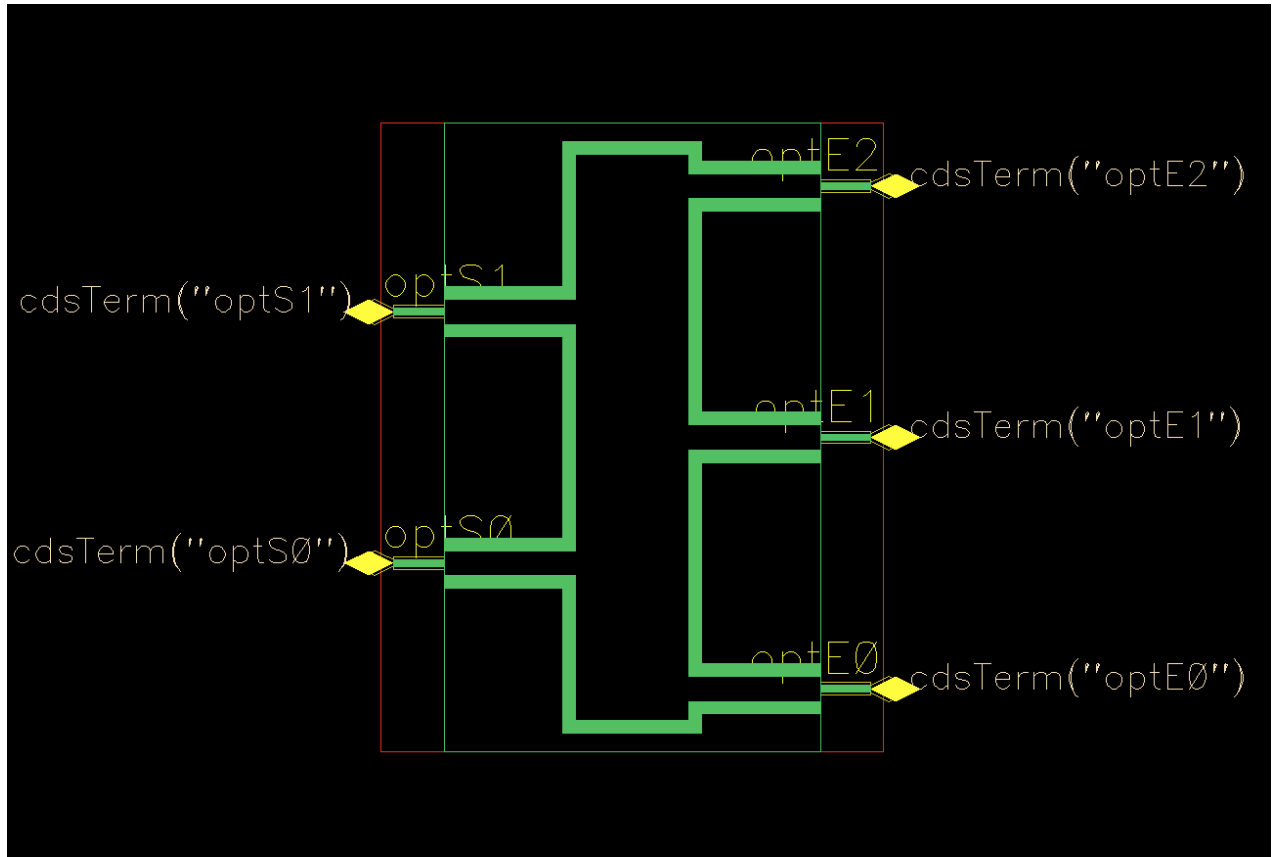
<code>t</code>	Symbol for the multimode interferometer is generated.
<code>nil</code>	Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawMMISymbol(cv 1.0)
```



```
cv = geGetEditCellView()  
phoDrawMMISymbol(cv 1.0 ?startPortNum 2 ?endPortNum 3)
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawPhotoDiodeBBSymbol

```
phoDrawPhotoDiodeBBSymbol(  
    d_cellViewId  
    n_magnification  
    ?Port1Name s_PortName  
    ?Port1Direction s_direction  
    ?cathodeName s_name  
    ?cathodeDirection s_direction  
    ?anodeName s_name  
    ?anodeDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a photo diode symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?Port1Name s_PortName</i>	Name of the optical port. The default is "opt1".
<i>?Port1Direction s_direction</i>	Direction of the optical port. The default is "inputOutput".
<i>?cathodeName s_Name</i>	Name of the cathode. The default is "cathode".
<i>?cathodeDirection s_direction</i>	Direction of the cathode. The default is "inputOutput".
<i>?anodeName s_Name</i>	Name of the anode. The default is "anode".
<i>?anodeDirection s_direction</i>	Direction of the anode. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

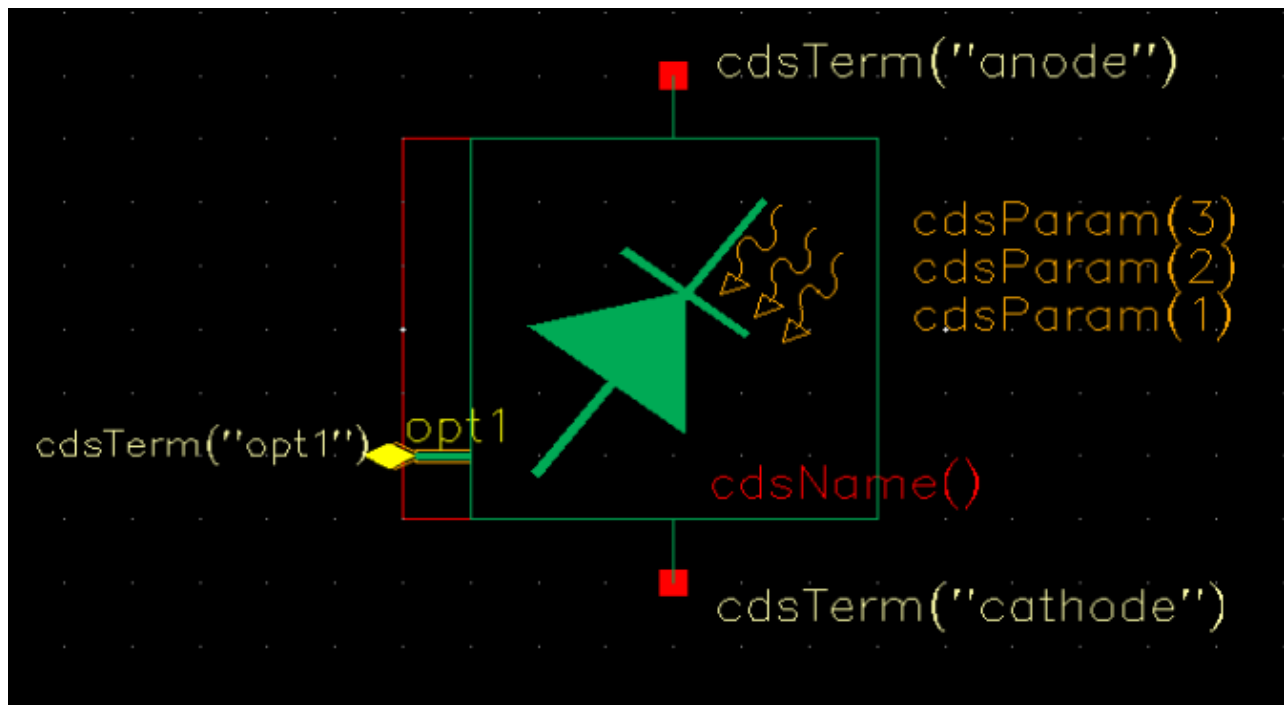
Photonics Functions (Virtuoso Photonics Option)

Return Value

t	Symbol for the photo diode is generated.
nil	Symbol is not generated.

Example

```
cv = dbOpenCellViewByType("test" "myPhotoDiode" "symbol" "schematicSymbol" "w")
phoDrawPhotoDiodeBBSymbol(cv 1.0 ?port1Name "RXOpt")
dbSave(cv)
dbClose(cv)
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawSBendWaveguideSymbol

```
phoDrawSBendWaveguideSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a bend waveguide symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the start port. The default is "opt0".
<i>?startPortDirection s_direction</i>	Direction of the start port. The default is "inputOutput".
<i>?endPortName s_PortName</i>	Name of the end port. The default is "opt1".
<i>?endPortDirection s_direction</i>	Direction of the end port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

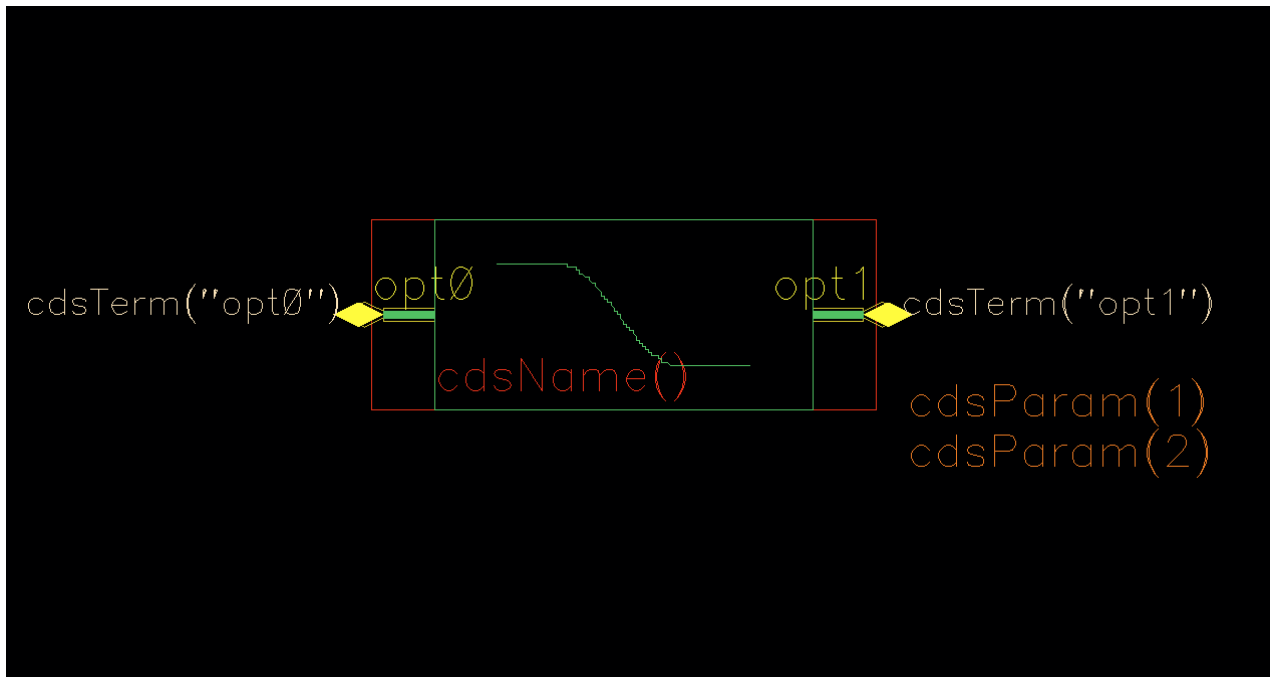
Photonics Functions (Virtuoso Photonics Option)

Return Value

<code>t</code>	Symbol for the bend waveguide is generated.
<code>nil</code>	Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawSBendWaveguideSymbol(cv 1.0)
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawStraightModulatorSymbol

```
phoDrawStraightModulatorSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
    ?cathodeName s_name  
    ?cathodeDirection s_direction  
    ?anodeName s_name  
    ?anodeDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a straight modulator symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the starting port. The default is "opt0".
<i>?startPortDirection s_direction</i>	Direction of the starting port. The default is "inputOutput".
<i>?endPortName s_PortName</i>	Name of the ending port. The default is "opt1".
<i>?endPortDirection s_direction</i>	Direction of the ending port. The default is "inputOutput".
<i>?cathodeName s_Name</i>	Name of the cathode. The default is "CA".
<i>?cathodeDirection s_direction</i>	Direction of the cathode. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?anodeName *s_Name*

Name of the anode. The default is "AN".

?anodeDirection *s_direction*

Direction of the anode. The default is "inputOutput".

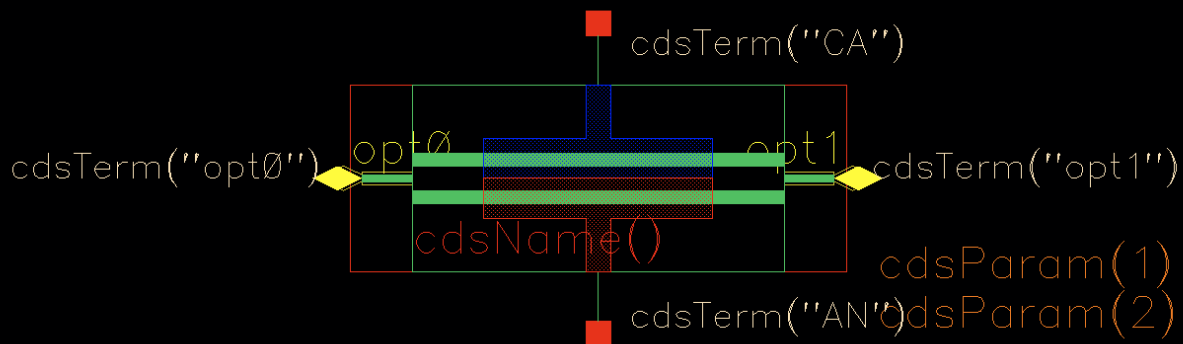
Return Value

t Symbol for the straight modulator is generated.

nil Symbol is not generated.

Example

```
cv = geGetEditCellView()
phoDrawStraightModulatorSymbol(cv 1.0)
```



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawStraightWaveguideSymbol

```
phoDrawStraightWaveguideSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
    ?thermalTerm1Name s_name  
    ?thermalTerm1Direction s_direction  
    ?thermalTerm2Name s_name  
    ?thermalTerm2Direction s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a straight waveguide symbol. You can add an optional thermal device to the display and ports of the symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the starting port. The default is "opt0".
<i>?startPortDirection s_direction</i>	Direction of the starting port. The default is "inputOutput".
<i>?endPortName s_PortName</i>	Name of the ending port. The default is "opt1".
<i>?endPortDirection s_direction</i>	Direction of the ending port. The default is "inputOutput".
<i>?thermalTerm1Name s_Name</i>	Name of the thermal device. The default is "nil".
<i>?thermalTerm1Direction s_direction</i>	

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Direction of the thermal device. The default is "inputOutput".

?thermalTerm2Name *s_Name*

Name of the thermal device. The default is "nil".

?thermalTerm2Direction *s_direction*

Direction of the thermal device. The default is "inputOutput".

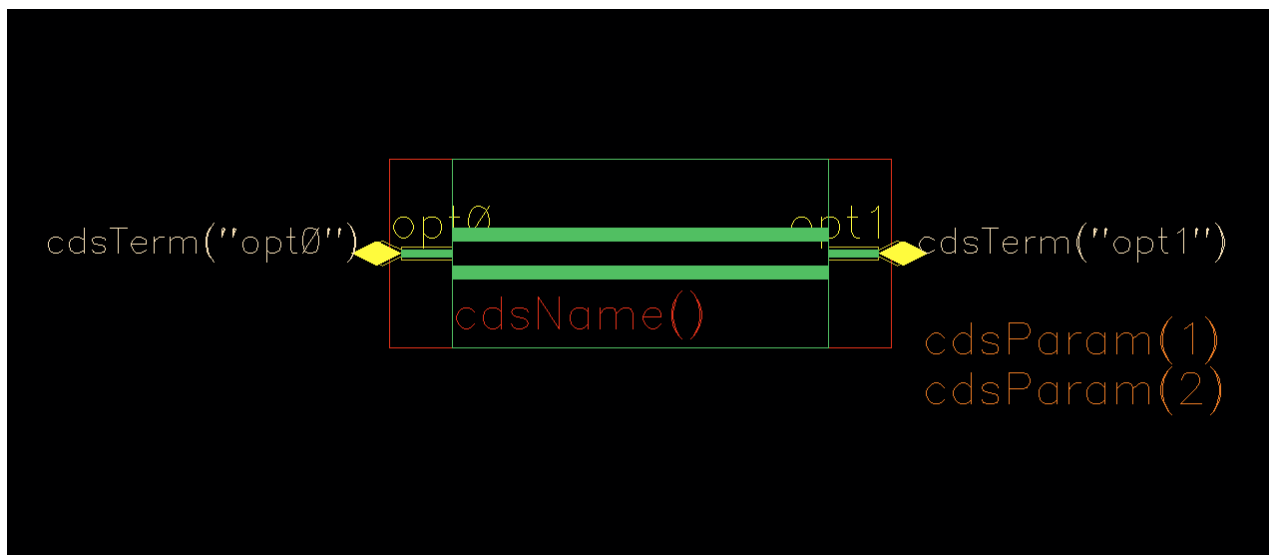
Return Value

t Symbol for the straight waveguide is generated.

nil Symbol is not generated.

Example

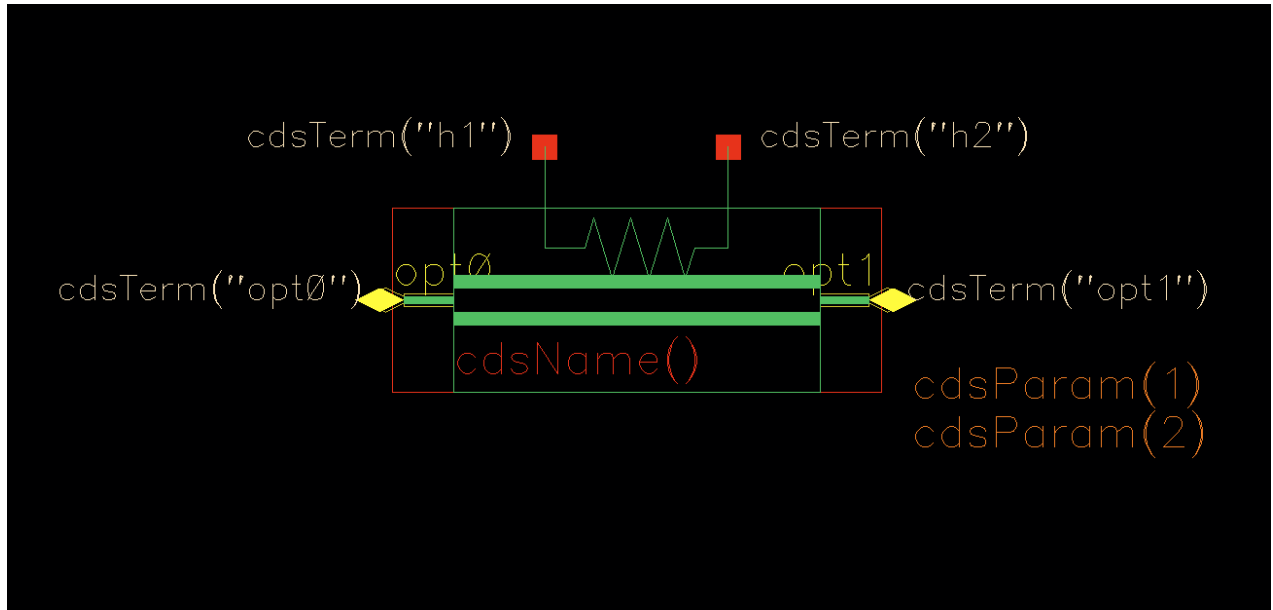
```
cv = geGetEditCellView()
phoDrawStraightWaveguideSymbol(cv 1.0)
```



```
cv = geGetEditCellView()
phoDrawStraightWaveguideSymbol(cv 1.
    ?thermalTerm1Name "T1" ?thermalTerm2Name "T2")
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawWayPointConnectorSymbol

```
phoDrawWayPointConnectorSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName s_PortName  
    ?endPortDirection s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a waypoint connector symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the start port. The default is "opt0".
<i>?startPortDirection s_direction</i>	Direction of the start port. The default is "inputOutput".
<i>?endPortName s_PortName</i>	Name of the end port. The default is "opt1".
<i>?endPortDirection s_direction</i>	Direction of the end port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

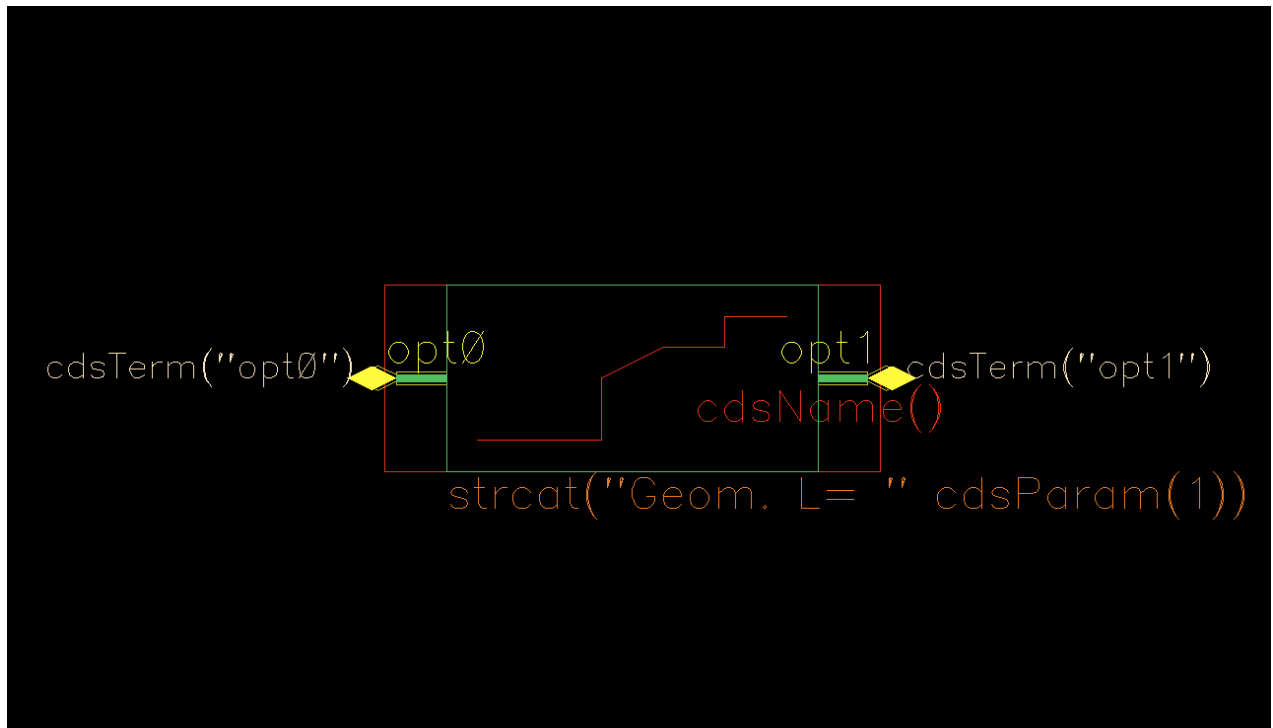
Photonics Functions (Virtuoso Photonics Option)

Return Value

<code>t</code>	Symbol for the waypoint connector is generated.
<code>nil</code>	Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawWayPointConnectorSymbol(cv 1.0)
```



Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

phoDrawYSplitterSymbol

```
phoDrawYSplitterSymbol(  
    d_cellViewId  
    n_magnification  
    ?startPortName s_PortName  
    ?startPortDirection s_direction  
    ?endPortName1 s_PortName  
    ?endPortDirection1 s_direction  
    ?endPortName2 s_PortName  
    ?endPortDirection2 s_direction  
)  
=> t / nil
```

Description

(Virtuoso Photonics Option) Draws a Y splitter symbol.

Arguments

<i>d_cellViewId</i>	Database ID of a cellview.
<i>n_magnification</i>	Database magnification value to set on the instance. Default: 1.0
<i>?startPortName s_PortName</i>	Name of the start port. The default is "in0".
<i>?startPortDirection s_direction</i>	Direction of the start port. The default is "inputOutput".
<i>?endPortName1 s_PortName</i>	Name of the end port. The default is "out0".
<i>?endPortDirection1 s_direction</i>	Direction of the end port. The default is "inputOutput".
<i>?endPortName2 s_PortName</i>	Name of the start port. The default is "out1".
<i>?endPortDirection2 s_direction</i>	Direction of the end port. The default is "inputOutput".

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

t	Symbol for the Y splitter is generated.
nil	Symbol is not generated.

Example

```
cv = geGetEditCellView()  
phoDrawBendConnectorSymbol(cv 1.0)
```

Related Topics

[Photonics Symbol Generator Functions](#)

[Symbol Pcell Example](#)

Photonics Connector Functions

Photonics connectors are a type of generator functions that speed up the generation of simple layouts for a photonics PDK.

Related Topics

[phoBendConnector](#)

[phoCurveDeltaConnector](#)

[phoDrawCompWaveguideLayout](#)

[phoSineDeltaConnector](#)

[phoStraightConnector](#)

[phoWayPointConnector](#)

phoBendConnector

```
phoBendConnector(  
    d_cellViewId  
    t_waveguideLayer  
    [ ?rotation n_rotation  
    [ ?offset l_offset  
    [ ?full g_full  
    [ ?startPortName t_PortName  
    [ ?endPortName t_PortName ]  
    [ ?startPortDirection t_Direction ]  
    [ ?endPortDirection t_Direction ]  
    [ ?width f_width ]  
    [ ?taperStyle s_taperStyle ]  
    [ ?startWidth f_width ]  
    [ ?endWidth f_width ]  
    [ ?taperStartClamp n_length ]  
    [ ?taperEndClamp n_length ]  
    [ ?angle f_angle ]  
    [ ?radius f_radius ]  
    [ ?arcPart f_arcPart ]  
    [ ?Rmin f_Rmin ]  
    [ ?modeSpec l_l_modeSpec ]  
    [ ?modePropOnly g_modePropOnly ]  
    [ ?portPinPurpose t_portPinPurpose ]  
    [ ?waveguideDefPurpose t_waveguideDefPurpose ]  
)  
=> l_dpl / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a bend connector that connects two port points on the same circle with angles equal to the angles of the tangents on that circle. The bend connector is described by the bend radius of the circle and the bend angle of the arc of that circle. In its simplest form, a bend connector consists of a single circular arc segment.

Arguments

<i>d_cellViewId</i>	Cellview ID of the layout to be used. For a connector, the generator checks its view type and whether the cellview is editable and is a maskLayout.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`t_waveguideLayer` Name of the abstract layer or the design intent layer in which you want to draw the connector. This must be a layer with the `waveguide` function defined in the PDK technology file. It is also used to determine the relevant `waveguideDef` (cross-section) in the technology file, to generate the needed layers to implement the waveguide.

`?rotation n_rotation`

Rotation angle in degrees for the bend connector.

The default value is `(0,0)`, which means there is no rotation.

The facet angle `0.0` points to the bottom.

`?offset l_offset list(n_lhs n_rhs)`

Offset value for the connector.

The default value is `(0,0)`, which means there is no offset.

`?full g_full`

A Boolean when set to `t` specifies whether a waveguide is created with all its layers from the cross section as defined in the technology file. This is the default.

When set to `nil`, only the design intent layer `t_waveguideLayer` is drawn, generating an abstract of the connector.

`?startPortName t_PortName`

Name of the start port. The default value is `"start"`.

A `nil` port name results in a connector generated without a port at the start. This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?endPortName t_PortName`

Name of the end port. The default value is `"end"`.

This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?startPortDirection t_Direction`

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

	Direction of the start port. The default value is "inputOutput".
<code>?endPortDirection t_Direction</code>	
	Direction of the end port. The default is "inputOutput".
<code>?width n_width</code>	Width of the waveguide layer. The default value is <code>nil</code> .
<code>?taperStyle s_taperStyle</code>	
	Style of the tapering.
	Valid values are: 'none, 'linear, 'exp and 'parabolic. The default value is 'none.
<code>?startWidth n_width</code>	
	Width of the taper at the start of the connector. The default value is <code>nil</code> .
<code>?endWidth f_width</code>	Width of the taper at the end of the connector. The default value is <code>nil</code> .
<code>?taperStartClamp f_length</code>	
	Length of the clamping region at the start of the connector. The default value is <code>nil</code> .
	This is the zone where the derived function is changed to align the curve with the facet angle.
	See <u>ccCreateConnectorWidthExpressionList</u> .
<code>?taperEndClamp f_length</code>	
	Length of the clamping region at the end of the connector. The default value is <code>nil</code> .
	See <u>ccCreateConnectorWidthExpressionList</u> .
<code>?angle f_angle</code>	Bend angle of the arc. The default value is <code>nil</code> .
<code>?radius f_radius</code>	Radius of the arc. The default value is <code>nil</code> .

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?arcPart f_arcPart`

Any value between 0.0 and 1.0 indicating the ratio of the circular bending angle to the overall bending angle.

- When the value is 1.0, the connector consists of a single circular arc.
- When the value is 0.0, the connector consists of two clothoids.

`?Rmin f_Rmin`

Overwrites or specifies the minimum curvature radius. The default value is `nil`.

If specified, the curve is not drawn unless the `?radius` is greater or equal to `?Rmin`.

`?modeSpec l_l_modeSpec`

List of list of arguments to calculate the mode properties (`modeProp`) for the connector:

```
list( list(wavelength1 wavelength2...) list(temperature1
      temperature2...) )
```

Example:

```
list(list(1550n) list(27)) )
```

`?modePropOnly l_modePropOnly`

Returns a partial connector disembodied property list, without drawing it in the cellview. The default value is `nil`.

`?portPinPurpose t_portPinPurpose`

Layer purpose to use when creating pins. The default purpose for pins is "drawing".

`?waveguideDefPurpose t_waveguideDefPurpose`

Enables changing the `waveguideDef` purpose and select a different `waveguideDef` profile from the technology file.

Value Returned

l_dpl

A disembodied property list (DPL) of curve, pins and their angles, and arguments passed to the function. This can be used to easily use the connector in a SKILL script.

A DPL starts with `nil` and has keys to access different information or values. A connector DPL has nested DPLs. You can access this information in SKILL using the `->` operator.

The main keys are:

- `connectorName`: A string with the name of the connector.
- `startPort`: A DPL containing the startPort information, such as name, direction, and facet information.
- `endPort`: A DPL containing the endPort information, such as name, direction, and facet information.
- `waveguideLayer`: A string with the name of the waveguide layer used by the connector.
- `args`: A list of all arguments used to generate the connector.

`nil`

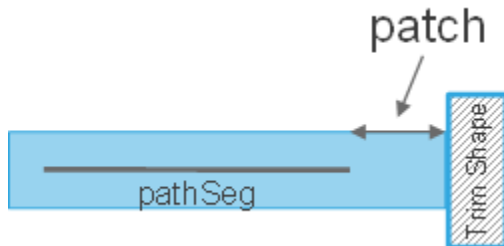
Connector is not created.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

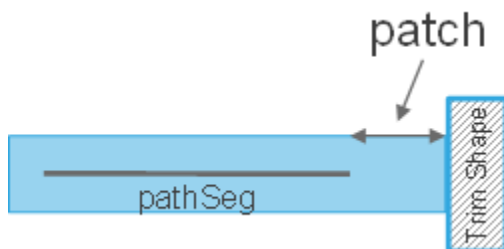
Examples

```
cv = geGetEditCellView()  
phoBendConnector(cv "waveguide" ?rotation 90.0 ?width 0.5 ?angle 90)
```



The following example shows the difference between the `arcPart 0.0` (clothoid) and the `arcPart 1.0` (arc).

```
phoBendConnector(cv "waveguide" ?rotation 90.0 ?width 0.5 ?angle 90 ?radius 30  
?arcPart 0.0/ 1.0 )
```



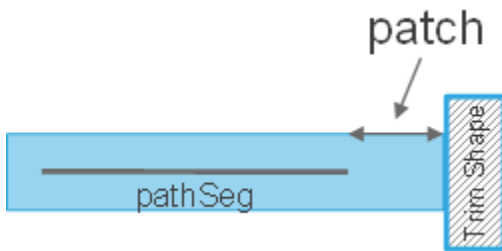
The following example shows the tapering:

```
phoBendConnector(cv "waveguide" ?rotation 90.0 ?width 0.5 ?angle 90  
?taperStyle 'exp  
?startWidth 1.0  
?endWidth 4.0
```


Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
?taperStartClamp 2.0  
?taperEndClamp 2.0  
)
```



Related Topics

[phoCurveDeltaConnector](#)

[phoDrawCompWaveguideLayout](#)

[phoSineDeltaConnector](#)

[phoStraightConnector](#)

[phoWayPointConnector](#)

[Photonics Connector Functions](#)

phoCurveDeltaConnector

```
phoCurveDeltaConnector(  
    g_cellViewId  
    S_waveguideLayer  
    [ ?rotation n_rotation ]  
    [ ?offset l_offset ]  
    [ ?full g_full ]  
    [ ?startPortName t_PortName ]  
    [ ?endPortName t_PortName ]  
    [ ?startPortDirection t_Direction ]  
    [ ?endPortDirection t_Direction ]  
    [ ?width f_width ]  
    [ ?deltaX f_deltaX ]  
    [ ?deltaY f_deltaY ]  
    [ ?startPortAngle f_startPortAngle ]  
    [ ?endPortAngle f_endPortAngle ]  
    [ ?taperStyle s_taperStyle ]  
    [ ?startWidth f_width ]  
    [ ?endWidth f_width ]  
    [ ?taperStartClamp n_length ]  
    [ ?taperEndClamp n_length ]  
    [ ?startPortRadius f_startPortRadius ]  
    [ ?endPortRadius f_endPortRadius ]  
    [ ?modeSpec l_l_modeSpec ]  
    [ ?modePropOnly g_modePropOnly ]  
    [ ?portPinPurpose t_portPinPurpose ]  
    [ ?Rmin n_Rmin ]  
    [ ?method x_method ]  
    [ ?costFunction l_costFunction ]  
)  
=> l_dpl / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a delta connector, defined by deltaX, deltaY, and facet angles. The connector calculates the curve to connect the start and end with their respective angles according to the passed parameters.

Arguments

<i>d_cellViewId</i>	Cellview ID of the layout to be used. For a connector, the generator checks its view type and whether the cellview is editable and is a maskLayout.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`t_waveguideLayer` Name of the abstract layer or the design intent layer in which you want to draw the connector. This must be a layer with the `waveguide` function defined in the PDK technology file. It is also used to determine the relevant `waveguideDef` (cross-section) in the technology file, to generate the needed layers to implement the waveguide.

`?rotation n_rotation`

Rotation angle in degrees for the bend connector.

The default value is `(0,0)`, which means there is no rotation.

The facet angle `0.0` points to the bottom.

`?offset l_offset list(n_lhs n_rhs)`

Offset value for the connector.

The default value is `(0,0)`, which means there is no offset.

`?full g_full`

A Boolean when set to `t` specifies whether a waveguide is created with all its layers from the cross section as defined in the technology file. This is the default.

When set to `nil`, only the design intent layer `t_waveguideLayer` is drawn, generating an abstract of the connector.

`?startPortName t_PortName`

Name of the start port. The default value is `"start"`.

A `nil` port name results in a connector generated without a port at the start. This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?endPortName t_PortName`

Name of the end port. The default value is `"end"`.

This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?startPortDirection t_Direction`

Direction of the start port. The default value is `"inputOutput"`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?endPortDirection *t_Direction*

Direction of the end port. The default is "inputOutput".

?width *n_width*

Width of the waveguide layer. The default value is `nil`.

?deltaX *f_deltaX*

Distance in microns along the x axis between the start and end points of the connector. The default value is `nil`.

?deltaY *f_deltaY*

Distance in microns along the y axis between the start and end points of the connector. The default value is `nil`.

?startPortAngle *f_startPortAngle*

Angle of the facet at the start port. The default value is `nil`.

?endPortAngle *f_endPortAngle*

Angle of the facet at the end port. The default value is `nil`.

?taperStyle *s_taperStyle*

Style of the tapering.

Valid values are: 'none, 'linear, 'exp and 'parabolic.
The default value is 'none

?startWidth *n_width*

Width of the taper at the start of the connector. The default value is `nil`.

?endWidth *f_width*

Width of the taper at the end of the connector. The default value is `nil`.

?taperStartClamp *f_length*

Length of the clamping region at the start of the connector. The default value is `nil`.

This is the zone where the derived function is changed to align the curve with the facet angle.

See [ccCreateConnectorWidthExpressionList](#).

?taperEndClamp *f_length*

Length of the clamping region at the end of the connector. The default value is `nil`.

See [ccCreateConnectorWidthExpressionList](#).

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?startPortRadius f_startPortRadius`

Radius of the start port. If specified, it must be set to a value greater than the minimum bend radius of the waveguideLayer. This value changes the inflection of the curve at the start port. The default value is `nil`. This parameter is not always active. For example, there is no effect from setting the radius of a port on a straight connector.

`?endPortRadius f_endPortRadius`

Radius of the end port. If specified, it must be set to a value greater than the minimum bend radius of the waveguideLayer. This value changes the inflection of the curve at the start port. The default value is `nil`.

`?modeSpec l_l_modeSpec`

List of list of arguments to calculate the mode properties (modeProp) for the connector:

```
list( list(wavelength1 wavelength2...) list(temperature1
      temperature2...) )
```

Example:

```
list(list(1550n) list(27)) )
```

`?modePropOnly l_modePropOnly`

Returns a partial connector disembodied property list, without drawing it in the cellview. The default value is `nil`.

`?portPinPurpose t_portPinPurpose`

Layer purpose to use when creating pins. The default purpose for pins is "drawing".

`?Rmin f_Rmin`

Overwrites or specifies the minimum curvature radius. The default value is `nil`.

If specified, the curve is not drawn unless the `?radius` is greater or equal to `?Rmin`.

`?method x_method`

Method number from 0 to 9. The default value is 0.

See [ccCurveConnector](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?costFunction l_costFunction`

`list(n_A n_B n_C n_D)`

List of additional numeric parameters passed to the connector calculation methods. The default value is `nil`. 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_costFunction` is empty, the result is the first solution that is found. If `l_costFunction` is not empty, it is assumed to contain list of four weights of the cost function.

Value Returned

`l_dpl`

A disembodied property list (DPL) of curve, pins and their angles, and arguments passed to the function. This can be used to easily use the connector in a SKILL script.

A DPL starts with `nil` and has keys to access different information or values. A connector DPL has nested DPLs. You can access this information in SKILL using the `->` operator.

The main keys are:

- `connectorName`: A string with the name of the connector.
- `startPort`: A DPL containing the startPort information, such as name, direction, and facet information.
- `endPort`: A DPL containing the endPort information, such as name, direction, and facet information.
- `waveguideLayer`: A string with the name of the waveguide layer used by the connector.
- `args`: A list of all arguments used to generate the connector.

`nil`

Connector is not created.

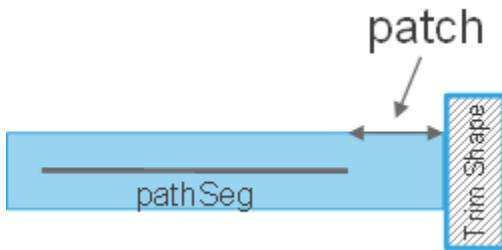
Examples

```
cv = geGetEditCellView()
phoCurveDeltaConnector( cv "waveguide" ?width 0.5
?deltaX 50.0 ?deltaY 20.0
?startPortAngle 90.0 ?endPortAngle 270.0
```

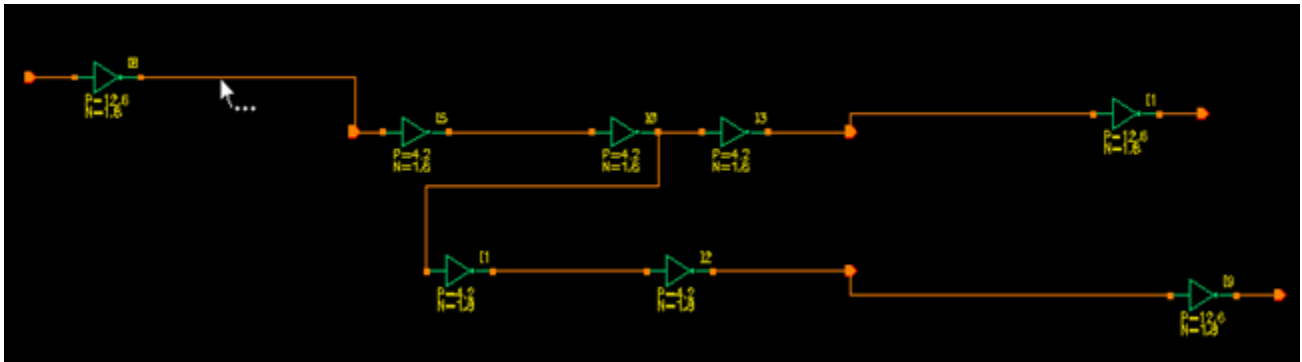
Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

)



```
phoCurveDeltaConnector(cv "waveguide" ?width 0.5
?deltaX 50.0 ?deltaY 20.0
?startPortAngle 0.0 ?endPortAngle 33.0
)
```



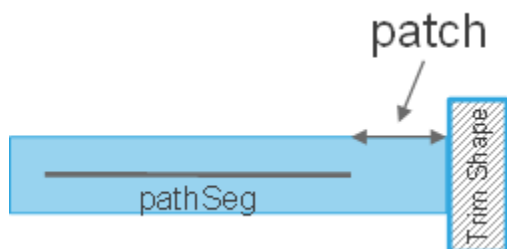
The following example shows the influence of `?startPortRadius` and `?endPortRadius`. In the image, red represents the parameters set to `nil` and white represents the parameters set to 20:

```
phoCurveDeltaConnector(cv "waveguide" ?width .5 ?deltaX 50.0 ?deltaY 20.0
?startPortAngle 90.0 ?endPortAngle 270.0
?startPortRadius nil / 20
?endPortRadius nil / 20
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

)

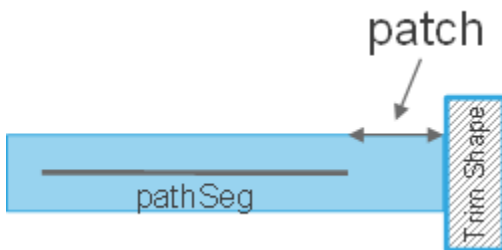


Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

The following example shows the influence of `Rmin` on the connector for values between 10 and 14.

```
phoCurveDeltaConnector(cv "waveguide" ?width .5 ?deltaX 100.0 ?deltaY 40.0
  ?startPortAngle 90.0 ?endPortAngle 270.0
  ?startPortRadius 20 ?endPortRadius 20
  ?Rmin 10...14
)
```

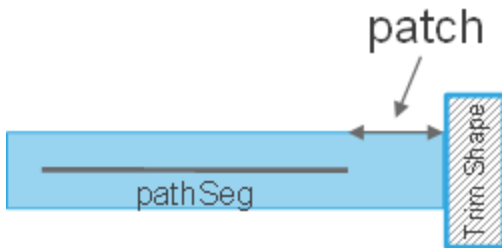


The following example shows the effect when the `?costFunction` parameter is set to `nil` compared to when it is set to a list of parameters

```
phoCurveDeltaConnector(cv "waveguide" ?width .5
  ?deltaX 100.0 ?deltaY 100.0
  ?startPortAngle 90.0 ?endPortAngle 270.0
  ?startPortRadius 20 ?endPortRadius 20
  ?Rmin 10 ?method 0
  ?costFunction nil / list(1 1 0.01 0.5)
)
```

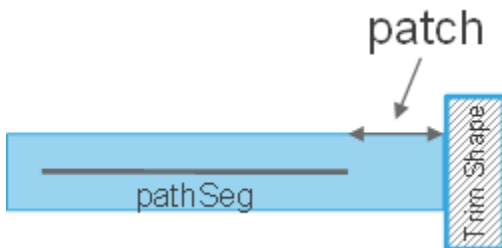
Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)



The following shows below the tapering.

```
phoCurveDeltaConnector( cv "waveguide" ?width 0.7 ?deltaX 50.0 ?deltaY 20.0
?startPortAngle 90.0 ?endPortAngle 270.0
?taperStyle 'parabolic
?startWidth 1.0
?endWidth 4.0
?taperStartClamp 2.0
?taperEndClamp 2.0
)
```



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Related Topics

[phoBendConnector](#)

[phoDrawCompWaveguideLayout](#)

[phoSineDeltaConnector](#)

[phoStraightConnector](#)

[phoWayPointConnector](#)

[Photonics Connector Functions](#)

phoDrawCompWaveguideLayout

```
phoDrawCompWaveguideLayout (
    d_cellViewId
    l_waveguideDevice
    [ ?rotation n_rotation ]
    [ ?offset l_offset ]
    [ ?full g_full ]
    [ ?startPortName t_PortName ]
    [ ?endPortName t_PortName ]
    [ ?startPortDirection t_Direction ]
    [ ?endPortDirection t_Direction ]
    [ ?width n_width ]
    [ ?modeSpec l_l_modeSpec ]
    [ ?portPinPurpose t_portPinPurpose ]
    [ ?instPrefix ? t_instPrefix ]
)
=> l_dpl / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a composite waveguide connector that takes a list of waveguide cells, such as the straight or the bend and creates a unique component. Each device can optionally have its own parameter set applied, such as tapering and width.

Arguments

<i>d_cellViewId</i>	Cellview ID of the layout to be used. For a connector, the generator checks its view type and whether the cellview is editable and is a maskLayout.
<i>l_waveguideDevice</i>	List of waveguide devices and their optional parameters. <pre>list(list(list(libName1 cellName1 viewName1) [list(l_paramA l_paramB)]), list(list(libName2 cellName2 viewName2)))</pre> Where <i>libName</i> is the library name of the device, such as <i>gopdk</i>, <i>cellName</i> is the cell name of the device, such as <i>wgStraight</i>, and <i>viewName</i> is the view of the device, such as <i>layout</i>.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?rotation n_rotation`

Rotation angle in degrees for the bend connector.

The default value is `(0,0)`, which means there is no rotation.

The facet angle `0.0` points to the bottom.

`?offset l_offset list(n_lhs n_rhs)`

Offset value for the connector.

The default value is `(0,0)`, which means there is no offset.

`?full g_full`

A Boolean when set to `t` specifies whether a waveguide is created with all its layers from the cross section as defined in the technology file. This is the default.

When set to `nil`, only the design intent layer `t_waveguideLayer` is drawn, generating an abstract of the connector.

`?startPortName t_PortName`

Name of the start port. The default value is `"start"`.

A `nil` port name results in a connector generated without a port at the start. This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?endPortName t_PortName`

Name of the end port. The default value is `"end"`.

This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?startPortDirection t_Direction`

Direction of the start port. The default value is `"inputOutput"`.

`?endPortDirection t_Direction`

Direction of the end port. The default is `"inputOutput"`.

`?width n_width`

Width of the waveguide layer. The default value is `nil`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?modeSpec l_l_modeSpec`

List of list of arguments to calculate the mode properties (modeProp) for the connector:

```
list( list(wavelength1 wavelength2...) list(temperature1
      temperature2...) )
```

Example:

```
list(list(1550n) list(27)) )
```

`?portPinPurpose t_portPinPurpose`

Layer purpose to use when creating pins. The default purpose for pins is "drawing".

`?instPrefix ? t_instPrefix`

Prefix for the device. All devices defined in the *l_waveguideDevice* will start with prefix `?instPrefix` followed by an indexed number, such as `Opt0`, `Opt1` and so on. The default value is `Opt`.

Value Returned

`l_dpl`

A disembodied property list (DPL) of curve, pins and their angles, and arguments passed to the function. This can be used to easily use the connector in a SKILL script.

A DPL starts with `nil` and has keys to access different information or values. A connector DPL has nested DPLs. You can access this information in SKILL using the `->` operator.

The main keys are:

- `connectorName`: A string with the name of the connector.
- `startPort`: A DPL containing the startPort information, such as name, direction, and facet information.
- `endPort`: A DPL containing the endPort information, such as name, direction, and facet information.
- `waveguideLayer`: A string with the name of the waveguide layer used by the connector.
- `args`: A list of all arguments used to generate the connector.

Virtuoso Studio Design Environment SKILL Reference

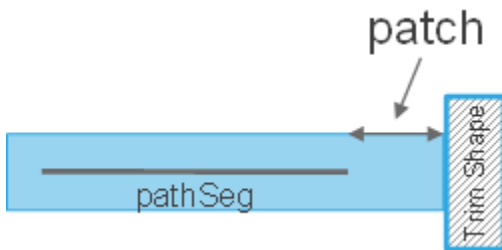
Photonics Functions (Virtuoso Photonics Option)

nil Connector is not created.

Examples

The following example uses the cells "wgStraight" and "wgBend" from the library "gopdk". The cells are automatically abutted.

```
cv = geGetEditCellView()
phoDrawCompWaveguideLayout(cv
list(
list( list("gopdk" "wgStraight" "layout"))
      list( list("gopdk" "wgBend" "layout"))
    )
)
```

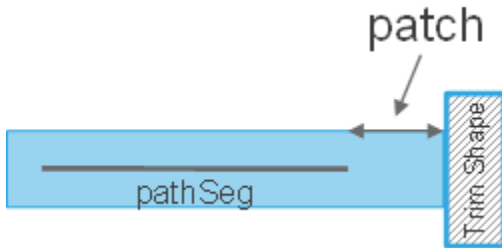


The following example shows how to change the width of the previous combination:

```
phoDrawCompWaveguideLayout(cv
list(
  list(
list("gopdk" "wgStraight" "layout")
      list(list("width" "string" "0.7u"))
    )
  list(
list("gopdk" "wgBend" "layout")
      list(list("width" "string" "0.7u"))
    )
  )
)
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

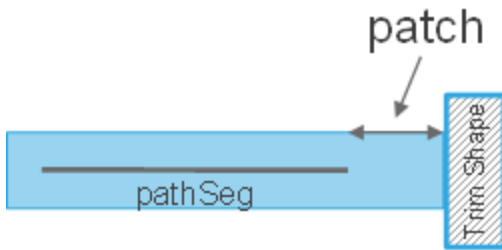


Parameters for each device can be set. The following example shows how to change the length of the first device and the bend angle of the second device.

```
phoDrawCompWaveguideLayout (
    cv
    list(
        list(list("gopdk" "wgStraight" "layout")
            list(
                list("width" "string" "0.7u")
                list("length" "string" "10u")
            )
        )
        list( list("gopdk" "wgBend" "layout")
            list(
                list("width" "string" "0.7u")
                list("angle" "string" "180")
            )
        )
    )
)
```


Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)



Related Topics

[phoBendConnector](#)

[phoCurveDeltaConnector](#)

[phoSineDeltaConnector](#)

[phoStraightConnector](#)

[phoWayPointConnector](#)

[Photonics Connector Functions](#)

phoSineDeltaConnector

```
phoSineDeltaConnector(  
    d_cellViewId  
    t_waveguideLayer  
    [ ?rotation n_rotation ]  
    [ ?offset l_offset ]  
    [ ?full g_full ]  
    [ ?startPortName t_PortName ]  
    [ ?endPortName t_PortName ]  
    [ ?startPortDirection t_Direction ]  
    [ ?endPortDirection t_Direction ]  
    [ ?waveguidePurpose t_waveguidePurpose ]  
    [ ?startPortAngle f_startPortAngle ]  
    [ ?endPortAngle f_endPortAngle ]  
    [ ?deltaX f_deltaX ]  
    [ ?deltaY f_deltaY ]  
    [ ?taperStyle s_taperStyle ]  
    [ ?startWidth n_width ]  
    [ ?endWidth f_width ]  
    [ ?taperStartClamp f_length ]  
    [ ?taperEndClamp f_length ]  
    [ ?modeSpec l_l_modeSpec ]  
    [ ?modePropOnly l_modePropOnly ]  
    [ ?portPinPurpose t_portPinPurpose ]  
    [ ?waveguideDefPurpose t_waveguideDefPurpose ]  
)  
=> l_dpl / nil
```

Description

(ICADVM20.1 Photonics Only) Creates a curvy sineDelta connector defined by deltaX and deltaY. Facet angles need to be opposite (difference of 180 degrees). The connector calculates the curve to connect the start and end with their respective angles according to the passed parameters.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Arguments

<code>d_cellViewId</code>	Cellview ID of the layout to be used. For a connector, the generator checks its view type and whether the cellview is editable and is a <code>maskLayout</code> .
<code>t_waveguideLayer</code>	Name of the abstract layer or the design intent layer in which you want to draw the connector. This must be a layer with the <code>waveguide</code> function defined in the PDK technology file. It is also used to determine the relevant <code>waveguideDef</code> (cross-section) in the technology file, to generate the needed layers to implement the waveguide.
<code>?rotation n_rotation</code>	Rotation angle in degrees for the bend connector. The default value is <code>(0,0)</code>, which means there is no rotation. The facet angle <code>0.0</code> points to the bottom.
<code>?offset l_offset</code>	<code>list(n_lhs n_rhs)</code> Offset value for the connector. The default value is <code>(0,0)</code>, which means there is no offset.
<code>?full g_full</code>	A Boolean when set to <code>t</code> specifies whether a waveguide is created with all its layers from the cross section as defined in the technology file. This is the default. When set to <code>nil</code>, only the design intent layer <code>t_waveguideLayer</code> is drawn, generating an abstract of the connector.
<code>?startPortName t_PortName</code>	Name of the start port. The default value is <code>"start"</code>. A <code>nil</code> port name results in a connector generated without a port at the start. This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?endPortName t_PortName`

Name of the end port. The default value is "end".

This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?startPortDirection t_Direction`

Direction of the start port. The default value is "inputOutput".

`?endPortDirection t_Direction`

Direction of the end port. The default is "inputOutput".

`?width n_width` Width of the waveguide layer. The default value is `nil`.

`?waveguidePurpose t_waveguidePurpose`

Purpose of the waveguide layer. The default is "drawing".

`?startPortAngle f_startPortAngle`

Angle of the facet at the start port. If specified, the difference between `?startPortAngle` and `?endPortAngle` must be 180 degrees. The default value is `90.0`.

`?endPortAngle f_endPortAngle`

Angle of the facet at the end port. If specified, the difference between `?startPortAngle` and `?endPortAngle` must be 180 degrees. The default value is `-90.0`.

`?deltaX f_deltaX` Distance in microns along the x axis between the start and end points of the connector. The default value is `nil`.

`?deltaY f_deltaY` Distance in microns along the y axis between the start and end points of the connector. The default value is `nil`.

`?taperStyle s_taperStyle`

Style of the tapering.

Valid values are: 'none', 'linear', 'exp' and 'parabolic'. The default value is 'none'.

`?startWidth n_width`

Width of the taper at the start of the connector. The default value is `nil`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?endWidth f_width` Width of the taper at the end of the connector. The default value is `nil`.

`?taperStartClamp f_length`

Length of the clamping region at the start of the connector. The default value is `nil`.

This is the zone where the derived function is changed to align the curve with the facet angle.

See [ccCreateConnectorWidthExpressionList](#).

`?taperEndClamp f_length`

Length of the clamping region at the end of the connector. The default value is `nil`.

See [ccCreateConnectorWidthExpressionList](#).

`?modeSpec l_l_modeSpec`

List of list of arguments to calculate the mode properties (`modeProp`) for the connector:

```
list( list(wavelength1 wavelength2...) list(temperature1
      temperature2...))
```

Example:

```
list(list(1550n) list(27)) )
```

`?modePropOnly l_modePropOnly`

Returns a partial connector disembodied property list, without drawing it in the cellview. The default value is `nil`.

`?portPinPurpose t_portPinPurpose`

Layer purpose to use when creating pins. The default purpose for pins is "drawing".

`?waveguideDefPurpose t_waveguideDefPurpose`

Enables changing the waveguideDef purpose and select a different waveguideDef profile from the technology file.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

l_dpl

A disembodied property list (DPL) of curve, pins and their angles, and arguments passed to the function. This can be used to easily use the connector in a SKILL script.

A DPL starts with `nil` and has keys to access different information or values. A connector DPL has nested DPLs. You can access this information in SKILL using the `->` operator.

The main keys are:

- `connectorName`: A string with the name of the connector.
- `startPort`: A DPL containing the startPort information, such as name, direction, and facet information.
- `endPort`: A DPL containing the endPort information, such as name, direction, and facet information.
- `waveguideLayer`: A string with the name of the waveguide layer used by the connector.
- `args`: A list of all arguments used to generate the connector.

`nil`

Connector is not created.

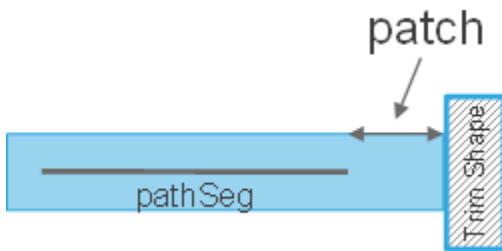
Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Examples

```
cv = geGetEditCellView()
```

```
phoSineDeltaConnector(cv "waveguide" ?width 0.5  
?deltaX 20 ?deltaY 20  
?startPortAngle 90?endPortAngle -90  
)
```

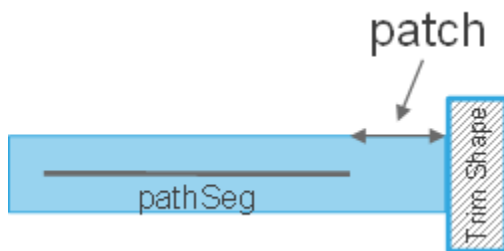


Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

The following shows the effect of tapering.

```
phoSineDeltaConnector(cv "waveguide" ?width 0.5 ?deltaX 20 ?deltaY 20  
?startPortAngle 90 ?endPortAngle -90  
?taperStyle 'linear  
?startWidth 1.0  
?endWidth 2.0  
?taperStartClamp 2.0  
?taperEndClamp 2.0)
```



Related Topics

[phoBendConnector](#)

[phoCurveDeltaConnector](#)

[phoDrawCompWaveguideLayout](#)

[phoStraightConnector](#)

[phoWayPointConnector](#)

[Photonics Connector Functions](#)

phoStraightConnector

```
phoStraightConnector(  
    d_cellViewId  
    t_waveguideLayer  
    [ ?rotation n_rotation ]  
    [ ?offset l_offset ]  
    [ ?full g_full ]  
    [ ?startPortName t_PortName ]  
    [ ?endPortName t_PortName ]  
    [ ?startPortDirection t_Direction ]  
    [ ?endPortDirection t_Direction ]  
    [ ?length f_length ]  
    [ ?width n_width ]  
    [ ?modeSpec l_l_modeSpec ]  
    [ ?modePropOnly l_modePropOnly ]  
    [ ?portPinPurpose t_portPinPurpose ]  
    [ ?taperStyle s_taperStyle ]  
    [ ?startWidth n_width ]  
    [ ?endWidth f_width ]  
    [ ?autoWide g_autoWide ]  
    [ ?autoWideWideWidth n_wideWideWidth ]  
    [ ?autoWideTaperStyle s_autoWideTaperStyle ]  
    [ ?autoWideMinWideSectionLength n_wideLength ]  
    [ ?autoWideTaperLength n_autoWideTaperLength ]  
    [ ?autoWideTaperAngle n_wideTaperAngle ]  
    [ ?autoWideStubLength n_wideStubLength ]  
    [ ?waveguideDefPurpose t_waveguideDefPurpose ]  
)  
=> l_dp1 / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a straight connector with tapering and auto-widening options.

Arguments

<i>d_cellViewId</i>	Cellview ID of the layout to be used. For a connector, the generator checks its view type and whether the cellview is editable and is a <code>maskLayout</code> .

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`t_waveguideLayer` Name of the abstract layer or the design intent layer in which you want to draw the connector. This must be a layer with the `waveguide` function defined in the PDK technology file. It is also used to determine the relevant `waveguideDef` (cross-section) in the technology file, to generate the needed layers to implement the waveguide.

`?rotation n_rotation`

Rotation angle in degrees for the bend connector.

The default value is `(0,0)`, which means there is no rotation.

The facet angle `0.0` points to the bottom.

`?offset l_offset list(n_lhs n_rhs)`

Offset value for the connector.

The default value is `(0,0)`, which means there is no offset.

`?full g_full`

A Boolean when set to `t` specifies whether a waveguide is created with all its layers from the cross section as defined in the technology file. This is the default.

When set to `nil`, only the design intent layer `t_waveguideLayer` is drawn, generating an abstract of the connector.

`?startPortName t_PortName`

Name of the start port. The default value is `"start"`.

A `nil` port name results in a connector generated without a port at the start. This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?endPortName t_PortName`

Name of the end port. The default value is `"end"`.

This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

`?startPortDirection t_Direction`

Direction of the start port. The default value is `"inputOutput"`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?endPortDirection t_Direction`

Direction of the end port. The default is "inputOutput".

`?length f_length`

Length of the connector. The default value is `nil`.

`?width n_width`

Width of the connector. The default value is `nil`.

`?modeSpec l_l_modeSpec`

List of list of arguments to calculate the mode properties (modeProp) for the connector:

```
list( list(wavelength1 wavelength2...) list(temperature1
      temperature2...))
```

Example:

```
list(list(1550n) list(27)) )
```

`?modePropOnly l_modePropOnly`

Returns a partial connector disembodied property list, without drawing it in the cellview. The default value is `nil`.

`?portPinPurpose t_portPinPurpose`

Layer purpose to use when creating pins. The default purpose for pins is "drawing".

`?taperStyle s_taperStyle`

Style of the tapering.

Valid values are: 'none, 'linear, 'exp and 'parabolic.
The default value is 'none.

`?startWidth n_width`

Width of the taper at the start of the connector. The default value is `nil`.

`?endWidth f_width`

Width of the taper at the end of the connector. The default value is `nil`.

Enables or disables the auto-widening feature. Tapering and auto-widening cannot be enabled at the same time. The default value is `nil`.

`?autoWideWideWidth n_wideWideWidth`

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Width of auto-widening. It must be larger than the waveguide width. The default value is `nil`.

?autoWideTaperStyle *s_autoWideTaperStyle*

Style of the tapering. Options are 'none, 'linear, 'exp and 'parabolic. The default value is 'none.

?autoWideMinWideSectionLength *n_wideLength*

Length of the auto-widening cone. This value must be less than the connector length. The default value is `nil`.

?autoWideTaperLength *n_autoWideTaperLength*

Minimum length of the taper. The default value is `nil`.

To specify this argument, the length of the connector must be the sum of two times `autoWideStubLength`, two times `autoWideTaperLength`, and `autoWideMinLength`.

If this condition is not met, the `autoWide` parameter will be ignored. `autoWideTaperAngle` and `autoWideTaperLength` are mutually exclusive.

?autoWideTaperAngle *n_wideTaperAngle*

Angle of the taper cone in degrees. The default value is `nil`.

`autoWideTaperAngle` and `autoWideTaperLength` are mutually exclusive. This angle and the `autoWideWideWidth` determine the `autoWideTaperLength` and the final value of the `autoWide` length. If the angle is too small and `autoWideMinLength` is not met, the `autoWide` process will not occur.

?autoWideStubLength *n_wideStubLength*

Length of the offset between the start or end of the connector and the beginning of `autoWide`. The default value is `nil`.

?waveguideDefPurpose *t_waveguideDefPurpose*

Enables changing the waveguideDef purpose and select a different waveguideDef profile from the technology file.

Value Returned

l_dpl

A disembodied property list (DPL) of curve, pins and their angles, and arguments passed to the function. This can be used to easily use the connector in a SKILL script.

A DPL starts with `nil` and has keys to access different information or values. A connector DPL has nested DPLs. You can access this information in SKILL using the `->` operator.

The main keys are:

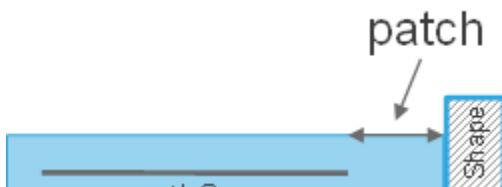
- `connectorName`: A string with the name of the connector.
- `startPort`: A DPL containing the startPort information, such as name, direction, and facet information.
- `endPort`: A DPL containing the endPort information, such as name, direction, and facet information.
- `waveguideLayer`: A string with the name of the waveguide layer used by the connector.
- `args`: A list of all arguments used to generate the connector.

`nil`

Connector is not created.

Examples

```
cv = geGetEditCellView()  
phoStraightConnector(cv "waveguide" ?offset 5:5 ?rotation 90 ?length 10 ?width  
0.5)
```



Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
phoStraightConnector(cv "waveguide" ?offset 5:5 ?rotation 90 ?length 30 ?width 0.7  
?taperStyle 'exp  
?startWidth 5 ?endWidth 0.7  
?taperStartClamp 0.1 ?taperEndClamp 0.1  
)
```



```
phoStraightConnector(cv "waveguide" ?offset 5:5 ?rotation 90 ?length 30 ?width 0.7  
?autoWide t ?autoWideWidth 4.0  
?autoWideMinWideSectionLength 5 ?autoWideStubLength 2.0  
?autoWideTaperStyle 'parabolic ?autoWideTaperLength 5  
)
```



Related Topics

[phoBendConnector](#)

[phoCurveDeltaConnector](#)

[phoDrawCompWaveguideLayout](#)

[phoSineDeltaConnector](#)

[phoWayPointConnector](#)

[Photonics Connector Functions](#)

phoWayPointConnector

```
phoWayPointConnector(  
    d_cellViewId  
    t_waveguideLayer  
    [ ?rotation n_rotation ]  
    [ ?full g_full ]  
    [ ?startPortName t_PortName ]  
    [ ?endPortName t_PortName ]  
    [ ?width f_width ]  
    [ ?shapeDataList l_shapeDataList ]  
    [ ?startPortAngle f_startPortAngle ]  
    [ ?endPortAngle f_endPortAngle ]  
    [ ?autoWide g_autoWide ]  
    [ ?autoWideWideWidth n_wideWideWidth ]  
    [ ?autoWideMinWideSectionLength n_wideLength ]  
    [ ?autoWideTaperLength n_autoWideTaperLength ]  
    [ ?autoWideTaperAngle n_wideTaperAngle ]  
    [ ?autoWideStubLength n_wideStubLength ]  
    [ ?autoWideTaperStyle s_autoWideTaperStyle ]  
    [ ?arcPart f_arcPart ]  
    [ ?minRadius f_minRadius ]  
    [ ?vfoProtocolClass t_vfoProtocolClass ]  
    [ ?modeSpec l_l_modeSpec ]  
    [ ?modePropOnly l_modePropOnly ]  
    [ ?portPinPurpose t_portPinPurpose ]  
    [ ?waveguideDefPurpose t_waveguideDefPurpose ]  
)  
=> l_dpl / nil
```

Description

(ICADVM20.1 Photonics Only) Returns a waypoint connector that connects the start port to the end port by a sequence of straight-line segments and bend connectors. The straight-line segments are defined by a polynomial line through a series of waypoints. This series of waypoints is defined by the key argument `?shapeDataList`.

Arguments

<code>d_cellViewId</code>	Cellview ID of the layout to be used. For a connector, the generator checks its view type and whether the cellview is editable and is a maskLayout.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

t_waveguideLayer Name of the abstract layer or the design intent layer in which you want to draw the connector. This must be a layer with the `waveguide` function defined in the PDK technology file. It is also used to determine the relevant `waveguideDef` (cross-section) in the technology file, to generate the needed layers to implement the waveguide.

?rotation n_rotation

Rotation angle in degrees for the bend connector.

The default value is `(0,0)`, which means there is no rotation.

The facet angle `0.0` points to the bottom.

?offset l_offset `list(n_lhs n_rhs)`

Offset value for the connector.

The default value is `(0,0)`, which means there is no offset.

?full g_full

A Boolean when set to `t` specifies whether a waveguide is created with all its layers from the cross section as defined in the technology file. This is the default.

When set to `nil`, only the design intent layer `t_waveguideLayer` is drawn, generating an abstract of the connector.

?startPortName t_PortName

Name of the start port. The default value is `"start"`.

A `nil` port name results in a connector generated without a port at the start. This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

?endPortName t_PortName

Name of the end port. The default value is `"end"`.

This may be useful when composing a complex layout with multiple connectors in which ports are not needed in the middle because there is no hierarchical connection point at that location.

?width f_width Width of the waveguide layer. The default value is `nil`.

?shapeDataList l_shapeDataList

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

List of points of interest (waypoints) for the connector. This parameter cannot be left empty and the list must contain at least two points. The default value is `nil`.

`?startPortAngle f_startPortAngle`

Angle of the facet at the start port. The default value is `nil`.

When `?startPortAngle` is specified, a new point and additional bend might be inserted between first point and second point of `l_shapeDataList` to meet this constraint.

`?startPortRadius f_startPortRadius`

Radius of the start port. If specified, it must be set to a value greater than the minimum bend radius of the `waveguideLayer`. This value changes the inflection of the curve at the start port. The default value is `nil`. This parameter is not always active. For example, there is no effect from setting the radius of a port on a straight connector.

If `?startPortAngle` is also specified, the connector will use `?startPortRadius` on the radius of the newly added bend.

`?endPortAngle f_endPortAngle`

Angle of the facet at the end port. The default value is `nil`.

If `?endPortAngle` is specified, a new point and additional bend might be inserted between the last point and second to last point of `l_shapeDataList` to meet this constraint.

`?autoWide g_autoWide`

Enables or disables the auto-widening feature. Tapering and auto-widening cannot be enabled at the same time. The default value is `nil`.

`?autoWideWideWidth n_wideWideWidth`

Width of auto-widening. It must be larger than the waveguide width. The default value is `nil`.

`?autoWideMinWideSectionLength n_wideLength`

Length of the auto-widening cone. This value must be less than the connector length. The default value is `nil`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

?autoWideTaperLength *n_autoWideTaperLength*

Minimum length of the taper. The default value is `nil`.

To specify this argument, the length of the connector must be the sum of two times `autoWideStubLength`, two times `autoWideTaperLength`, and `autoWideMinLength`.

If this condition is not met, the `autoWide` parameter will be ignored. `autoWideTaperAngle` and `autoWideTaperLength` are mutually exclusive.

?autoWideTaperAngle *n_wideTaperAngle*

Angle of the taper cone in degrees. The default value is `nil`.

`autoWideTaperAngle` and `autoWideTaperLength` are mutually exclusive. This angle and the `autoWideWidth` determine the `autoWideTaperLength` and the final value of the `autoWide` length. If the angle is too small and `autoWideMinLength` is not met, the `autoWide` process will not occur.

?autoWideStubLength *n_wideStubLength*

Length of the offset between the start or end of the connector and the beginning of `autoWide`. The default value is `nil`.

?autoWideTaperStyle *s_autoWideTaperStyle*

Style of the tapering. Options are `'none`, `'linear`, `'exp` and `'parabolic`. The default value is `'none`.

?arcPart *f_arcPart* Any value between 0.0 and 1.0 indicating the ratio of the circular bending angle to the overall bending angle.

- When the value is 1.0, the connector consists of a single circular arc.
- When the value is 0.0, the connector consists of two clothoids.

?minRadius *f_minRadius*

Overwrites or specifies the minimum curvature radius. The default value is `nil`.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

`?vfoProtocolClass t_vfoProtocolClass`

Name of the SKILL handler called to manage the interactive fluid edit. The default value is "phoFluidWGModifyClass".

See [Introduction to Fluid Guard Rings](#).

`?modeSpec l_l_modeSpec`

List of list of arguments to calculate the mode properties (modeProp) for the connector:

```
list( list(wavelength1 wavelength2...) list(temperature1
      temperature2...))
```

Example:

```
list(list(1550n) list(27)) )
```

`?modePropOnly l_modePropOnly`

Returns a partial connector disembodied property list, without drawing it in the cellview. The default value is nil.

`?portPinPurpose t_portPinPurpose`

Layer purpose to use when creating pins. The default purpose for pins is "drawing".

`?waveguideDefPurpose t_waveguideDefPurpose`

Enables changing the waveguideDef purpose and select a different waveguideDef profile from the technology file.

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Value Returned

l_dpl

A disembodied property list (DPL) of curve, pins and their angles, and arguments passed to the function. This can be used to easily use the connector in a SKILL script.

A DPL starts with `nil` and has keys to access different information or values. A connector DPL has nested DPLs. You can access this information in SKILL using the `->` operator.

The main keys are:

- `connectorName`: A string with the name of the connector.
- `startPort`: A DPL containing the startPort information, such as name, direction, and facet information.
- `endPort`: A DPL containing the endPort information, such as name, direction, and facet information.
- `waveguideLayer`: A string with the name of the waveguide layer used by the connector.
- `args`: A list of all arguments used to generate the connector.

`nil`

Connector is not created.

Examples

Define first the list of waypoints:

```
shapeDataList = list(  
0:0  
15.708:20  
31.4159:0  
47.12385:-20  
62.8318:0  
78.54:20  
100:20)
```

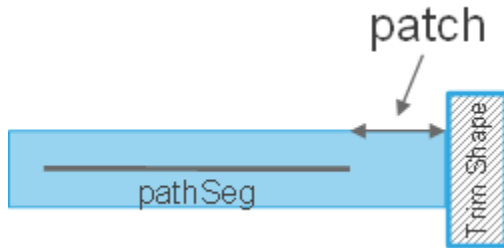
Call the `phoWayPointConnector` SKILL function.

```
cv = geGetEditCellView()
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
phoWayPointConnector(cv "waveguide" ?width 0.5 ?shapeDataList shapeDataList)
```

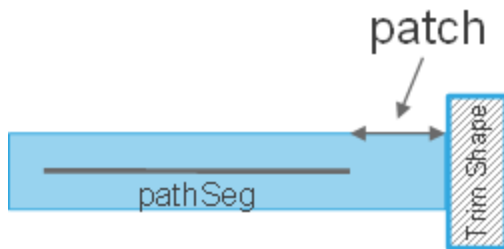


Consider the following parameter list:

```
shapeDataList = list(  
list(0.0 0.0)  
list(150 0.0)  
list(150 -23)  
list(180 -23)  
list(180 40)  
list(200 40)  
list(200 0.0)  
list(230 0.0)  
list(230 40)  
)
```

Call the function for specific parameters.

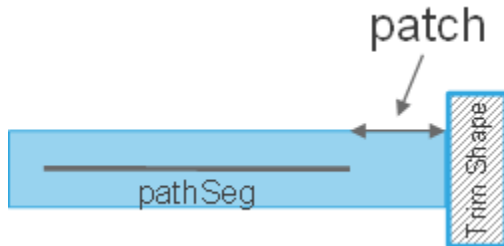
```
cv = geGetEditCellView()  
phoWayPointConnector(cv "waveguide" ?width 0.5 ?shapeDataList shapeDataList)
```



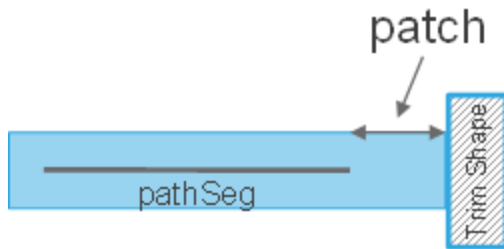
Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

The following is the same curve with the points of shapeDataList, for visibility. Each point is represented by a white square.

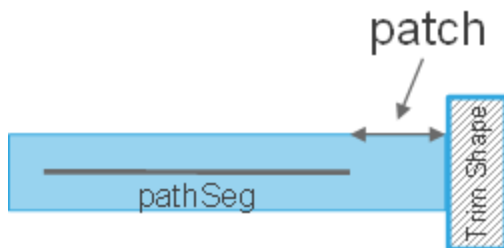


```
phoWayPointConnector( cv "waveguide" ?shapeDataList shapeDataList ?width 0.5
?startPortAngle 30 ?startPortRadius nil)
```



The following show the influence of ?startPortRadius, for values 10, 20 and 30.

```
phoWayPointConnector(cv "waveguide" ?shapeDataList shapeDataList ?width 0.5
?startPortAngle 30
?startPortRadius 10/20/30)
```



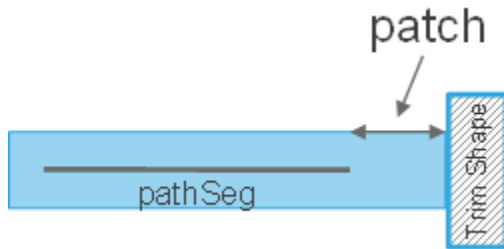
The following image shows the effect of auto-widening on the straight sections:

```
phoWayPointConnector(cv "waveguide" ?width .5 ?shapeDataList shapeDataList
?autoWide t
?autoWideTaperAngle 30
?autoWideTaperStyle 'exp
```

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

```
?autoWideWidth 3.0  
?autoWideStubLength 2.0  
?autoWideMinWideSectionLength 1.0
```



Related Topics

[phoBendConnector](#)

[phoCurveDeltaConnector](#)

[phoDrawCompWaveguideLayout](#)

[phoSineDeltaConnector](#)

[phoStraightConnector](#)

[Photonics Connector Functions](#)

Virtuoso Studio Design Environment SKILL Reference

Photonics Functions (Virtuoso Photonics Option)

Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions

This topic provides a list of Cadence® SKILL functions that you can use in conjunction with the Pegasus Interactive and the Virtuoso Integrated Physical Verification System (VIPVS) editing feature.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Related Topics

[pgssDRC](#)

[pvsApplyDRC](#)

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

<i>d_cellviewID</i>	Specifies the cellview to process. This must be of type maskLayout. It cannot be a Pcell variant.
<i>t_rulesFile</i>	Specifies the path to the Pegasus rules file.
<i>t_layerMap</i>	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.
<i>t_objectMap</i>	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".

Virtuoso Studio Design Environment SKILL Reference

Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions

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
```

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.

Virtuoso Studio Design Environment SKILL Reference

Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions

Value Returned

<code>t</code>	No operational errors occur and Pegasus DRC was launched successfully.
<code>nil</code>	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) )
```

Related Topics

[pvsApplyDRC](#)

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

<i>d_cellview</i>	Specifies the cellview to process. This must be a mask layout cell. It cannot be a Pcell variant.
<i>t_rulesFile</i>	Specifies the path to the PVS rules file.
<i>t_layerMap</i>	Specifies the path to the layerMap file. If no layerMap file is specified, layerMap in the cell's technology library is used.
<i>t_objectMap</i>	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 <code>"/dev/null"</code> .

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.

Virtuoso Studio Design Environment SKILL Reference

Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions

Value Returned

<code>t</code>	No operational errors occur and Virtuoso IPVS launched successfully.
<code>nil</code>	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)
```

Related Topics

[pgssDRC](#)

Virtuoso Studio Design Environment SKILL Reference

Pegasus Interactive and Virtuoso Integrated Physical Verification System Functions

Attribute Retrieval and Modification Functions

You can set several types of attributes on valid dbObjects. This topic lists the attributes that can be set on a dbObject.

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*.

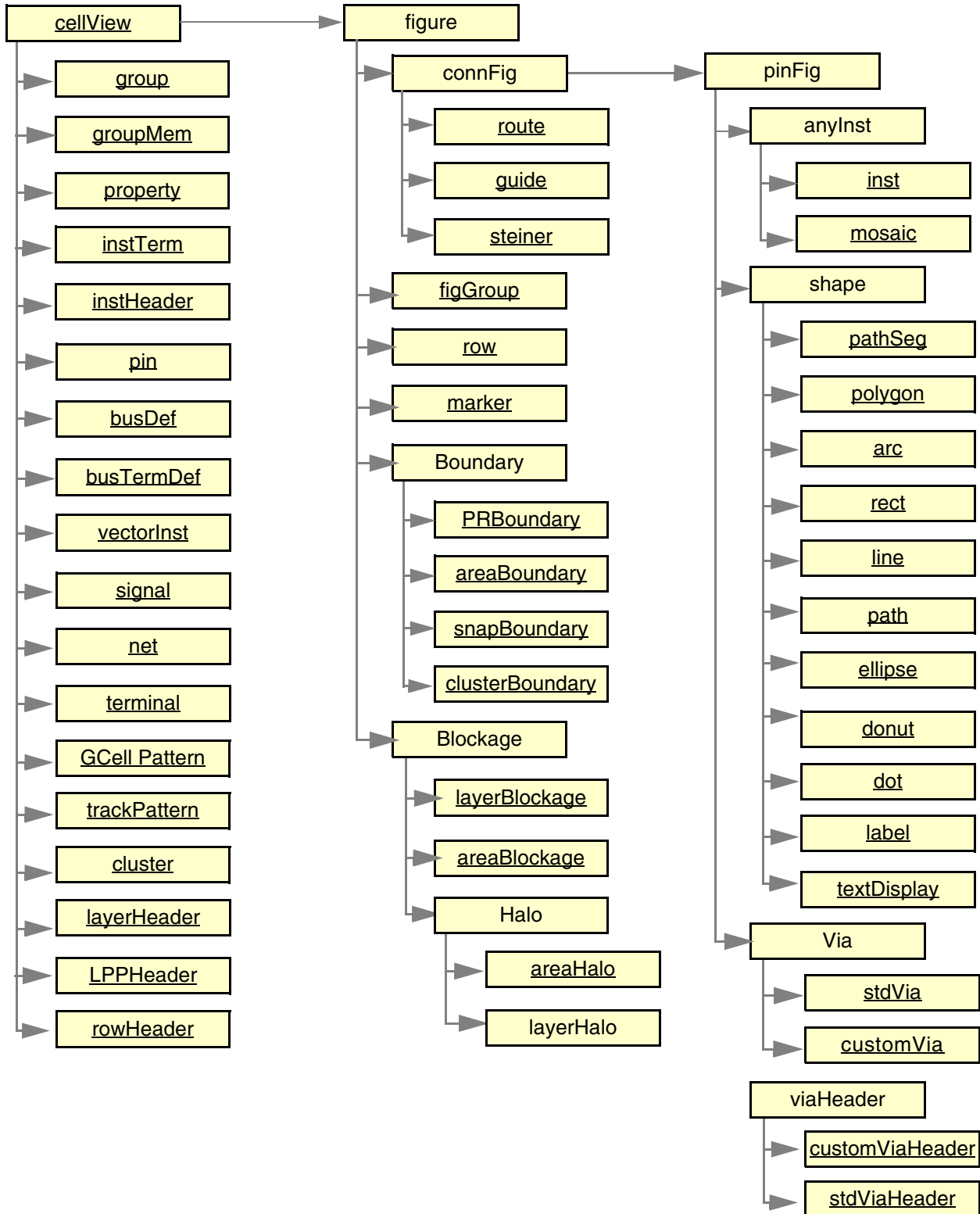
Type	Meaning
bBox	The value is a boundary box (list of two points, lower left and upper right of a rectangular area that usually encloses the object).
integer	The value is an integer.
float	The value is a floating-point number.
string	The value is a string. For strings whose possible values are enumerated, 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 (<i>dbObject</i>).
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 *objIds*. 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 *objIds*. Therefore, 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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

The Cadence design database supports the following object types:



Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Related Topics

- [dbGet Functions](#)
- [dbSet Functions](#)
- [Generic Attributes](#)
- [Cellview Attributes](#)
- [Instance Attributes](#)
- [Instance Header Attributes](#)
- [Mosaic Attributes](#)
- [Instance Terminal Attributes](#)
- [Pin Attributes](#)
- [Terminal Attributes](#)
- [Bus Net Definition Attributes](#)
- [Layer-Purpose Pair Attributes](#)
- [Attributes of Objects and Figures](#)
-
- [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.

Related Topics

[dbGetq, dbGet](#)

Retrieving the List of Attribute Names for a dbObject Using the ~> Operator

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.
- 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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

`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...
)
```

Arguments

<i>d_object</i>	<i>dbObject</i> of the object to be queried.
<i>St_AttrOrPropName</i>	The attribute or property name to be retrieved. It can be specified as a SKILL symbol or a string. For <code>dbGet</code> , this can also be a SKILL variable containing a symbol or a string that represents the attribute or property name.

Value Returned

<i>g_value</i>	Value of the attribute or property <i>St_AttrOrPropName</i> of object <i>d_Object</i> .

Examples

Retrieves the value of the *shapes* attribute associated with a cellview object *c* by any of the following:

<code>dbGetq(c shapes)</code>	<code>; Does not evaluate second argument</code>
<code>c~>shapes</code>	<code>; Equivalent to above</code>
<code>c~>"shapes"</code>	<code>; Equivalent to above</code>
<code>dbGet(c 'shapes)</code>	<code>; Evaluates both its arguments</code>
<code>str = 'shapes</code>	<code>; Assigns attribute name</code>
<code>dbGet(c str)</code>	<code>; "str" is evaluated to shapes</code>

Related Topics

[dbGet Functions](#)

[Retrieving the List of Attribute Names for a *dbObject* Using the `~>` Operator](#)

Retrieving the List of Attribute Names for a *dbObject* Using the `~>` Operator

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
```

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 or Values for dbObjects Using the ~> Operator

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.

```
<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)
```

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

```
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
)
)
```

Related Topics

dbGet Functions

Retrieving the List of Attribute Names for a *dbObject* Using the *~>* Operator

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.

Related Topics

[dbSetq, dbSet](#)

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.

Arguments

<i>d_object</i>	Database object.
<i>g_value</i>	The new value for the attribute or property <i>St_AttrOrPropName</i> .
<i>St_AttrOrPropName</i>	Specifies the name of the attribute or property to be set. It can be specified as a SKILL symbol or a string. For <code>dbSet</code> , this can also be a SKILL variable containing a symbol or a string that represents the attribute or property name.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Value Returned

<i>g_value</i>	Returns the new value if successful.
<i>nil</i>	Returns <i>nil</i> 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
```

Related Topics

dbSet Functions

Generic Attributes

Some attributes are generic to figures and shapes in the database. Generic attributes are listed in the following tables.

- [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 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](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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	I_objId	List of properties on this object.

Related Topics

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic Figure Attributes

Figure attributes are common to database figures (shapes and anyInst).

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
assocTextDisplays	no	I_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	I_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	I_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	I_objId	List of markers for this object.
matchPoints	yes	boolean	Indicates whether the figure's children have points that match the figure's points.
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	I_objId	List of text displays owned by this figure object.

Related Topics

[Generic Object Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

[Generic Figure Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic anyInst Attributes

Attribute	Modify	Type	Description
Note: include Generic Figure Attributes.			
baseName	yes	string	Base name of the instance. The <i>name</i> of an instance can contain a vector expression by which <i>baseName</i> and <i>numInst</i> are derived. <i>numInst</i> can also be defined at creation.
cellName	no	string	Cell name of the master for this instance.
instHeader	no	objId	Instance header that belongs to this instance.
instTerms	no	I_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.

Related Topics

[Generic Figure Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic Boundary Attributes

Attribute	Modify	Type	Description
Note: include <u>Generic Object Attributes</u> .			
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.

Related Topics

Generic Object Attributes

Generic Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic Blockage Attributes

Attribute	Modify	Type	Description
Note: include <u>Generic Object Attributes</u> .			
area	no	float	Area of the blockage in user units.
assocTextDisplays	no	I_objId	List of text displays with this blockage as their associate.
bBox	no	bBox	The blockage's bounding box.
children	no	I_objId	List of objects that have this blockage as their parent.
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	I_objId	List of group members for this blockage.
isOrthogonal	no	boolean	Indicates if the blockage is orthogonal. When <code>t</code> , the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When <code>t</code> , 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	I_objId	List of text displays owned by this blockage.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Related Topics

Generic Object Attributes

Generic Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic Via Attributes

Via attributes are common to via objects.

Attribute	Modify	Type	Description
Note: include <u>Generic Object Attributes</u> .			
objType is listed as: stdVia, customVia, or nil.			
bBox	no	bBox	bBox associated with this via.
children	no	I_objId	List of objects that have this via as their parent.
connRoutes	no	I_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	I_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	I_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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

[Generic Object Attributes](#)

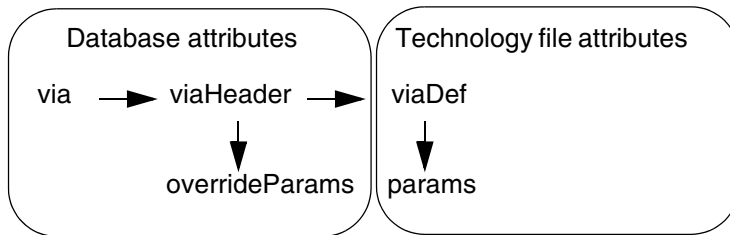
[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic Via Header Attributes

Finding Via Parameters



Via header attributes are common to via headers.

Attribute	Modify	Type	Description
Note: include <u>Generic Object Attributes</u> .			
viaDef	no	objId	Database ID for the technology file of the <code>viaDef</code> associated with this <code>viaHeader</code> . The return value is <code>nil</code> if the <code>viaHeader</code> is not bound.
viaDefName	no	string	Cached string attribute which returns the name of the associated <code>viaDef</code> . Usually available even if the <code>viaHeader</code> is not bound.
master	no	objId	ID of the cellview that is acting as the master of the <code>viaHeader</code> . Can be <code>nil</code> if the <code>viaHeader</code> is not bound.
subHeaders	no	l_objId	List of database IDs that identifies sub-via-Headers. The attribute value is <code>nil</code> if the <code>viaHeader</code> being queried is not a super- <code>viaHeader</code> .

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
vias	no	I_objId	Database ID of the vias associated with the viaHeader.
overrideParams	no	I_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 "

Related Topics

[Generic Object Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.
startingColor	no	string	Color of the first track. Valid values are "grayColor", "mask1Color", "mask2Color", and "mask3Color".

Related Topics

[Generic Object Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic widthSpacingPatternGroup Attributes

Note: Includes [Generic Object Attributes](#).

Attribute	Modify	Type	Description
name	no	string	Name of the object.
patternNames	no	I_patterns	List of pattern names in the group.

Related Topics

[Generic Object Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic relatedSnapPattern Attributes

Note: Includes [Generic Object Attributes](#).

Attribute	Modify	Type	Description
name	no	string	Name of the object.
relatedSnapPattern List	no	I_relatedsnap Patterns	List of related snap pattern specifications.
rowRegionSpecNa mes	no	I_rowRegion SpecNames	List of row region specification names.

Related Topics

[Generic Object Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.
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	I_snapLayers	List of snapping layer specifications.
patternNames	yes	I_string	List of pattern names in the snap pattern definition.
patternGroupNames	yes	I_string	List of pattern group names in the snap pattern definition.
defaultActiveName	yes	string	Name of the default active pattern.

Related Topics

[Generic Object Attributes](#)

[Generic Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Generic rowRegion Attributes

The rowRegion Attributes include [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 <code>t</code> .
rightBoundaryCell	yes	string	Name of the right boundary cell.
rightBoundaryCellMirrored	yes	Boolean	Specifies whether the right boundary cell is mirrored. The default is <code>nil</code> .
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 <code>nil</code> .
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 <code>t</code> .
topBoundaryCells	yes	<code>l_string</code>	List of top boundary cells.
bottomBoundaryCells	yes	<code>l_string</code>	List of bottom boundary cells.
tapCells	yes	<code>l_string</code>	List of tap cell names.
fillCells	yes	<code>l_string</code>	List of fill cell names.
libList	yes	<code>l_string</code>	List of boundary cell library names.
viewList	yes	<code>l_string</code>	List of boundary cell view names list. The default is <code>layout</code> .

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Row Attributes

Generic relatedSnapPattern Attributes

Generic Attributes

Virtuoso Studio Design Environment SKILL Reference
Attribute Retrieval and Modification Functions

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 <code>areaBoundaries</code> 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.
conns	no	l_objId	List of all connected instance terminals that exist in the cellview.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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.
layerHeaders	no	l_objId	List of all layerHeaders that exist in the cellview.
lib	no	objId	ID of the library that contains the cellview.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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.
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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
sitePattern	yes	l_objId	List of all sitePatterns that exist in the cellview. Setting a cellview sitePattern to <code>nil</code> is equivalent to deleting the sitePattern.
snapBoundary	no	l_objId	List of all boundaries of type <code>snapBoundary</code> 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).
viaMasters	no	l_objId	List of masters associated with all the viaHeaders in the cellview.
view	no	objId	View expressed as a database object.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
viewName	no	string	Name of the view.
widthSpacingSnapPatternDefs	no	I_objId	List of all widthSpacingSnapPatternDefs in the cellview.

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Instance Attributes

Design hierarchy is modeled through the placement of cell instances.

Attribute	Modify	Type	Description
Note: include Generic anyInst Attributes.			
conns	no	I_objId	List of all connected instance terminals for this instance (applicable only if the instance is not a mosaic).
clusters	yes	I_objId	List of all clusters that exist in the instance.
blockages	no	I_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
			"MX " mirrored on the x axis
			"MXR90 " mirrored on the x axis and rotated 90 degrees counterclockwise

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
physOnly	no	boolean	When <code>t</code>, 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 <code>physOnly</code> status from the cellview.
source	yes	string	The different sources you can assign to an instance. Valid Values: <code>netlist</code> <code>timing</code> <code>dist</code> <code>test</code> <code>user</code>
status	yes	string	Status of this instance for placement
			<code>suggested</code> the object has been placed but it may be placed again by future attempts
			<code>firm</code> placement of the object cannot be changed by an automatic placement tool
			<code>locked</code> placement of the object cannot be changed
			<code>placed</code> the object has been placed but it may be placed again by future attempts
			<code>unplaced</code> defined logically in database, but not physically placed in the design
transform	yes	<code>I_points</code>	List of values describing the instance's transformation with respect to the master. The list contains the following information in this order:

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
			offset Same as <i>xy</i> attribute
			orient Same as <i>orient</i> 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).

Related Topics

[Generic anyInst Attributes](#)

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	I_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.
timeStamp	no		Time stamps are no longer supported and have been replaced by counters. Return value is NULL.
variants	no	I_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.

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference
Attribute Retrieval and Modification Functions

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	I_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	I_objId	List of objects that have this mosaic as their parent.
groupMembers	no	I_objId	List of group members for this mosaic.
instHeader	no	string	Name of the instance header.
instanceList	no	I_objId	List of mosaic instances of masters used in the mosaic.
instTerms	no	I_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.
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	I_tile	Not supported. Row-major vector of mosaic tiles that have pairs of index and icon for the mosaic.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
uX	yes	I_float	List of distances (in user units) between adjacent columns.
uY	yes	I_float	List of distances (in user units) between adjacent rows.
viewName	yes	integer	Name of the view for this mosaic.

Related Topics

[Generic Object 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	I_objId	List of text displays with this instTerm as their associate.
connRoutes	no	I_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	I_objId	List of group members for this instTerm.
inst	no	objId	Instance to which the instTerm belongs.
markers	no	I_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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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	I_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.

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.			
<code>accessDir</code>	yes	<code>l_string</code>	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". <code>nil</code> means unspecified.
<code>accessDirection</code>	yes	<code>integer</code>	Value of <code>accessDir</code> 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). <code>nil</code> means unspecified and not 0.
<code>assocTextDisplays</code>	no	<code>l_objId</code>	List of text displays with this pin as their associate
<code>connRoutes</code>	no	<code>l_objId</code>	A list of all start/end connection of routes that connected to this pin.
<code>fig</code>	no	<code>objId</code>	The first figure that describes the pin.
<code>figs</code>	no	<code>l_objId</code>	Multiple pin figures are supported. <code>figs</code> will list all the figures of the pin when there is more than one.
<code>groupMembers</code>	no	<code>l_objId</code>	List of group members for this pin
<code>markers</code>	no	<code>l_objId</code>	List of all markers for this pin.
<code>name</code>	yes	<code>string</code>	Name of the pin. This is sometimes confused with the terminal name, which is used more often
<code>net</code>	yes	<code>objId</code>	Net attached to the pin.
<code>objType</code>	no	<code>string</code>	Type of object, in this case "pin"
<code>term</code>	no	<code>objId</code>	Terminal that contains the pin
<code>textDisplays</code>	no	<code>l_objId</code>	List of text displays owned by this pin

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
type	yes	string	Pin type of the <code>oaPin</code> object. <code>nil</code>: Specifies that the pin is a default pin, which is connected by normal "default" routing. <code>core</code>: Specifies the pin is a "core" ring connection point. <code>bump</code>: Specifies the pin is a "bump" connection point. A bump pin should only be connected to a bump.

Related Topics

[Generic Object Attributes](#)

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.			
assocTextDisplays	no	I_objId	List of text displays with this terminal object as their associate
connRoutes	no	I_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	I_objId	List of all group members for this terminal.
isInterface	no	boolean	Indicates whether this terminal is an interface terminal. See isInterface Attribute .
markers	no	I_objId	List of all markers for this terminal
mustJoinTerms	no	I_objId	Indicates that terminals on different nets must be connected. See Soft Connect Terminal Functions for a description of pin modeling. Value values are <code>nil</code> (not a mustJoinTerm) or a list of mustJoin terminals in that mustJoin group.
name	yes	string	Name of this terminal.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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".
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 <code>t</code> , 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 <code>physOnly</code> status from the cellview.
pins	no	I_objId	List of all pins on the terminal.
routeMethod	yes	string	Describes the route method. "geometric" "symbolic"
textDisplays	no	I_objId	List of text displays owned by the terminal object.

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 OpenAccess 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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

Generic Object Attributes

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.

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Net Attributes

Attribute	Modify	Type	Description
Note: Include Generic Object Attributes.			
allInstTerms	no	I_objId	List of instance terminals connected to this net and any of the net's subnets.
aliases	no	I_string	List of names depicting alternative names for the signal (no duplicates are printed).
assocTextDisplays	no	I_objId	List of text displays with this net object as their associate.
connStatus	yes	string	The status of net connections.
			"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	I_objId	List of all figures that use this net.
groupMembers	no	I_objId	List of all group members for this object.
instTermCount	no	integer	Number of instance terminals connected to the net.
instTerms	no	I_objId	List of all instTerms connected to the net.
isGlobal	yes	boolean	Defines whether the signal is known globally.
markers	no	I_objId	List of all markers associated with the net.

Virtuoso Studio Design Environment SKILL Reference
Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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.
numBits	no	integer	Number of signals in this net.
parent	no	objId	Parent net if this is a child net, otherwise <code>nil</code> .
pins	no	I_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	I_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	I_objId	List of signals associated with this net.
sigNames	no	I_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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
			"ground" The reference signal for the cellview
			"optical" An optical signal
			"singleModeOptical" A single mode optical signal
			"multiMode" A multimode optical signal
			"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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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 one terminal is not supported in DFII and indicates a shorted terminal.
textDisplays	no	l_objId	List of text displays owned by this net object.

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Signal Attributes

Attribute	Modify	Type	Description
Note: Include Generic Object Attributes.			
aliases	no	I_string	List of names depicting alternative names for the signal (no duplicates are printed).
groupMembers	no	I_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 to.
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
			"scan" Part of a test scan chain
			"signal" An ordinary signal

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
			"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

Related Topics

Generic Object Attributes

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	I_objId	List of all markers on this layer purpose pair.
shapes	no	I_objId	List of all shapes with this layer and purpose.

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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
```

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Group Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
deleteWhen	yes	string	The values that can be used are: <code>deleteNever</code> The group itself is never automatically deleted no matter which group member deleted. <code>deleteFirst</code> The group itself is automatically deleted once the first group member is deleted. <code>deleteLast</code> 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	<code>I_objId</code>	List of group members contained in this group
isCollection	yes	boolean	<code>t</code> if the group is a collection, <code>nil</code> otherwise
isOrdered	yes	boolean	<code>t</code> if the groupMembers are ordered, <code>nil</code> otherwise
isSet	yes	boolean	<code>t</code> if the group is a set, <code>nil</code> otherwise
isUniqueName	yes	boolean	<code>t</code> if the group name is unique, <code>nil</code> otherwise
members	no	<code>I_objId</code>	List of group members contained in this group
name	yes	string	Name of this group
objects	no	<code>I_objId</code>	List of objects contained in this group (referred to by group members)

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Property Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
assocTextDisplays	no	I_objId	List of text displays with this property object as their associate.
enum	no	string	If the property's type is <i>fileName</i> , <i>ILExpr</i> , <i>NLPEExpr</i> , or <i>string</i> , it can be an enumerated property, which can assume only one of the values in this list. <i>Enum</i> 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 <i>float</i> , <i>int</i> , or <i>time</i> , this property is either <code>nil</code> or a list containing the lower and upper bound. The type of this attribute, therefore, depends on <i>valueType</i> .
value	yes	see =>	Value of this property. The type of this attribute depends on <i>valueType</i> .
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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
			"NLPExpr" NLP expression string
			"string" String
			"time" Time-valued string

Related Topics

Generic Object Attributes

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](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Area Blockage Attributes

Attribute	Modify	Type	Description
Note: include Generic Blockage Attributes.			
nPoints	no	I_point	Number of points for the blockage.
objType	no	string	Type of object, in this case "areaBlockage".
points	yes	I_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.

Related Topics

[Generic Blockage Attributes](#)

Area Halo Attributes

Attribute	Modify	Type	Description
Note: include Generic Blockage Attributes.			
nPoints	no	I_point	Number of points for the blockage.
offsets	yes	I_points	List of points for the offset of the area halo.
points	no	I_points	List of points of the blockage.
type	no	string	Type of blockage routing wiring fill slot pin feedthru screen

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
isSoft	no	boolean	Indicates the soft attribute for a derived blockage. An <code>areaHalo</code> with this attribute indicates that standard cells should not be placed within the halo, but later optimization phases can use the halo area.

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
```

Related Topics

[Generic Blockage Attributes](#)

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	I_objId	LayerHeader for the layer of the blockage.
nPoints	no	I_point	Number of points for the blockage.
objType	no	string	Type of object, in this case "layerBlockage".
points	yes	I_points	List of points of the blockage.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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
```

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
```

Related Topics

Generic Blockage Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Layer Halo Attributes

Attribute	Modify	Type	Description
Note: include Generic Blockage Attributes.			
nPoints	no	I_point	Number of points for the blockage.
points	yes	I_points	List of points of the blockage.

Related Topics

[Generic Blockage Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <code>t</code> , the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When <code>t</code> , the blockage is a rectangle.

Related Topics

[Generic Boundary Attributes](#)

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 <code>t</code> , the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When <code>t</code> , the blockage is a rectangle.
name	yes	string	Name of boundary.
objType	no	string	Type of object "clusterBoundary".

Related Topics

[Generic Boundary Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <i>t</i> , the blockage is orthogonal.
isRectangle	no	boolean	Indicates if the blockage is rectangular. When <i>t</i> , 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
```

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Related Topics

[Generic Boundary Attributes](#)

Snap Boundary Attributes

Attribute	Modify	Type	Description
Note: include Generic Boundary Attributes.			
objType	no	string	Type of object "snapBoundary".

Related Topics

[Generic Boundary Attributes](#)

Cluster Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
blockages	no	I_objId	List of text displays with this cluster as their associate.
boundaries	no	I_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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
groupMembers	no	I_objId	List of group members for this object.
instances	no	I_objId	List of instances that are direct members of this cluster. Instances can be added to or removed from a cluster using <i>instanceId~>cluster</i> .
isParent	no	boolean	Indicates whether the cluster has other clusters (child-clusters or sub-clusters) as its members.
markers	no	I_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 <i>nil</i> to break a parent-child relationship.
subClusters	no	I_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 <i>clusterId~>parent</i> .

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

figGroup Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
assocTextDisplays	no	I_objId	List of text displays with this figure object as their associate.
bBox	no	bBox	The figure's bounding box.
children	no	I_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	I_objId	List of all figures that are members of this figGroup.
groupMembers	no	I_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	I_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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <code>t</code> , members in the figGroup will maintain the same order as when they were added. When <code>nil</code> , members can be removed in any order.
textDisplays	no	I_objId	List of text displays owned by this figure object.
type	no	string	Type of figGroup: none modgen syncClone placeArea rowFigure

Related Topics

[Generic Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <code>t</code> will define columns, and GCellPatterns with isHorizontal set to <code>nil</code> 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 <code>t</code> will be the left X coordinate of the first column and the startCoord of the GCellPattern with isHorizontal set to <code>nil</code> will define the bottom Y coordinate of the lowest row.
textDisplays	no	l_objId	List of text displays owned by the object.

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Guide Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
endPoint	yes	I_points	Ending point of the guide.
startPoint	yes	I_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.

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Marker Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
assocTextDisplays	no	I_objId	List of text displays with this figure object as their associate.
bBox	yes	bBox	The figure's bounding box.
children	no	I_objId	List of shapes that have this object as their parent.
deleteWhen	yes	string	Indicates when markers are automatically deleted. When modifying the attribute, <code>t</code> is returned if the <code>deleteWhen</code> value is set successfully. Returns <code>nil</code> if the set is unsuccessful.
			<code>deleteNever</code> Delete the marker only by explicit deletion.
			<code>deleteFirst</code> Delete the marker when any associated object is removed.
			<code>deleteLast</code> Delete the marker when all associated objects are removed.
			<code>deleteModify</code> Delete the marker when any associated object is modified.
groupMembers	no	I_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	I_objId	List of object database IDs associated with the marker.
parent	yes	objId	Parent figure of the marker.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
tool	yes	string	Indicates name of the tool associated with the marker.
textDisplays	no	I_objId	List of text displays owned by the terminal object.

Related Topics

Generic Object Attributes

Route Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
assocTextDisplays	no	I_objId	List of text displays with this figure object as their associate.
bBox	no	bBox	The figure's bounding box.
children	no	I_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	I_objId	List of group members for this object.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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 <code>nil</code> .
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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
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: <pre>(l_routeElement1 l_routeElement2 l_routeElement3 ... l_routeElementN)</pre> 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: <pre>('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)</pre>
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 <code>nil</code>. Valid start/end objects: steiner, steinerPin, pin, term, instTerm, shape, route.
textDisplays	no	l_objId	List of text displays owned by the terminal object.

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <code>siteDef</code> 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	I_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.

Related Topics

[Generic Figure Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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	I_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.

Related Topics

[Generic Object Attributes](#)

Steiner Attributes

Attribute	Modify	Type	Description
Note: include Generic Object Attributes.			
bBox	yes	bBox	bBox associated with the specified steiner
connRoutes	no	I_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
```

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Steiner pin attributes can be modified as follows:

```
steinerPinId~> attributeName = attributeValue
```

Example:

```
steinerId~>layer => t_layerNum  
steinerId~>layer = tx_layerNum => t
```

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <code>t</code> , the track pattern is horizontal.
markers	no	l_objId	List of markers associated with the track pattern.
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 <code>nil</code> , 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.

Related Topics

Generic Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

Generic Via Attributes

Virtuoso Studio Design Environment SKILL Reference
Attribute Retrieval and Modification Functions

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 <pre>list("layer1Enc" '(0.25 0.75))</pre>
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 <pre>list("layer2Enc" '(0.25 0.75))</pre>

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <pre>list("imp1Enc" '(0.4 0.6))</pre>
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 <pre>list("imp2Enc" '(0.4 0.6))</pre>
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 <pre>list("layer1Offset" '(0.1 0.2))</pre>

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
layer2Offset	yes	float	List of offsets for layer 2 in horizontal and vertical directions, in user units. See layer1Offset. Example <pre>list("layer2Offset" '(0.1 0.2))</pre>
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 <pre>list("cutSpacing" '(2.5 2.5))</pre>
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 <pre>list("originOffset" '(0.5 0.5))</pre>

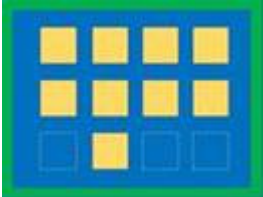
Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
cutLayer	yes	Integer	Name of the cut layer. Example <pre>list("cutLayer" "Cont1")</pre>
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 <pre>list("cutRows" 3)</pre>
cutRows	yes	integer	Number of rows in the array of cut. See cutColumns. Example <pre>list("cutRows" 3)</pre>
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 <pre>list("cutWidth" 1.3)</pre>
cutHeight	yes	float	Height of the cut layer. See cutWidth. Example <pre>list("cutHeight" 1.4)</pre>

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 <pre>list("cutPattern" "4fff")</pre> 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. 

Related Topics

[Generic Via Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Custom Via Header Attributes

Attribute	Modify	Type	Description
Note: include Generic Via Header Attributes.			

Related Topics

[Generic Via Header Attributes](#)

Standard Via Header Attributes

Attribute	Modify	Type	Description
Note: include Generic Via Header Attributes.			

Related Topics

[Generic Via Header Attributes](#)

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	I_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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
viaDefName	no	objId	Name of the viaDef associated with the given viaVariant.

Related Topics

Generic Object Attributes

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	I_objId	List of viaVariant parameter values.
paramslist	no	I_objId	List of viaVariant and values in the given syntax: <code>((("param1" value1) ("param2" value2)...) ...)</code>
viaDef	no	objId	ID of the viaDef associated with the given viaVariant.
viaDefName	no	objId	Name of the viaDef associated with the given viaVariant.

Related Topics

Generic Object Attributes

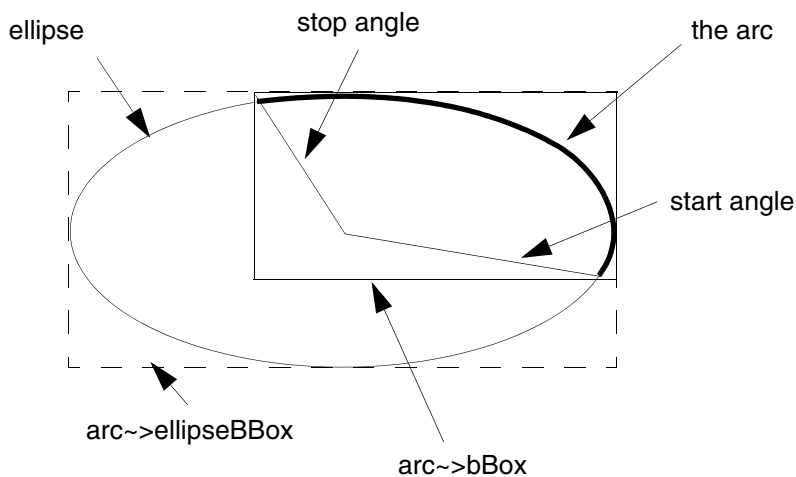
Attributes

- [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 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.



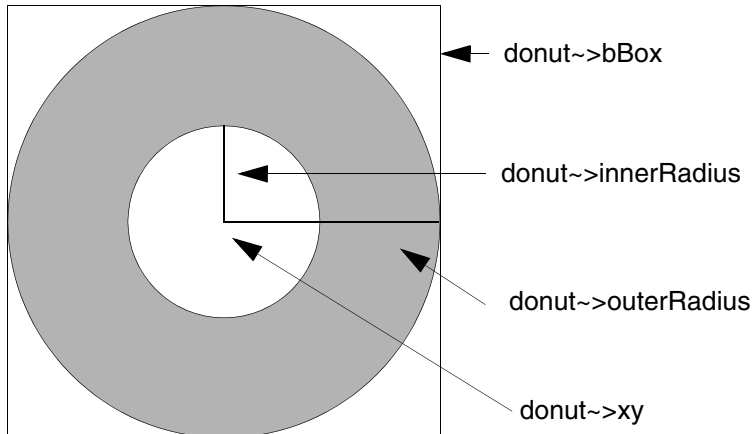
Attribute	Modify	Type	Description
Note: include Generic Shape Attributes.			
ellipseBBox	yes	bBox	Bounding box of the ellipse containing the arc.
startAngle	yes	float	Starting angle of the arc (in radians).
stopAngle	yes	float	Stopping angle of the arc (counterclockwise in radians).

Related Topics

Generic Shape Attributes

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 <i>innerRadius</i>).
xy	yes	float	Point describing donut's center (in user units).

Related Topics

[Generic Shape Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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).

Related Topics

[Generic Shape Attributes](#)

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.			

Related Topics

[Generic Shape Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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).
isDrafted	yes	boolean	<code>t</code> if the label reads left to right when horizontal and bottom to top when vertical; the label is always readable. <code>nil</code> if the label can appear upside down and backwards if the orient is set properly.
isOverbar	yes	boolean	<code>t</code> if the label has a bar on top of the text. <code>nil</code> otherwise.
isVisible	yes	boolean	Label is visible if set to <code>t</code> .
justify	yes	string	"lowerLeft" lower left corner of the label

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
			"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"
orient	yes	string	Orientation of this label:
			"R0 " not rotated
			"R90 " rotated 90 degrees
			"R180 " rotated 180 degrees
			"R270 " rotated 270 degrees

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
			"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).

Related Topics

[Generic Shape Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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).

Related Topics

Generic Shape Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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	I_point	List of points for the path centerline (in user units).
width	yes	float	Width of the path (in user units).

Related Topics

[Generic Shape Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

pathSeg Attributes

Attribute	Modify	Type	Description
Note: include Generic Shape Attributes.			
boundary	no	I_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	I_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	I_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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Related Topics

Generic Shape Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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	I_point	List of points for the polygon (in user units).

Related Topics

[Generic Shape Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

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
			"script" Script font
			"stick" Stick font
			"swedish" Swedish font
height	yes	float	Height of the text display (in user units)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
isDrafted	yes	boolean	<code>t</code> if the text display reads left to right when horizontal and bottom to top when vertical; the text display is always readable. <code>nil</code> 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 <code>t</code> . Only applicable if associate is a property/attribute.
isOverbar	yes	boolean	<code>t</code> if the text display has a bar on top of the text, else <code>nil</code> .
isValueVisible	yes	boolean	Value portion is visible if set to <code>t</code> . Only applicable if associate is a property/attribute.
isVisible	yes	boolean	Text display is visible if set to <code>t</code> .
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
orient	yes	string	Orientation of this label:
			"R0" not rotated

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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 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).

Related Topics

[Generic Shape Attributes](#)

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~>??
```

Generic Technology File Object Attributes

Attribute	Modify	Type	Description
groupMembers	no	I_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	I_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	I_objId	List of displays in the technology file
layers	no	I_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	I_objId	List of purposes in the technology file
timeStamp	yes	string	Time stamp of the technology file

Related Topics

Administrative Functions

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Layer Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes.			
blink	yes	boolean	Layer's blink attribute
brightColors	yes	I_objId	List of bright colors for the layer
displays	no	I_objId	List of displays that the layer references
fillColors	yes	I_objId	List of fill colors for the layer
fillStyle	yes	string	Layer fill style. One of the following:
			"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	I_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	I_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	I_objId	List of stipples for the layer
techFile	no	objId	Techfile the layer belongs to
twoLayers	no	I_objId	List of two layer relationships for the layer
visible	yes	boolean	Layer's visibility

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Related Topics

Generic Technology File Object Attributes

Administrative Functions

Display Attributes

Attribute	Modify	Type	Description
Note: include Generic Technology File Object Attributes.			
colors	no	I_objId	List of colors in the display
lineStyles	no	I_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	I_objId	List of stipples in the display
techFile	no	objId	Techfile the layer belongs to

Related Topics

Generic Technology File Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Related Topics

[Generic Technology File Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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

Related Topics

[Generic Technology File Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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

Related Topics

[Generic Technology File Object Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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
layer2	no	objId	Second of the two layers that are related
objType	no	string	Type of the object: "twoLayer"

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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Purpose	Description
pin	Object is associated with a pin, no longer used
tool1	
tool0	
unknown	
warning	

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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Layer	Description
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
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

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Layer	Description
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
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

Related Topics

Generic Technology File Object Attributes

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Attribute	Modify	Type	Description
params	no	l_params	List of via parameters.

Related Topics

[Technology File Attributes](#)

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. <code>t</code> allows the instance to be placed as R0 or MX.
symmetricInY	yes	boolean	Indicates the symmetry in the Y direction. <code>t</code> allows the instance to be placed as R0 or MY.
symmetricInR90	yes	boolean	Indicates the 90 degree rotation. <code>t</code> 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>??
```

Related Topics

[Technology File Attributes](#)

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Virtuoso Studio Design Environment SKILL Reference

Attribute Retrieval and Modification Functions

Design Management Functions

This topic provides a list of Cadence® SKILL functions associated with design management.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- [ddAmUsingDM](#)
- [ddAutoCheckin](#)
- [ddAutoCheckout](#)
- [ddAutoCtlGetVars](#)
- [ddAutoCtlSetVars](#)
- [ddCatRefresh](#)
- [ddCheckin](#)
- [ddCheckout](#)
- [ddClearLibUnmanaged](#)
- [ddCreateLib](#)
- [ddDeleteObj](#)
- [ddDeleteLocal](#)
- [ddDumpLibList](#)
- [ddGetCoManagedSet](#)
- [ddGetCombineValue](#)
- [ddGetDisplayValue](#)
- [ddGetForcedLib](#)
- [ddGetForcedLibEx](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- [ddGetLibList](#)
- [ddSetForcedLibEx](#)
- [ddSetLibUnmanaged](#)
- [ddSetLibDmType](#)
- [ddSetTmpdirWarn](#)
- [ddUnregHiddenCellsFunc](#)
- [ddUnregTrigger](#)
- [ddUpdateLibList](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddAmUsingDM

```
ddAmUsingDM(  
    b_objId  
)  
=> t / nil
```

Description

Uses a Design Management system if the library containing *b_objId*.

Arguments

<i>b_objId</i>	ID of an object.

Value Returned

t	The library containing <i>b_objId</i> uses a Design Management system.
nil	Object was not found or is not within a library under Design Management control.

Examples

```
ddAmUsingDM( myViewFile ) => t
```

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

<i>l_ddIds</i>	A ddd for a single object or a list of dddIds, 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.
<i>g_isBatch</i>	The batch override. Set to <code>t</code> 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 <code>nil</code> by virtually all applications.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Values Returned

l_list

A list of elements 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.

ddAutoCheckout

```
ddAutoCheckout (
    l_ddIds
    [ g_isBatch( { t | nil } ) ]
)
=> l_list
```

Description

Checks out an object or list of objects automatically. 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

<i>l_ddIds</i>	A ddd for a single object or a list of ddd's, 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.
<i>g_isBatch</i>	The batch override. Set to <code>TRUE</code> 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 <code>FALSE</code> by virtually all applications. It will be used by the SKILL “~>” operator in the case of setting a <code>propBag</code> parameter in a bag that has not been checked out.

Value Returned

<i>l_list</i>	A list of elements, all being <code>t</code> or <code>nil</code> , corresponding to each ID being checked out. The matching index will be set to <code>TRUE</code> if the checkout was done successfully or if none was needed. The matching index will be set to <code>FALSE</code> 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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

`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.

Additional Information

The `PreAutoCheckout` trigger is a pre-trigger invoked to allow preparation of the user environment. It is invoked each time any object is accessed.

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Related Topics

[ddAutoCheckout](#)

ddAutoCtlGetVars

```
ddAutoCtlGetVars(  
    )  
    => l_settings
```

Description

Returns a list of four elements in the following order:

```
x_promptCkout  
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:

<code>x_promptCkout</code>	Prompt for check out for views or files.
<code>x_promptCkin</code>	Prompt for check in for views or files.
<code>x_AutoCkout</code>	Automatically check out views or files.
<code>x_AutoCkin</code>	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

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

<i>x_promptCkout</i>	Prompts check out for views or files.
<i>x_promptCkin</i>	Prompts check in for views or files.
<i>x_AutoCkout</i>	Automatically checks out views or files.
<i>x_AutoCkin</i>	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

Turns on prompt check out, prompt check in, automatic check out, and automatic check in for views and files

```
ddAutoCtlSetVars (3 3 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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
ddAutoCtlSetVars (0 0 3 3)
```

You can 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.

```
ddAutoCtlSetVars (0 0 3 0)
```

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

<i>b_catFile</i>	ID of a category file.

Value Returned

<i>t</i>	The in-memory copy is update.
<i>nil</i>	Execution of the function failed.

Example

```
catId = ddCatOpen(libId, "mycat.Cat" "r")  
ddCatRefresh(catId)
```

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

<i>g_files</i>	A single ddld or a list of ddlds of files to be checked in. The files can be from more than one library.
<i>t_description</i>	Description of what you changed after you checked out the file.
<i>t_switches</i>	Switches to pass to the <code>gdmci</code> command. The <code>c</code> switch cancels the <i>checkout</i> of the named files. Valid Value: <code>c</code>

Value Returned

<i>t</i>	The checkin succeeded.
<i>nil</i>	Execution of the function failed.

Example

```
ddCheckin( viewId "fixed short" ) => t
```

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

<i>g_files</i>	A single ddID or a list of ddIDs of files to be checked out. The files can be from more than one library.
<i>t_switches</i>	Checks out files as read-only (<code>r</code>) or specifies a particular version of the file to check out (<code>v</code>). You must always follow this switch with the version of the file to check out.

Value Returned

<code>t</code>	The checkout succeeded.
<code>nil</code>	Execution of the function failed.

Examples

```
ddCheckout( viewId "rv1.1" ) => t
ddCheckout( filesList ) => t
```


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

<code>b_libId</code>	The ID of a library object.

Value Returned

<code>t</code>	The DMTYPE is removed.
<code>nil</code>	The DMTYPE is not removed.

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

<code>t_libName</code>	Name of the library to create.
<code>t_libPath</code>	Absolute or relative directory name where the library is created. A relative path is interpreted relative to the current directory. If <code>t_libPath</code> is omitted, <code>nil</code> , or an empty string (""), the library is created in the current working directory.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

b_ddId The ID of the library.

Examples

```
TTL7400libraryId = ddCreateLib( "TTL7400_series" )
```

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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
```

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

<i>b_objectId</i>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .

Value Returned

<i>t</i>	Execution of the function succeeded.
<i>nil</i>	Execution of the function failed.

Example

```
ddDeleteLocal( TTL7400libraryId ) => t
```

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

<code>t</code>	Information about the library list is printed.

Examples

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddGetCoManagedSet

```
ddGetCoManagedSet (
    g_viewID
)
=> l_coManagedList / nil
```

Description

Returns the list of comanaged files for the specified view.

Arguments

<i>g_viewID</i>	A <i>viewId</i> returned by <code>ddGetObj</code> .

Value Returned

<i>l_coManagedList</i>	The list of comanaged files for the specified view.
<i>nil</i>	There was an error.

Examples

```
viewfileId=ddGetObj("mylib" "mycell" "schematic" "sch.oa" nil "w")
viewId=ddGetObj("mylib" "mycell" "schematic")
list=ddGetCoManagedSet(viewId)
```


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.

Arguments

<i>g_libID</i>	The ID of the combined library.

Value Returned

<i>l_libNames</i>	A list of names of libraries in the combined library, if the library specified has the <code>COMBINE</code> attribute assigned.
<i>nil</i>	The library was not found or did not have the <code>COMBINE</code> attribute assigned.

Example

```
when( libNames=ddGetCombineValue(libId)  
    printf("'s' combines libraries %L\n" libId~>name libNames)  
)
```

Related Topics

Setting Attributes on a Library

Creating Combined Libraries

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>g_libID</code>	The ID of the library.

Value Returned

<code>t_attrName</code>	The value of the <code>DISPLAY</code> attribute.
<code>nil</code>	The library was not found or did not have the <code>DISPLAY</code> attribute assigned.

Examples

```
when( attrName=ddGetDisplayValue(libId)  
    printf("Library '%s' has DISPLAY attribute '%s' assigned\n" libId->name attrName)  
)
```

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

<code>t_filePath</code>	The path to the <code>cds.lib</code> file or empty string if no forced <code>cds.lib</code> file was specified.
<code>nil</code>	The path to the <code>cds.lib</code> file is not found.

Example

```
ddGetForcedLib() => "/user/library/cds.lib"
```

Related Topics

[ddGetForcedLibEx](#)

[ddSetForcedLib](#)

[ddSetForcedLibEx](#)

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.

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

<i>g_boolean</i>	<code>t</code> or <code>nil</code> . The <code>t</code> option is deprecated. Pass <code>nil</code> to get the <code>cds.lib</code> file that is set as a forced file.

Value Returned

<i>t_filePath</i>	The path to the <code>cds.lib</code> file that is set (if <i>g_boolean</i> is <code>nil</code>), or an empty string ("") if the file is not set or if the deprecated <code>t</code> option was passed.

Example

```
ddGetForcedLibEx(nil)  
returns "/usr1/mnt4/cds.lib" or "" if it was not set.
```

Related Topics

[ddGetForcedLib](#)

[ddSetForcedLib](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddSetForcedLibEx

ddGetHiddenCellsFuncList

```
ddGetHiddenCellsFuncList(  
    )  
=> l_functions/nil
```

Description

Returns a list of registered hidden cell functions.

Arguments

None

Value Returned

<i>l_functions</i>	A list of registered functions applied to all the libraries followed by functions applied to specific libraries. The format is: <pre>list("func1" "func2" list("lib1" list("func3" "func4"))) list("lib2" list("func5")))</pre>
<i>nil</i>	No functions are registered.

Examples

Returns a list of registered functions and prints the statement when there is at least one function registered.

```
funcList=ddGetHiddenCellsFuncList()  
when(funcList != nil println("There are hidden cell functions registered"))
```

Related Topics

[ddRegHiddenCellsFunc](#)

[Hiding Cells and Showing Hidden Cells](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddGetLibList

```
ddGetLibList(  
    )  
=> l_libs/ nil
```

Description

Generates a list of the libraries in the `libList`.

Arguments

None

Value Returned

<code>l_libs</code>	A list with one element for each library in the <code>libList</code> . Each element of <code>l_libs</code> is the ddId of a library.

<code>nil</code>	Execution of the function failed.

Examples

```
ddGetLibList( ) => (dd:2543240 dd:2611120 )  
(ddGetLibList) ~> name => ("cdsDefTechLib" "13")
```

ddsGetLibManLogFileName

```
ddsGetLibManLogFileName(  
    [g_query]  
)  
=> t_logPath / l_dpl / nil
```

Description

Returns the path to the log file created by Library Manager.

Arguments

<i>g_query</i>	Specifies the query for the current or previous log file path created by Library Manager. When query is <code>nil</code> or a string, or if no argument was specified, then the SKILL API returns a string for the log file path, or <code>nil</code> for no file. When query is a Disembodied property list, a new DPL is returned containing multiple values.

Values Returned

<i>t_logPath</i>	The path to the log file created by Library Manager.
<i>l_dpl</i>	Disembodied property list containing recognized property names from query DPL. Returns the following values: ■ <code>log</code>: Returns <code>nil</code> or the string for the path to the log file. ■ <code>pid</code>: Returns <code>nil</code> or the integer process ID of the active Library Manager. ■ <code>logPid</code>: Returns <code>nil</code> or the integer process ID of the Library Manager used to create the log file.
<i>nil</i>	No log file exists.

Examples

The following examples show queries for the path of the current or previous Library Manager log file.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
ddsGetLibManLogFileName()  
=>nil  
ddsGetLibManLogFileName( nil )  
=>nil  
ddsGetLibManLogFileName( "string" )  
=>nil
```

This following example shows query with DPL input to consider only an active Library Manager log file

```
detailsDpl = ddsGetLibManLogFileName( '( myTag log nil pid nil logPid nil ) )  
when( pairp( detailsDpl)  
    when( fixp( detailsDpl->pid ) && stringp( detailsDpl->log ) &&  
        detailsDpl->pid == detailsDpl->logPid  
        printf("Library Manager is running with log file %s\n" detailsDpl->log)  
    )  
))
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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 it 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 <code>ddUpdateLibList</code> 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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ❑ `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

<code>t_libName</code>	The library name to find the object you want. The named library must appear in the <code>libList</code> . If you do not specify <code>t_libName</code> (by specifying <code>nil</code> or <code>" "</code>) but you do specify <code>b_contextId</code> , DDPI takes the <code>libName</code> from the context object.
<code>t_cellName</code>	The name of a cell in the path to the object you are looking for. <code>ddGetObj</code> looks for the cell in <code>t_libName</code> . If you do not specify <code>t_cellName</code> (by specifying <code>nil</code> or <code>" "</code>) but you specify <code>b_contextId</code> , DDPI takes the <code>cellName</code> from the context object.
<code>t_viewName</code>	The name of a view in the path to the object you are looking for. <code>ddGetObj</code> looks for the view in <code>t_libName</code>, <code>t_cellName</code>. If you do not specify <code>t_viewName</code>, but you do specify <code>b_contextId</code>, DDPI takes the <code>viewName</code> from the context object. Note: <code>t_viewName</code> 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.
<code>t_fileName</code>	The name of the file object that you are looking for. <code>ddGetObj</code> looks for the file in <code>t_libname</code> only.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ❑ `wḍ`
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.
- ❑ `aḍ`
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 `wḍ`.
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`, `wḍ`, `a` or `aḍ` 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	<code>t_libName</code> or <code>b_contextId</code> .

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

a library's file	<i>t_libName*</i> or <i>b_contextId</i> and <i>t_fileName</i> .
a cell	<i>t_libName*</i> or <i>b_contextId</i> and <i>t_cellName</i> .
a cell's file	<i>t_libName</i> , <i>t_cellName</i> , and <i>t_fileName</i> or <i>b_contextId</i> (of a library), <i>t_cellName</i> , <i>t_fileName</i> or <i>b_contextId</i> (of an object below the cell), <i>t_fileName</i> .
a view	<i>t_libName*</i> , <i>t_cellName*</i> , and <i>t_viewName</i> or <i>b_contextId</i> (of a library), <i>t_cellName</i> , and <i>t_viewName</i> or <i>b_contextId</i> (of an object below the cell, and <i>t_viewName</i> .
a view's file	<i>t_libName</i> , <i>t_cellName</i> , <i>t_viewName</i> , and <i>t_fileName</i> or <i>b_contextId</i> (of a library), <i>t_cellName</i> , <i>t_viewName</i> , and <i>t_fileName</i> or <i>b_contextId</i> (of an object below the cell), <i>t_viewName</i> , and <i>t_fileName</i> or <i>b_contextId</i> (of a view), and <i>t_fileName</i> .

* 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

<i>b_ddId</i>	An ID for the requested object.
<i>nil</i>	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)
```

Additional Information

The *ObjAccess* trigger is a pre-trigger invoked to allow preparation of the user environment. It is invoked each time any object is accessed.

The *PreObjAccess* should avoid calling *ddGetObj()* on any object because it would be recursive and dangerous.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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*. 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.

Related Topics

[ddGetObj](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_objId</i>	Object ID returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .
<i>t_which</i>	The values for <i>which</i> may be <i>o</i> (owner), <i>g</i> (group), and <i>p</i> (public).

Value Returned

<i>t_access</i>	A string consisting of the letters <i>r</i> , <i>w</i> , and <i>x</i> corresponding to UNIX usage.
<i>nil</i>	The string is empty if there are no permissions.

Examples

```
ddGetObjAccess ( TTL7400libraryId "o" ) => "rwx"
```

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.

Arguments

<i>b_objId</i>	Object ID returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .
<i>t_which</i>	One of the following values: <code>o</code> (owner), <code>g</code> (group), or <code>p</code> (public).
<i>x_pathType</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<i>t_access</i>	A string consisting of the letters <i>r</i> , <i>w</i> , and <i>x</i> corresponding to UNIX usage.
<i>nil</i>	There are no permissions.

Related Topics

[ddGetObjAccess](#)

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

<i>b_objId</i>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .

Value Returned

<i>l_children</i>	The IDs of the children of <i>b_objId</i> .
<i>nil</i>	<i>b_objId</i> does not exist or does not have any children.

Examples

```
l = ddGetObj( "l1" )  
ch = ddGetObjChildren( l ) => (dd:2660660 dd:2660628)  
ch ~> name => ("c2" "c1")
```

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

<i>b_objId</i>	Object ID returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .
<i>x_pathType</i>	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

<i>l_children</i>	The IDs of the children of <i>b_objId</i> .
<i>nil</i>	<i>b_objId</i> does not exist or does not have any children.

Related Topics

[ddGetObjChildren](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddGetObjDMSys

```
ddGetObjDMSys (
    b_objId
)
=> t_sysName / nil
```

Description

Returns the name of the Design Management system of *b_objId*.

Arguments

<i>b_objId</i>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .

Value Returned

<i>t_sysName</i>	The name of the Design Management system.
<i>nil</i>	The Design Management system name could not be found.

Example

```
ddGetObjDMSys ( myView )
=> "tdm"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_objId</i>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .

Value Returned

<i>l_files</i>	All of the files connected to the specified object.
<i>nil</i>	No files were found.

Examples

```
l = ddGetObj( "l1" )  
fl = ddGetObjFiles( l ) => (dd:2660732 dd:2660692)  
fl ~> name => ( "test.ckpt" "cdsinfo.tag" )
```

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

<i>b_objId</i>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .
<i>x_pathType</i>	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

<i>l_files</i>	All of the files connected to the specified object.
<i>nil</i>	No files were found.

Related Topics

[ddGetObjFiles](#)

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 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

<i>b_objId</i>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .

Value Returned

<i>x_int</i>	The group ID as an integer.
<i>nil</i>	No group was found.

Examples

```
ddGetObjGroup( 1 ) => 12
```

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

<i>b_objId</i>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .
<i>x_pathType</i>	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

<i>x_int</i>	The group ID as an integer.
<i>nil</i>	No group was found.

Related Topics

[ddGetObjGroup](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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 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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t_time</i>	The last time the object was modified.
nil	Object was not found.

Examples

```
ddGetObjLastModify( 1 ) => "Jan 16 10:45:10 1996"
```

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

<i>b_objId</i>	Object ID returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .
<i>x_pathType</i>	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

<i>t_time</i>	The last time the object was modified.
<i>nil</i>	Object was not found.

Related Topics

[ddGetObjLastModify](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t_lib</i>	The library ID.
nil	Object was not found.

Example

```
libId = ddGetObjLib( cellId ) => dd:2544064  
libId ~> name => "l1"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddGetObjName

```
ddGetObjName (
    b_objId
)
=> t_name / nil
```

Description

Returns the name of *b_objId*.

Arguments

<i>b_objId</i>	ID of the object.

Value Returned

<i>t_name</i>	Name of the object.
<i>nil</i>	Object was not found.

Example

```
ddGetObjName ( 1 ) => "l1"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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 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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t_owner</i>	The owner's name.
nil	An invalid object ID was given.

Examples

```
ddGetObjOwner( 1 ) => "guest"
```

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

<i>b_objId</i>	ID of the object.
<i>x_pathType</i>	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

<i>t_owner</i>	The owner's name.
<i>nil</i>	An invalid object ID was given.

Related Topics

[ddGetObjOwner](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddGetObjParent

```
ddGetObjParent (
    b_objId
)
=> b_parent / nil
```

Description

Returns the ID of the logical (often physical as well) owner-object of *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

<i>b_objId</i>	ID of the object.

Value Returned

<i>b_parent</i>	The ID of the parent of the specified object.
<i>nil</i>	Object was not found.

Examples

```
p = ddGetObjParent( cell11 ) => dd:2544064
```

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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t_path</i>	The read path to <i>b_objId</i> .
nil	Object was not found.

Examples

```
f = ddGetObjReadPath( file1 ) => "/tmp/l1.lib/c1/v1/sch.oa"
```

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

<i>b_objId</i>	ID of the object.
<i>x_pathType</i>	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

<i>t_path</i>	The read path to <i>b_objId</i> .
<i>nil</i>	The object was not found.

Related Topics

[ddGetObjReadPath](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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 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

<i>b_objId</i>	ID of a file object.

Value Returned

<i>x_size</i>	The file length, in bytes.
-1	The object is not a file object.

Example

```
ddGetObjSize( myViewFile ) => 1705
```

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

<i>b_objId</i>	ID of a file object.
<i>x_pathType</i>	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

<i>x_size</i>	The file length, in bytes.
-1	The object is not a file object.

Related Topics

[ddGetObjSize](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t_path</i>	The write path of <i>b_objId</i> .
nil	Object was not found.

Example

```
ddGetObjWritePath( f ) => "/tmp/l1.lib/c1/v1/sch.oa"
```

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

<i>b_objId</i>	ID of the object.
<i>x_pathType</i>	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

<i>t_path</i>	The write path of <i>b_objId</i> .
<i>nil</i>	The object was not found.

Related Topics

[ddGetObjWritePath](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_path</i>	The path name of the file or directory.

Value Returned

<i>t_dmSysStrin</i>	The name of the system.
<i>nil</i>	Object was not found.

Example

```
ddGetPathDMSys("/usr/home/libs/libx/cell") returns tdm
```

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

<i>t_fileName</i>	The first filename found.
<i>b_contextId</i>	The function changes the working directory to the one implied by <i>b_contextId</i> (if it differs from the current working directory).

Value Returned

<i>t_filePath</i>	The path to the first file found.
<i>nil</i>	The filename was not found.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Example

```
ddGetStartup( "cds.lib" )  
=> "/home/guest/cadence/cds.lib"
```

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

<code>t</code>	Current setting is set to <code>t</code> , implying that the warning will be printed.

<code>nil</code>	Current setting is set to <code>nil</code> , 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 suppress warning  
(Example: by ddSetTmpdirWarn(nil))
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Related Topics

[ddSetTmpdirWarn](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddGetTriggerList

```
ddGetTriggerList(  
    t_triggerName  
)  
=> l_triggerFunc
```

Description

Returns a list of functions registered to be executed when *t_triggerName* is activated.

Arguments

<i>t_triggerName</i>	The pre- or post-trigger name.

Value Returned

<i>l_triggerFunc</i>	A list of trigger Descriptions, in the format: <i>function</i> The trigger function name. <i>t / nil</i> Indicating whether the function is a SKILL function (<i>t</i> = SKILL). <i>integer</i> The function priority.

Example

```
ddGetTriggerList( "preupdateliblist" )  
=> (( "testUpdateLibListPreTrigger" t 2147483647 ))
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_filePath</i>	The path to the <code>cds.lib</code> file that was last used by <code>ddUpdateLibList</code> .

Examples

```
ddGetUpdatedLib( )  
=> "/install/home/cds.lib"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t</i>	<i>b_objId</i> is an opaque ID known to the PI.
<i>nil</i>	The argument is not a valid <i>ddId</i> .

Examples

```
ddIsId( myCell ) => t
```


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

<i>b_cellId</i>	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)  
)
```

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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t</i>	The library that owns <i>b_objId</i> is marked to be deleted or its path is changed.
<i>nil</i>	The object could not be found, or the object is not marked.

Example

```
ddIsObjLibMarked( myCell ) => t
```

ddlIsObjReadable

```
ddlIsObjReadable(  
    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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t</i>	<i>b_objId</i> can be read by the current process.
<i>nil</i>	Object was not found or is not readable.

Example

```
ddlIsObjReadable( myViewFile ) => t
```

Related Topics

[ddlIsObjReadable](#)

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

<i>b_objId</i>	ID of the object.
<i>x_pathType</i>	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

<i>t</i>	<i>b_objId</i> can be read by the current process.
<i>nil</i>	The object was not found or is not readable.

Related Topics

[ddIsObjReadable](#)

ddlObjWritable

```
ddlObjWritable(  
    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 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. [ddlObjWritable](#)

Arguments

<i>b_objId</i>	ID of the object.

Value Returned

<i>t</i>	<i>b_objId</i> can be modified by the current process.
<i>nil</i>	Object was not found or is not writable.

Examples

```
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  
  
ddlObjWritable(ddGetObj("libFolder"))  
=> nil
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Related Topics

[ddlsObjReadableEx](#)

ddlObjWritableEx

```
ddlObjWritableEx(  
    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 `ddlObjWritable` 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

<i>b_objId</i>	ID of the object.
<i>x_pathType</i>	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

<i>t</i>	<i>b_objId</i> can be modified by the current process.
<i>nil</i>	The object was not found or is not writable.

Related Topics

[ddlObjReadableEx](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddlIsPathManageable

```
ddlIsPathManageable(  
    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

<i>t_path</i>	The supplied path.

Value Returned

<i>t</i>	The supplied path is manageable.
<i>nil</i>	The supplied path is not manageable.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddlIsPathManaged

```
ddlIsPathManaged(  
    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 `ddlIsPathManaged` 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

<i>t_path</i>	The specified path.

Value Returned

<i>t</i>	The specified path is managed.
<i>nil</i>	The specified path is not managed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddLibIs5X

```
ddLibIs5X(  
    b_libId  
)  
=> t / nil
```

Description

Determines if the library was created with the 5.X library structure.

Arguments

<i>b_libId</i>	ID of the library.

Value Returned

<i>t</i>	The library was created as a 5.X library.
<i>nil</i>	The library was not created as a 5.X library.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_objId</i>	ID of the library.

Value Returned

<code>t</code>	The library has a temporary directory.
<code>nil</code>	The library does not have a temporary directory.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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`.

Arguments

<i>b_lockId</i>	ID of a locked file as returned from <code>ddLockPath</code> .

Value Returned

<i>t_message</i>	A message used by interactive programs to tell you why there is a delay in accessing a file.
<i>nil</i>	Object was not locked.

Examples

```
ddLockDescribe( myLibFileLock ) => "is currently locked by user  
fabio(pid:1234)@cds3028 (age:10 sec)."
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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*.

Arguments

<i>b_lockId</i>	ID of a locked file as returned from <i>ddLockPath</i> .

Value Returned

<i>t</i>	Releases the lock on the file.
<i>nil</i>	Lock could not be freed.

Examples

```
ddLockFree( myLibFileLock ) => t
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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`.

Arguments

b_lockId ID of a locked file as returned from `ddLockPath`.

Value Returned

<i>x_sysId</i>	Always returns 0.
<i>t_type</i>	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.
<i>x_value</i>	Always returns 0.
<i>t_userName</i>	If the lock type is anything except "Write" or "Exclusive", the return value is "UNKNOWN".
<i>t_userHost</i>	User name of host running user process. If the lock type is anything except "Write" or "Exclusive", the return value is "UNKNOWN".
<i>t_fileHost</i>	Always returns "UNKNOWN"
<i>t_filePath</i>	Path locked.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Examples

```
ddLockGetStatus( myFileLock ) => (0 "Write" 0 "guest" "big_sun" "UNKNOWN" "/tmp/  
my.file" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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`.

Arguments

<i>t_filePath</i>	The path to the file. The path must be a file (only files need to have locks).

Value Returned

<i>b_lockId</i>	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.

Examples

```
ddLockPath( "/tmp/myFile" ) => dd:2542360  
ddLockPath( ddGetObjWritePath( myViewFile ) ) => dd:2542360
```


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.

This function uses the Cadence locking system (CLS) instead of *cdsd*.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

<i>g_block</i>	When set to <code>t</code> (default), <code>ddLockSet</code> does not return until you get the lock you have requested. When set to <code>nil</code> , <code>ddLockSet</code> returns at once, whether or not it gets the lock. But you need to loop repeatedly calling <code>ddLockSet</code> until the process, which has the lock and is preventing you from getting yours, frees it.
<i>t_message</i>	A string you can set to explain why you want the lock, so other users will see it using <code>cdsdAdmin</code> . Defaults to <code>NULL</code> .

Value Returned

<code>t</code>	The lock was obtained, or <i>b_blockId</i> was the type of lock that you already had on the file.
<code>nil</code>	The lock was not obtained.

Example

```
ddLockSet( myFileLock "r" nil "Test lock" ) => t
```

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

<i>g_block</i>	When set to <code>t</code> (default), <code>ddLockSet</code> does not return until you get the lock you have requested. When set to <code>nil</code> , <code>ddLockSet</code> returns at once, whether or not it gets the lock. But you need to loop repeatedly calling <code>ddLockSet</code> until the process, which has the lock and is preventing you from getting yours, frees it.
<i>t_message</i>	A string you can set to explain why you want the lock, so other users will see the information. The default is <code>NULL</code> .
<i>g_doErrMsgs</i>	When <code>t</code> , warnings are reported.
<i>g_timeout</i>	Time out in seconds after which the function will quit trying to get a lock on the file if the <i>g_block</i> argument is <code>t</code> and another process has the lock.

Value Returned

<code>t</code>	The lock was obtained.
<code>nil</code>	The lock was not obtained.

Example

```
viewlmaster = ddGetObj("lib1" "cell1" "view1" "master.tag")
viewlmasterpath = ddGetObjReadPath(viewlmaster)
lockId = ddLockPath(viewlmasterpath)
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`

Related Topics

[ddLockSet](#)

ddMapGetDataTypeFileName

```
ddMapGetDataTypeFileName (
    t_dataType
)
=> t_fileName / nil
```

Description

Returns the pattern registered in the registry for *t_dataType*.

Arguments

<i>t_dataType</i>	A data type from the registry.

Value Returned

<i>t_fileName</i>	The pattern for filenames holding <i>t_dataType</i> .
<i>nil</i>	The data type is not found or has no registered pattern.

Examples

```
ddMapGetDataTypeFileName ( "schematic" )
=> "sch.oa"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddMapGetDataTypeInfoList

```
ddMapGetDataTypeInfoList(  
    )  
    => 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.

Examples

```
ddMapGetDataTypeInfoList()  
=> ("AHDNetlist" "AHDNetText" "AmadeusPackage" "AmadeusVerilogMap" "AsciiText"  
"CDBAGraphics" "CDBANetlist" "CDBAstranger" "ComposerPIC" "ComposerSchematic"  
"ComposerSymbol" "ConceptSchematic" "DesignFlow" "HierarchyDescription"  
"MaskLayout" "ThermaxBoundary" "ThermaxDesign" "ThermaxProject" "ThermaxResult"  
"VHDLNetText" "VerilogNetlist" "VerilogText")
```

ddMapGetFileDataType

```
ddMapGetFileDataType(  
    b_fileObjId  
)  
=> t_dataType / nil
```

Description

Returns the name of the data type associated with *b_fileObjId*.

Arguments

<i>b_fileObjId</i>	The object file ID.

Value Returned

<i>t_dataType</i>	The data type name.
<i>nil</i>	The data type is not found.

Example

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )  
ddMapGetFileDataType( vf ) => "ComposerSchematic"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_fileObjId</i>	The object file ID.

Value Returned

<i>t_fileTool</i>	File tool of the file, a permanent string.
nil	The file tool is not found.

Examples

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )  
ddMapGetFileTool( vf ) => "Composer-Schematic"
```


ddMapGetFileViewType

```
ddMapGetFileViewType(  
    b_fileObjId  
)  
=> t_viewType / nil
```

Description

Returns the name of the view type associated with *b_fileObjId*.

Arguments

<i>b_fileObjId</i>	The object file ID.

Value Returned

<i>t_viewType</i>	The view type name.
<i>nil</i>	The view type is not found.

Examples

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )  
ddMapGetFileViewType( vf ) => "schematic"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_toolName</i>	The name of the tool.

Value Returned

<i>t_dataType</i>	The data type name.
<i>nil</i>	The data type is not found.

Examples

```
ddMapGetToolNameDataType ( "Composer-Schematic" )
=> "ComposerSchematic"
```

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

<i>l_toolList</i>	A list of tool names.

Examples

```
ddMapGetToolNameList(  
=> ( "Amadeus-Package-Editor" ... "vi" )
```

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

<i>t_toolName</i>	Name of the tool from the registry.

Value Returned

<i>t_viewTypeName</i>	The view type name.
<i>nil</i>	The view type is not found.

Examples

```
ddMapGetToolNameViewType ( "Composer-Schematic" )
=> "schematic"
```

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

<i>t_viewType</i>	A view alias or data type name.

Value Returned

<i>t_fileName</i>	The filename.
<i>nil</i>	The view type is not found.

Examples

```
ddMapGetViewTypeFileName( "schematic" ) => "sch.oa"
```

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

<i>l_list</i>	A list of view type names.

Examples

```
ddMapGetViewTypeList( ) => ( "ConceptSchematic" ... "vhdl" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_fileName</i>	The file name from the registry.

Value Returned

<i>t</i>	The <i>t_fileName</i> is found in the registry.
<i>nil</i>	<i>t_fileName</i> was not found in the registry.

Examples

```
ddMapIsFileNameKnown( "sch.oa" ) => t
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_fileObjId</i>	The object file ID.
<i>t_toolName</i>	The tool name in the registry.

Value Returned

<i>t</i>	<i>t_toolName</i> is either the preferred editor or the default editor for <i>b_fileObjId</i> .
<i>nil</i>	The tool is not one of the tools specified for <i>b_fileObjId</i> .

Examples

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )  
ddMapIsToolLegal( vf "Virtuoso" ) => nil
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddMapIsToolNameKnown

```
ddMapIsToolNameKnown(  
    t_toolName  
)  
=> t / nil
```

Description

Returns *t* if *t_toolName* is defined in the registry.

Arguments

<i>t_toolName</i>	Name of the tool in the registry.

Value Returned

<i>t</i>	The tool is known.
<i>nil</i>	The tool name is not found.

Examples

```
ddMapIsToolNameKnown( "Composer" ) => t
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>b_objId</code>	ID of the object, which must be a <code>viewFileType</code> object.

Value Returned

<code>t</code>	The <code>master.tag</code> file was changed.
<code>nil</code>	The <code>master.tag</code> file could not be changed.

Examples

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )  
ddMarkObjMaster( vf ) => t
```

ddNeedCheckout

```
ddNeedCheckout (
    b_objId
)
=> t / nil
```

Description

Enables the tools that need to provide automatic checkout behavior to determine if these tools need to call the `ddCheckout` and `ddCheckin` functions.

Arguments

<i>b_objId</i>	The <code>ddId</code> for a file object.

Value Returned

<code>t</code>	<i>b_objId</i> is the DM-managed object to be checked out.
<code>nil</code>	The library is not managed by a Design Management system, <i>b_objId</i> does not need to be checked out, or <i>b_objId</i> exists only in the temporary directory of the library. Objects in the temporary directory cannot be managed.

Examples

```
vf = ddGetObj( "l1" "c1" "v1" "sch.oa" )
if ddNeedCheckout( vf ) then
    ddCheckout( vf )
```

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

<i>b_objId</i>	The object file ID.
<i>t_fileName</i>	The filename to search for.

Value Returned

<i>t</i>	The file was found.
<i>nil</i>	The file was not found.

Examples

```
l = ddGetObj ( "l1" ) => dd:2543176
ddObjHasNamedObj (l "c1") => t
ddObjHasNamedObj (l "v1") => nil
```

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.

Arguments

<i>b_objId</i>	The object file ID.
<i>t_fileName</i>	The file to search for.
<i>x_pathType</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

t	The file was found.
nil	The file was not found.

Related Topics

[ddObjHasNamedObj](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddObjIsInMaster

```
ddObjIsInMaster(  
    b_objId  
)  
=> t / nil
```

Description

Checks if the object exists in the master directory.

Arguments

<i>b_objId</i>	ID of the object.

Value Returned

<i>t</i>	The object is in the master directory.
<i>nil</i>	The object is not in the master directory.

Related Topics

[ddObjIsInTmp](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>b_objId</i>	ID of the object.

Value Returned

<i>t</i>	The object is in the temporary directory.
<i>nil</i>	The object is not in the temporary directory.

Related Topics

[ddObjIsInMaster](#)

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

<i>b_objId</i>	The parent object.
<i>g_consumer</i>	The <i>g_consumer</i> function should be of the form <pre>myClient(<i>t_objName</i>, <i>g_clientData</i>) => t/nil</pre>
	<i>t_objName</i> Is the name of the next objects (directory or file) being produced.
	<i>g_clientData</i> Is the client data that the producer was given.
<i>g_clientData</i>	Any arbitrary piece of data you want passed to the <i>g_consumer</i> .

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<code>t</code>	<code>g_consumer</code> returned <code>t</code> for each name passed to it.
<code>nil</code>	There are no objects or if the name production is interrupted by a <code>nil</code> return value of <code>g_consumer</code> .

Examples

Here is how you call this procedure from SKILL:

Sample client:

```
(procedure (myClient name arg)
  rplacd(last(arg) ncons(name)) t )
```

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" )
```

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.

Arguments

<i>b_objId</i>	The parent object.
<i>g_consumer</i>	The <i>g_consumer</i> function should be of the form: <pre>myClient(t_objName, g_clientData) => t/nil</pre> ❑ <i>t_objName</i> Is the name of the next objects (directory or file) being produced. ❑ <i>g_clientData</i> Is the client data that the producer was given.
<i>g_clientData</i>	Any arbitrary piece of data you want passed to the <i>g_consumer</i> .

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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*.

Related Topics

[ddProduceObjRelNames](#)

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

<i>t_hiddenCellsFunc</i>	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 <code>ddUpdateLibList</code> call is made.
<i>t_libName</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Example

Consider the `myHiddenCellsFunc` function, which hides cells that start with an underscore (`_`) in the `myLib` library:

```
procedure (myHiddenCellsFunc (libId)
  setof (cellName libId~>cells~>name strcmp (cellName "_" 1) == 0)
)
```

Register the `myHiddenCellsFunc` function as follows:

```
ddRegHiddenCellsFunc ('myHiddenCellsFunc "myLib")
```

Related Topics

[ddUpdateLibList](#)

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"

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

<i>s_triggerFunc</i>	The function you want to have called back when the trigger is activated.
<i>x_priority</i>	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

<i>t</i>	The function was successfully registered to trigger when the event occurs.
<i>nil</i>	Execution of the function failed.

Examples

```
ddRegTrigger( "PreCheckin" 'checkinTrigFunc 0 )  
=> t
```

Additional Information

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.

Note: 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

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Use the registration function, `ddRegTrigger`, to register a function to be called for any kind of triggering event.

- 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 )
```

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.

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`.

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

<i>b_objId</i>	The ID of the object you want to release.

Value Returned

t	The object was released.
nil	<i>b_objId</i> was not a valid object, or one of the PreDeleteObj triggers returns nil.

Examples

```
ddReleaseObj ( viewFileId ) => t
```

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.

`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

<code>t_path</code>	The path to the <code>cds.lib</code> file you want to use.

Value Returned

<code>t</code>	The path to the <code>cds.lib</code> file is valid.
<code>nil</code>	The path to the <code>cds.lib</code> file is not valid.

Example

```
ddSetForcedLib( "/user/library/cds.lib" ) => t
```

Related Topics

[ddUpdateLibList](#)

[ddSetForcedLibEx](#)

[ddGetForcedLibEx](#)

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.

There are a few differences between `cds.lib` and `lib.defs` format. For more information on `cds.lib`, see [Cadence Library Definition File](#).

The `lib.defs` are no longer supported by Virtuoso.

Arguments

<code>g_cdsLibFile</code>	The path to the <code>cds.lib</code> file to use.
<code>g_libDefsFile</code>	The path to the <code>lib.defs</code> file to use. If <code>cds.lib</code> is specified, this argument is ignored. If <code>cds.lib</code> is not specified, the specified <code>lib.defs</code> file will be handled as a <code>cds.lib</code> file. In some cases this could lead to warnings due to differences in format of <code>cds.lib</code> and <code>lib.defs</code> .

Value Returned

<code>t</code>	The file was set.
<code>nil</code>	There was an error.

Examples

```
ddSetForcedLibEx("../cds.lib" nil)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Related Topics

[ddSetForcedLib](#)

[ddGetForcedLibEx](#)

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

<code>b_libId</code>	The ID of a library object.

Value Returned

<code>t</code>	The DMTYPE entry is set.
<code>nil</code>	The DMTYPE entry is not set.

ddSetLibDmType

```
ddSetLibDmType (
    b_libId
    t_dmType
)
=> t / nil
```

Description

Sets the DMTYPE entry of the `cdsinfo.tag` file in the library.

Arguments

<i>b_libId</i>	The ID of a library object.
<i>t_dmType</i>	The Design Management system name to which the DMTYPE entry in the <code>cdsinfo.tag</code> file is being set.

Value Returned

<i>t</i>	The DMTYPE entry is set.
<i>nil</i>	The DMTYPE entry is not set.

Examples

Returns `t` when you set the DMTYPE entry in the `cdsinfo.tag` file for `lib1` to `"vs"`. Otherwise, returns `nil`.

```
libId = ddGetObj("lib1")
ddSetLibDmType(libId,"vs")
```

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

<i>g_doWarn</i>	Enables or disables the printing of the warning message. Valid Values: <code>t</code> (the current value will be set to <code>t</code> to enable printing of the warning) <code>nil</code> (the current value will be set to <code>nil</code> to disable printing of the warning)

Value Returned

<code>t</code>	Printing of the warning message is enabled.
<code>nil</code>	Printing of the warning message is disabled.

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

<code>t</code>	The function was unregistered.
<code>nil</code>	The function was not unregistered. The specified function might not exist or was not registered using <code>ddRegHiddenCellsFunc</code> .

Example

Unregister the `myHiddenCellsFunc` function as follows:

```
ddUnregHiddenCellsFunc('myHiddenCellsFunc)
```

Related Topics

[ddRegHiddenCellsFunc](#)

ddUnregTrigger

```
ddUnregTrigger(  
    t_triggerName  
    s_triggerFunc  
)  
=> t / nil
```

Description

Removes a callback (trigger function) from a named list of trigger functions.

Arguments

<i>t_triggerName</i>	Name of the trigger to remove the callback from. Refer to the <code>ddRegTrigger</code> definition for a list of the permissible values of <i>t_triggerName</i> .
<i>s_triggerFunc</i>	The function to remove from the trigger's callback list.

Value Returned

<i>t</i>	The function was successfully unregistered and will no longer be triggered when the event occurs.
<i>nil</i>	<i>s_triggerFunc</i> has not been registered as a trigger for <i>t_triggerName</i> .

Example

```
ddUnregTrigger( "preCheckIn" 'checkinTrigFunc ) => t
```

Related Topics

[ddRegTrigger](#)

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 `ddl`, call `ddIsObjLibMarked` to see if 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

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<code>t</code>	The library list update was successful.
<code>nil</code>	The library list update failed due to either syntax errors in the <code>cds.lib</code> file or one of the <code>PreUpdateLibList</code> triggers returning <code>nil</code> .

Examples

```
ddUpdateLibList( ) => t
```

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).

Design Data Services 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 `~/ .cdserv` file.

Tool	Variable	Type	Default	Choices
<code>ddserv.browser</code>	<code>onOpen</code>	boolean	<code>nil</code>	
<code>ddserv.browser</code>	<code>onPlace</code>	boolean	<code>nil</code>	
<code>ddserv.browser</code>	<code>onOther</code>	boolean	<code>nil</code>	
<code>ddserv.ciw</code>	<code>promptOnExit</code>	boolean	<code>t</code>	
<code>ddserv.lib</code>	<code>dmchoice</code>	string	<code>"No DM"</code>	
<code>ddserv.lib</code>	<code>newLibDir</code>	string	<code>" "</code>	
<code>ddserv.lib</code>	<code>techChoice</code>	cyclic	<code>"compile"</code>	<code>"compile"</code> <code>"bind"</code> <code>"none"</code> <code>"reference"</code>

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

<code>ddserv.he</code>	<code>topCvMode</code>	<code>cyclic</code>	<code>"same"</code>	<code>"same"</code>	Opens the top cellview in the same mode specified for opening the configuration cellview.
				<code>"read"</code>	Opens the top cellview in "read" mode.
				<code>"append"</code>	Opens the top cellview in "append" mode.

Library Functions

With the new library structure, there is no concept of a current library, 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.

Library Synchronization Functions

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.

Customer Library Browser Integration

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:

- **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

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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 be 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).

CIW Menu Customization

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ciwAddTranslatorItem

```
ciwAddTranslatorItem(  
    s_tag  
    t_itemText  
    t_callback  
)  
=> t / nil
```

Description

Adds an entry for the `Import` or `Export` commands.

Arguments

<i>s_tag</i>	A SKILL symbol. Valid Values: <code>'Import</code> , <code>'Export</code>
<i>t_itemText</i>	The Item text to be used as a display prompt.
<i>t_callback</i>	Called to execute the given translator.

Value Returned

<i>t</i>	The entry was added.
<i>nil</i>	The entry was not added.

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

<i>ls_menuDef</i>	The definition of a menu, either a list defining the contents of the pulldown menu or the SKILL symbol that represents the list.
<i>x_windowNum</i>	Window number.

Value Returned

<i>r_menu</i>	A menu suitable for use as a pop-up menu.
<i>nil</i>	The list defining the contents for the pop-up or SKILL symbol could not be found.

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

<i>ls_menuDef</i>	The definition of a menu.
<i>x_windowNum</i>	Window number.

Value Returned

<i>r_menu</i>	A menu suitable for use as a pulldown menu.
<i>nil</i>	A pulldown menu could not be created.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>t</code>	The session is terminated.
<code>nil</code>	The session is not terminated.

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

<code>?verbose <i>g_verbose</i></code>	Specifies whether to echo extra messages during processing
<code>?reset <i>g_reset</i></code>	Specifies whether to reset the CIW menus.

Value Returned

<code>t</code>	The function succeeds.
<code>nil</code>	The menu files in the hierarchy cannot be found.

Examples

When `ciwMenuInit` is called with the reset argument, the CIW menu is re-initialized and the latest customized menu is set.

```
ciwMenuInit( ?verbose t )  
ciwMenuInit( ?reset t )
```

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

<i>r_window</i>	The window to install the menu banner into.
<i>ls_bannerDef</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_appName</i>	The application name used for finding menu files.
<i>g_verbose</i>	Whether to echo extra messages during processing.

Value Returned

<i>t</i>	The function succeeded.
<i>nil</i>	No menu files are found.

ciwRemoveTranslatorItem

```
ciwRemoveTranslatorItem(  
    s_tag  
    t_itemText  
    t_callback  
)  
=> t / nil
```

Description

Removes an entry from the `Import` or `Export` commands.

Arguments

<i>s_tag</i>	A SKILL symbol. Valid Values: <code>'Import</code> , <code>'Export</code>
<i>t_itemText</i>	The Item text to be used as a display prompt.

Value Returned

<code>t</code>	The entry was removed.
<code>nil</code>	The entry was not removed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<code>t</code>	Any other SKILL symbol passed as an action (meaning that the Library Browser will be called).
<code>nil</code>	No library is selected or if the Library Browser is not synchronized with any particular form.

Examples

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_sessionName</i>	The session name.

Value Returned

<i>t_sessionName</i>	The session name.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddsCloseLibManager

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

Description

Closes the Library Manager.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_libName</i>	The name of the library. If <i>t_cellName</i> and <i>t_viewName</i> are blank, invokes the property editor on the library object.
<i>t_cellName</i>	The name of the cell or "" in the specified library. If <i>t_viewName</i> is blank, invokes the property editor on the cell object.
<i>t_viewName</i>	The name of the view or "" in the specified cell. Invokes the property editor on the cellview object.

Value Returned

<i>Property Editor</i>	Invokes the property editor on the named object.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>t</code>	The function succeeds.
<code>nil</code>	No library is selected or the Library Browser is not synchronized with any particular form.

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

<i>g_form</i>	Accepts a form or a page from a tab field.
<i>s_targetName</i>	The full path name specified as a symbol.
<i>t_filter</i>	Optional string argument which filters file names by using an asterisk at the beginning. Default is no filter.
<i>s_mode</i>	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 The 'existingFiles mode is not supported.
<i>t_caption</i>	Optional argument that allows changing the title of the dialog box that is created. Default is "Unix Browser".

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

t	The dialog executes successfully.
nil	The dialog encounters an error during execution.

Examples

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
)
```

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.
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
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))  
)
```

Related Topics

[ddsSyncWithForm](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>b_ddLibId</code>	The <code>ddId</code> of the library that is currently selected in the Library Browser if a form is registered to it.
<code>nil</code>	No library is selected or the Library Browser is not synchronized with any particular form.

Example

```
> x = ddsGetCurrentLib()  
dd:12347080
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_name</i>	The name as specified to the <code>ddsRegCustomLibSelect</code> function.
<code>nil</code>	Nothing is registered.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

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 from 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.

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.

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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
- ☐ *kpbs*
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.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>t</code>	The library manager opens.
<code>nil</code>	The library manager is not opened.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddsProgramName

```
ddsProgramName (  
    )  
    => t_programName
```

Description

Returns the name of the program minus any .exe extension.

Arguments

None

Value Returned

<i>t_programName</i>	The program name.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

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.

Examples

In this example, all open cellviews and CDFs that are not current with their corresponding files on disk are refreshed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
ddsRefresh(?cellview t ?cdf t)
```

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

<i>t_function</i>	Name of the SKILL callback function. The callback function has the following two arguments: ■ <i>g_type</i> specifies the type of data being refreshed. Supported value: 'cv ■ <i>l_dataList</i> lists IDs of the data that will be refreshed.

Value Returned

t	The specified function is registered successfully.
nil	The operation failed.

Examples

; Registers the function myRefreshCB, which is called before the Refresh command starts.

```
procedure( myRefreshCB_pre( 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)  
        )  
    )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
)  
)
```

It has misprint and should be fixed to:

```
procedure( myRefreshCB_pre( 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)  
    )  
  )  
)  
ddsRegPreRefreshTrigger( "myRefreshCB_pre" )
```

Related Topics

[ddsRefresh](#)

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

<i>t_function</i>	Name of the SKILL callback function. The callback function has the following three arguments: ■ <i>g_type</i> specifies the type of data being refreshed. Supported value: 'cv ■ <i>l_good</i> lists IDs of the data that successfully refreshed successfully. ■ <i>l_bad</i> 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"
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
        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" )
```

Related Topics

[ddsRefresh](#)

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

<i>t_name</i>	A string to name the registration by.
<i>s_sync</i>	The SKILL symbol of the function that performs the <code>ddsSyncWithForm</code> action.
<i>s_update</i>	The SKILL symbol of the function that performs the <code>ddsUpdateSyncWithForm</code> action.
<i>s_endSync</i>	The SKILL symbol of the function that performs the <code>ddsEndSyncWithForm</code> action.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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  
'dsbGetCurrentLib )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Examples

Displays the Save Modified Data form.

```
ddsHiSaveData (  
=>t
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_lib</i>	The name of a library.
<i>t_cell</i>	The name of a cell.
<i>t_view</i>	The name of a view.

Value Returned

<i>t</i>	Always returns <i>t</i> .

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

<i>t_lib</i>	The name of a library.
<i>t_cell</i>	The name of a cell. If not specified, only the specified library is selected.
<i>t_view</i>	The name of a view. If not specified, only the specified cell or library is selected.

Value Returned

<i>t</i>	The Library Manager is open and the specified objects exist in the database. If the top library is already selected, the child objects are shown in the window showing the top library.
<i>nil</i>	The Library Manager is not yet open or the specified objects do not exist in the database.

Examples

```
ddsSetLibManLCV( "library" "data" "layout" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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 ("" "" "").

Examples

```
ddsGetLibManLCV( )  
=> ("ether" "adc_sample_hold" "layout")
```

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

<code>g_form</code>	Accepts a form or a page from a tab field.
<code>s_action</code>	Specifies an action to further control starting the Library Browser ❑ <code>'design</code> or <code>'open</code> <code>ddserv.browser onOpen</code>: Opens automatically, when the Open Design form is displayed in the DFII software. ❑ <code>'place</code> <code>ddserv.browser onPlace</code>: Opens automatically, when a Place Instance form appears. (There are several types of Place Instance forms.) ❑ <code>'other</code> <code>ddserv.browser onOther</code>: Opens automatically, when any other type of linked form is displayed in the DFII software.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

<i>s_libField</i>	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 <code>hiCreateForm</code> .
<i>s_cellField</i>	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 <code>hiCreateForm</code> .
<i>s_viewField</i>	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 <code>hiCreateForm</code> .
<i>l_viewTypes</i>	A list of <code>viewType</code> strings if <i>s_viewField</i> 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

<code>t</code>	The execution of the function succeeds.
<code>nil</code>	No library is selected or if the Library Browser is not synchronized with any particular form.

Examples

This example creates the synchronization between form and Library Browser, where form is created by `exampleOpenForm()` and the Library Browser is created by `exampleBrowser()`.

```
exampleLibName = hiCreateStringField(  
  
    ?name 'exampleLibName  
  
    ?prompt "Library Name"  
  
    ?value ""  
  
    ?callback "ddsUpdateSyncWithForm( )"  
  
)  
  
exampleCellName = hiCreateStringField(  

```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
?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"
```

```
"exampleOpenForm( )"
```

```
list( exampleLibName
```

```
    exampleCellName
```

```
    exampleViewName
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
exampleBrowser )

""

)

procedure( exampleBrowser( )

    ddsSyncWithForm( exampleOpenForm 'browse 'exampleLibName

        'exampleCellName 'exampleViewName )

    )

procedure( exampleOpenForm( )

    ddsSyncWithForm( exampleOpenForm 'other 'exampleLibName

        'exampleCellName 'exampleViewName )

    hiDisplayForm( 'exampleOpenForm )

    )

    ddsEndSyncWithForm()
```

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)"

)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

; Create a browse button for selecting a file. This browse button is placed in a HI Tab field.

```
field1 = hiCreateStringField( ?name 'libname ?prompt "Library Name")

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)

        list(field2 0:40 200:0 100)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
list(fileBrowseBtn 210:40 80:30))

)

)

form=hiCreateAppForm(

    ?name      'browseForm

    ?formTitle "Example Form"

    ?callback ""

    ?dontBlock t

    ?fields    list(list(tabs 0:0 300:100))

)

hiInstantiateForm(form)

hiDisplayForm(form)
```

ddsUnregCustomLibSelect

```
ddsUnregCustomLibSelect(  
    [ g_verbose ]  
)  
=> t
```

Description

Un-registers the custom Library Browser interface during a session.

Argument

<i>g_verbose</i>	A message with the names of the functions it is registering.

Value Returned

t	Always returns t.

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

<code>t</code>	The execution of the function succeeds.
<code>nil</code>	No library is selected or the Library Browser is not synchronized with any particular form.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_libName</i>	The name of the library.
<i>t_cellName</i>	The name of the cell in the specified library.
<i>t_viewName</i>	The name of the view in the specified cell.

Value Returned

<i>t_viewFile</i>	The viewFile window to which the information is printed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

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 synchronization 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 synchronize 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
;; viewType 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 synchronize the library field so it is the
;; only required field from ddsSyncWithForm. Some forms only synchronize 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 mistyped
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
;; 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 synchronization 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 synchronized to it.
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
;;*****
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 registered
        ;; 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.
    )
)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
;;
(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)
    )
  )
)

;; *****
;; 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)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```

        )
        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"
            ) 5:0 200:30 60))
        )
    )
)

```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```

        ?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
    )
)

```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
        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)
                myOpenDBBox("Need Library Name" "enter a library name.")
                return()
            )
            (cellN == ""
                hiSetCallbackStatus(form nil)
                hiSetCurrentField(form 'cellName)
                hiHighlightField(form 'cellName hicError)
                myOpenDBBox("Need Cell Name" "enter a cell name.")
                return()
            )
            (viewN == ""
                hiSetCallbackStatus(form nil)
                hiSetCurrentField(form 'viewName)
                hiHighlightField(form 'viewName hicError)
                myOpenDBBox("Need View Name" "enter a view name.")
                return()
            )
        )
        (!ddGetObj(libN cellN viewN)
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
        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
    )
)
```

dmb Compatibility Functions

This topic provides a list of Cadence® SKILL function categories associated with tbackward compatibility. If you are writing new code, use the dds functions.

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

- dmbGetCurrentLib
- dmbLibBrowserFormDone
- dmbOpenLibBrowser
- dmbSetBrowserInteractions
- dmbSyncWithForm
- dmbSyncWithCellForm
- dmbUpdateBrowser

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>ddId</code>	The <code>ddId</code> of the library that is currently selected in the Library Browser.
<code>nil</code>	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.

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

<i>s_formtype</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

t Execution of the function succeeded.

Example

```
dmbLibBrowserFormDone( 'design ' )
```

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

<code>t</code>	The library manager is opened.
<code>nil</code>	The library manager is not opened.

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):

<code>ddserv.browser</code>	<code>onOpen</code>
<code>ddserv.browser</code>	<code>onPlace</code>
<code>ddserv.browser</code>	<code>onOther</code>

Or, you can explicitly call `envSetVal` in your `.cdsinit` file:

```
envSetVal( "ddserv.browser" "onOpen" 'boolean nil)  
envSetVal( "ddserv.browser" "onPlace" 'boolean t)
```

Arguments

<code>?openDesignOption</code>	Defines interactions between the Open Design form and the Library Browser. When set to <code>t</code>, the Library Browser is opened automatically when this form is opened.
<code>?openPlaceInstanceOption</code>	Defines interactions between any Place Instance form and the Library Browser. When set to <code>t</code>, the Library Browser is opened automatically when this form is opened.
<code>?openOthersOption</code>	Defines interactions between any other type of linked form and the Library Browser. When set to <code>t</code>, the Library Browser is opened automatically when this form is opened.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

All the arguments for closing are no longer applicable.

Value Returned

<code>t</code>	Execution of the function succeeded.
<code>nil</code>	Execution of the function failed if any of the input parameters is invalid.

Example

```
dmbSetBrowserInteractions( ?openOthersOption t)
```


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

<code>s_form</code>	Name of the form that is calling the Library Browser.
<code>s_synctype</code>	Specifies an action to further control the use of the Library Browser. Defines how the Library Browser is called: ❑ <code>'design</code> Opens automatically when the Open Design form is displayed in the DFII software. ❑ <code>'place</code> Opens automatically when the Place Instance form appears. (There are several types of Place Instance forms.)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ❑ '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.

<i>s_libraryName</i>	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 <code>hiCreateForm</code> .
<i>s_cellName</i>	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 <code>hiCreateForm</code> .
<i>s_viewName</i>	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 <code>hiCreateForm</code> .
<i>s_versionNumber</i>	The version number form field name specified in <code>hiCreateForm</code> .

Value Returned

<code>t</code>	The function succeeds.
<code>nil</code>	No library is selected or the Library Browser is not synchronized with any particular form.

Examples

```
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
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
        ?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 )
)
```

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

<code>s_form</code>	Name of the form that is calling the Library Browser.
<code>s_synctype</code>	Specifies an action to further control the use of the Library Browser. Defines how the Library Browser is called: ❑ <code>'design</code> Opens automatically when the Open Design form is displayed in the DFII software. ❑ <code>'place</code> Opens automatically when the Place Instance form appears. (There are several types of Place Instance forms.)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ❑ '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.

<i>s_libraryName</i>	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 <code>hiCreateForm</code> .
<i>s_cellName</i>	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 <code>hiCreateForm</code> .
<i>s_viewName</i>	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 <code>hiCreateForm</code> .
<i>s_versionNumber</i>	The version number form field name specified in <code>hiCreateForm</code> .

Value Returned

<code>t</code>	The function succeeds.
<code>nil</code>	No library is selected or the Library Browser is not synchronized with any particular form.

Examples

```
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
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
        ?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 )
)
```

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

<code>t</code>	Execution of the function succeeded.
<code>nil</code>	No library is selected or the Library Browser is not synchronized with any particular form.

Examples

```
exampleLibName = hiCreateStringField(  
    ?name 'exampleLibName  
    ?prompt "Library Name"  
    ?value ""  
    ?callback "dmbUpdateBrowser( )" )  
exampleCellName = hiCreateStringField(  
    ?name 'exampleCellName  
    ?prompt "Cell Name"  
    ?callback "dmbUpdateBrowser( )" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
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 )
)
```


Cell Category Functions (ddCat Interface)

This topic provides a list of Cadence® SKILL function categories associated with cell category. 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
```

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

- [ddCatOpen](#)
- [ddCatOpenEx](#)
- [ddCatAddItem](#)
- [ddCatSubItem](#)
- [ddCatSave](#)
- [ddCatRemove](#)
- [ddCatClose](#)
- [ddCatGetCatName](#)
- [ddCatFindCat](#)
- [ddCatGetCellsNotInCat](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ddCatIsObjInCat

ddCatOpen

```
ddCatOpen (
    b_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

<i>b_ddId</i>	The library ID or category ID (for hierarchical categories). If the object is a <i>libId</i> and the given name is "" or <code>nil</code> , the library's list of categories can be accessed or modified.
<i>t_categoryName</i>	The name of the category.
<i>t_mode</i>	The modes "a", "w", or "r".

Values Returned

<i>b_objId</i>	The ID for the opened category.
<code>nil</code>	The category does not exist or cannot be opened in the specified mode.

Examples

Updates (or creates) a category and adds a cell name to it.

```
cat = ddCatOpen(libId "myCat" "a")
ddCatAddItem(cat "cell1" "cell")
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

`ddCatSave (cat)`
`ddCatClose (cat)`

Related Topics

`ddGetObj`

ddCatOpenEx

```
ddCatOpenEx (
    b_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

<i>b_ddId</i>	The library ID or category ID (for hierarchical categories). If the object is a <i>libId</i> and the given name is " " or <i>nil</i> , the library's list of categories can be accessed or modified.
<i>t_categoryName</i>	The name of the category.
<i>t_mode</i>	The modes "a", "w", or "r". For more information, see ddGetObj .
<i>x_keepEmpty</i>	When <i>x_keepEmpty</i> is 1, the category files are not deleted, but, when <i>x_keepEmpty</i> is 0, the category files are deleted.

Value Returned

<i>b_objId</i>	The ID for the opened category.
<i>nil</i>	The category does not exist or cannot be opened in the specified mode.

By calling `ddCatOpenEx( g_ddId t_categoryName t_mode nil)`, you get the same behavior as in [ddCatOpen](#).

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Examples

```
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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddCatAddItem

```
ddCatAddItem(  
    b_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

<i>b_ddId</i>	ID of the category to be modified.
<i>t_name</i>	The name of the item.
<i>t_type</i>	The object type of the item. The caller must specify the type of the object being added.

Value Returned

<i>t</i>	The function was successful
<i>nil</i>	The function failed.

ddCatSubItem

```
ddCatSubItem(  
    b_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

<i>b_ddId</i>	ID of a category.
<i>t_name</i>	The name of a cell or category.

Value Returned

t	The function was successful
nil	The function failed

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddCatSave

```
ddCatSave(  
    b_ddId  
)  
=> t / nil
```

Description

Writes the category information to disk. This writes out any changes made to the category to disk.

Arguments

<i>b_ddId</i>	ID of a category.

Value Returned

t	The function was successful.
nil	The function failed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddCatRemove

```
ddCatRemove(  
    b_ddId  
)  
=> t / nil
```

Description

Removes the category and its contents from disk. All references to the category are removed from the library.

Arguments

<i>b_ddId</i>	ID of a category.

Value Returned

t	The function was successful.
nil	The function failed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddCatClose

```
ddCatClose(  
    b_ddId  
)  
=> t / nil
```

Description

Closes the category and purges its information from virtual memory. Any edits since the last `ddCatSave` are lost.

Arguments

<i>b_ddId</i>	ID of a category.

Value Returned

<i>t</i>	The function was successful.
<i>nil</i>	The function failed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

ddCatGetCatName

```
ddCatGetCatName (
    b_ddId
)
=> t_categoryName / nil
```

Description

Returns the category name of the given category ID.

Arguments

<i>b_ddId</i>	ID of a category.

Value Returned

<i>t_categoryName</i>	The category name of the given category ID.
<i>nil</i>	The function failed

ddCatGetCatMembers

```
ddCatGetCatMembers (  
    b_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

<i>b_ddId</i>	ID of a category.

Value Returned

<i>l_memberList</i>	The list of member name and type pairs in the given category.
<i>nil</i>	The category has no members.

Example

```
'(("nand2" "cell") ("Models" "category"))
```

Returns the list of member names and type pairs.

ddCatGetLibCats

```
ddCatGetLibCats(  
    b_ddId  
)  
=> l_categoryNames / nil
```

Description

Returns the list of library-level categories.

Arguments

<i>b_ddId</i>	ID of a category.

Value Returned

<i>l_categoryNames</i>	The list of library-level categories.
<i>nil</i>	The library has no categories.

ddCatFindCat

```
ddCatFindCat (
    b_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

<i>b_ddId</i>	ID of a category.
<i>t_name</i>	The category name.
<i>t_mode</i>	The modes "a", "w", or "r". For more information, see ddGetObj .

Value Returned

<i>b_objId</i>	The ID of a category with the given name in the given mode. The returned ID can be used to access the category members.
<i>nil</i>	The function failed.

ddCatGetCellsNotInCat

```
ddCatGetCellsNotInCat(  
    b_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

<i>b_ddId</i>	ID of a category.

Value Returned

<i>l_cellNames</i>	List of cells that are not in any category in the specified library.
<i>nil</i>	All cells are listed in some category.

ddCatIsObjInCat

```
ddCatIsObjInCat (
    t_name
    b_ddId
)
=> t_type / ddcCatUnknownType
```

Description

Searches for the named object in the given category.

Arguments

<i>t_name</i>	The object name.
<i>b_ddId</i>	The category ID.

Value Returned

<i>t_type</i>	The type of the object found if the category contains something of that name.
ddcCatUnknownType	The category does not contain anything of that name.

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
- dmCreateCellCategory
- dmRemoveCellCategory
- dmRemoveCellFromCategory

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 checkpoint 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

<code>g_categoryId</code>	ID of the category to which the specified cells are added.
<code>l_cellList</code>	List of cells to be added to the category.

Value Returned

<code>t</code>	The cells are added.
<code>nil</code>	The specified cells or the category ID cannot be found.

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

<i>g_libOrCatId</i>	ID of the library or cell category to be created.
<i>t_categoryName</i>	Name of the category to be created.
<i>l_cellList</i>	List of cells to be added to the category.

Value Returned

<i>t</i>	The category is created.
<i>nil</i>	The category cannot be created.

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

<i>g_categoryId</i>	ID of the cell category to be removed.

Value Returned

<i>t</i>	The cell category is removed.
<i>nil</i>	The library or parent cell category cannot be found.

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

<i>g_categoryId</i>	ID of the cell category to be removed.
<i>l_cellList</i>	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.

Design Editor Functions

This topic provides a list of Cadence® SKILL function categories associated with the Design Editor.

Registering and Unregistering 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.

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

- [deAddAppCapabilities](#)
- [deRegApp](#)
- [deRemoveAppCapabilities](#)
- [deUnRegApp](#)
- [deRegUserTriggers](#)
- [deInheritEnabling](#)
- [deInheritMenus](#)
- [deUnInheritMenus](#)
- [deRegProbeTrigger](#)
- [deUnRegProbeTrigger](#)
- [deoRequireToolbars](#)
- [deoOnEvent](#)

deAddAppCapabilities

```
deAddAppCapabilities(  
    t_viewType  
    [ ?newFile g_newFile ]  
    [ ?openFile g_openFile ]  
)  
=> t / nil
```

Description

Adds the required application capabilities to the specified view type.

Arguments

<code>t_viewType</code>	View type to which the specified capabilities are to be added.
<code>?newFile g_newFile</code>	Adds the <i>New File</i> capability if set to non-nil.
<code>?openFile g_openFile</code>	Adds the <i>Open File</i> capability if set to non-nil.

Value Returned

<code>t</code>	Application capabilities were added to the specified view.
<code>nil</code>	The view type is not found.

Example

Adds `openFile` capability to the registered view type `document`.

```
(deAddAppCapabilities "document" ?openFile t)
```

Related Topics

[deRemoveAppCapabilities](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

deRegApp

```
deRegApp (
    + ?viewType t_viewType +
    + ?parentType t_parentType
    + ?appName t_appName +
    + ?help t_help
    + ?dataTrigger s_dataTrigger +
    + ?appTrigger s_appTrigger +
    [ ?parentType t_parentType ]
    [ ?help t_help ]
    [ ?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`

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

View type or view subtype that *t_viewType* subclasses.

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*

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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 it 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`

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>t</code>	The registration was successful.
<code>nil</code>	the registration was not successful.

deRemoveAppCapabilities

```
deRemoveAppCapabilities(  
    t_viewType  
    [ ?newFile g_newFile ]  
    [ ?openFile g_openFile ]  
)  
=> t / nil
```

Description

Removes the required application capabilities from the specified view type.

Arguments

<code>t_viewType</code>	Name of the view type to which the <code>g_newFile</code> and <code>g_openFile</code> capabilities are to be removed.
<code>?newFile g_newFile</code>	Removes the <i>New File</i> capability if set to non-nil.
<code>?openFile g_openFile</code>	Removes the <i>Open File</i> capability if set to non-nil.

Value Returned

<code>t</code>	Application capabilities were removed from the specified view.
<code>nil</code>	The view type is not found.

Example

Removes `newFile` capability above `cmd` from the registered view type `document`.

```
(deRemoveAppCapabilities "document" ?newFile t)
```

Related Topics

[deAddAppCapabilities](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_viewType</i>	View type to be removed from the list of applications known to the Design Editor.

Value Returned

<i>t</i>	The view type was unregistered.
<i>nil</i>	The view type was not unregistered.

Example

Removes "graphic" from the list of registered view types and returns *t*.

```
deUnRegApp( "graphic" )
```

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

<i>t_viewType</i>	View type or view subtype to which <i>s_appTrigger</i> , <i>s_menuTrigger</i> , and <i>s_postInstallTrigger</i> will be attached. An application can attach user triggers to any number of view types, but must call <code>deRegUserTriggers</code> once for each one.
<i>s_appTrigger</i>	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 .

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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)
```

The following example registers trigger for Virtuoso Layout Suite EXL. Similar triggers can be registered for higher tiers.

```
deRegUserTriggers("maskLayoutEXL" nil nil 'trigger)
```

deUnRegUserTriggers

```
deUnRegUserTriggers(  
    t_viewType  
)  
=> t / nil
```

Description

Removes any user triggers associated with *t_viewType*.

Arguments

<i>t_viewType</i>	View type or view subtype for which the user triggers are to be removed.

Value Returned

<i>t</i>	The triggers were unregistered.
<i>nil</i>	The triggers were not unregistered.

Examples

Removes any user triggers associated with view type "graphic" and returns *t*.

```
deUnRegUserTriggers( "graphic" )
```

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

<i>t_viewType</i>	View subtype for which enabling is to be inherited.

Value Returned

<i>t</i>	The enabling was inherited.
<i>nil</i>	The enabling was not inherited.

Examples

Sets view type "graphic" to inherit the enabling of its parent and returns *t*.

```
deInheritEnabling( "graphic" )
```

deUnInheritEnabling

```
deUnInheritEnabling(  
    t_viewType  
)  
=> t / nil
```

Description

Removes the inheritance of enabling for *t_viewType*.

Arguments

<i>t_viewType</i>	View subtype for which the inheritance of enabling is to be removed.

Value Returned

<i>t</i>	The enabling inheritance was removed.
<i>nil</i>	The enabling inheritance was not removed.

Example

```
deUnInheritEnabling( "graphic" )
```

Removes dependence of view type "graphic" enabling on its parent and returns *t*.

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

<i>t_viewType</i>	View subtype for which the inheritance of menus is desired.

Value Returned

<i>t</i>	The menus could be inherited.
<i>nil</i>	The menus could not be inherited.

Examples

Sets view type "graphic" to inherit the menus of its parent and returns *t*.

```
deInheritMenus( "graphic" )
```

deUnInheritMenus

```
deUnInheritMenus(  
    t_viewType  
)  
=> t / nil
```

Description

Removes the inheritance of menus for *t_viewType*.

Arguments

<i>t_viewType</i>	View subtype for which the inheritance of menus is to be removed.

Value Returned

<i>t</i>	The menu inheritance was removed.
<i>nil</i>	the menu inheritance was not removed.

Examples

Removes dependence of view type "graphic" menus on its parent and returns *t*.

```
deUnInheritMenus( "graphic" )
```

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

<i>s_probeTrigger</i>	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.

Examples

Installs `'myProbeTrigger` as the current global probing trigger and returns t.

```
deRegProbeTrigger( 'myProbeTrigger )
```

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

Removes the current probe trigger and returns t.

```
deUnRegProbeTrigger()
```


deGetProbeTrigger

```
deGetProbeTrigger(  
    )  
=> s_probeTrigger / nil
```

Description

Returns the current probe trigger.

Arguments

None

Value Returned

<i>s_probeTrigger</i>	SKILL symbol for the current global probe trigger.
<i>nil</i>	No current probe trigger is registered.

Example

Returns 'myProbeTrigger.

```
deGetProbeTrigger()
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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:

<i>l_appInfo->viewType</i>	viewType	(string)
<i>l_appInfo->parentType</i>	Parent type	(string)
<i>l_appInfo->appName</i>	Application name	(string)
<i>l_appInfo->help</i>	Application help string	(string)
<i>l_appInfo->dataTrigger</i>	SKILL data trigger	(symbol)
<i>l_appInfo->appTrigger</i>	SKILL application trigger	(symbol)
<i>l_appInfo->userAppTrigList</i>	SKILL user app trigger list	(symbol)
<i>l_appInfo->userMenuTrigList</i>	SKILL user menu trigger list	(symbol)
<i>l_appInfo->userPostInstallTrigList</i>	SKILL user post-install trigger list	(symbol)
<i>l_appInfo->inheritEnabling</i>	Whether or not to inherit enabling	(t/nil)
<i>l_appInfo->inheritMenus</i>	Whether or not to inherit menus	(t/nil)
<i>l_appInfo->widgetType</i>	<i>widgetType</i> of the window	(string)
<i>l_appInfo->noCadenceMenu</i>	Whether or not to install the Cadence Menu	(t/nil)
<i>l_appInfo->subTypes</i>	<i>subviewType</i> list of this <i>viewType</i>	(list)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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*.

- 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.

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
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
userAppTrigger nil userMenuTrigger nil inheritMenus nil
inheritEnabling nil subTypes nil
)
```

Note: `deGetAppInfo()` returns a pointer to the application information. If the application information is then changed, your local pointer to the application information also changes.

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")
```

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
```

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.

Arguments

None

Value Returned

None

Example

```
deHiSetDefaultApplication()
```

Related Topics

[Set Default Application Form](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	The given window.
<i>t_workspaceName</i>	The workspace name to be set for the given window (must be valid).

Value Returned

<i>t</i>	Workspace name set to window successfully.
<i>nil</i>	Workspace name not set and warning printed.

Examples

Sets `window(3)` to the workspace name "Classic".

```
deSetWorkspace(window(3) "Classic")
```

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.

Arguments

<i>t_appName</i>	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.
<i>s_className</i>	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 <code>deInstallPlugin</code> is applied to <i>t_appName</i> or <i>s_className</i> . For information about autoloading your classes, see " Autoloading Your Classes ".
<i>s_superClass</i>	Specifies the name of the superclass from which the <i>s_className</i> is derived

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

?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

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<i>l_appClassInfo</i>	Returns a copy of the registration information associated with the plugin, if the registration is successful
<i>nil</i>	Returns <i>nil</i> 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"))
```

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

<i>w_windowID</i>	Specifies the ID of an application window on which the plugin is to be installed
<i>S_plugin</i>	Specifies the name (string) or class name (symbol) of the plugin to be installed

Value Returned

<i>stdobj</i>	Returns an instance of the plugin installed in the given application window
<i>nil</i>	Returns <code>nil</code> if the named plugin (string) has not been registered with <code>deRegPlugin</code> , the class name (symbol) is not a sub-class of <code>DEPlugin</code> , or the plugin is not pluggable to the given application window (pluggable applications are specified when registering the plugin using <code>deRegPlugin</code>)

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.

Arguments

<i>w_window</i>	Specifies the ID of the window from which the plugin is to be removed.
<i>S_plugin</i>	Specifies the name (string) or class name (symbol) of the plugin to be removed.

Value Returned

<i>t</i>	Returns <i>t</i> if the plugin is successfully removed.
<i>nil</i>	Returns <i>nil</i> if the named plugin is not found installed in the specified window or its <code>exitTrigger</code> returns <i>nil</i> .

Related Topics

[deoExitTrigger](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>S_plugin</i>	Specifies the name (string) or class name (symbol) of the plugin you want to search for.
<i>g_window</i>	Specifies the ID of the application window—session or tab—in which you want to search for the named plugin.

Value Returned

<i>stdobj</i>	Returns an instance of the named plugin you are searching for
<i>nil</i>	Returns <i>nil</i> if the named plugin is not found installed in the specified window

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.

Nested calls of `deBeginConfigurePlugins` and `deEndConfigurePlugins` are not supported.

Arguments

<code>w_window</code>	An application window.

Value Returned

<code>nil</code>	Always returns <code>nil</code> . The value is not significant.

Examples

Informs the Design Editor that a series of calls for `deInstallPlugin` or `deRemovePlugin`, or both will be made in `window:2`.

```
(deBeginConfigurePlugins (window 2))
```

Related Topics

[deInstallPlugin](#)

[deRemovePlugin](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

hiApplyWinConfig

deEndConfigurePlugins

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.

Nested calls of `deBeginConfigurePlugins` and `deEndConfigurePlugins` are not supported.

Arguments

<code>w_window</code>	Application window in which <code>deBeginConfigurePlugins</code> was called.

Value Returned

<code>nil</code>	Always returns <code>nil</code> . The value is not significant.

Examples

Indicates the end of the series of calls for `deInstallPlugin` and `deRemovePlugin` marked by the `deBeginConfigurePlugins` in `window:2`.

```
(deEndConfigurePlugins (window 2))
```

Related Topics

[deInstallPlugin](#)

[deRemovePlugin](#)

[deBeginConfigurePlugins](#)

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

<i>t_assistant</i>	Name of the assistant to be searched.
<i>w_sessionWindow</i>	The Session window in which the assistant needs to be searched.
?search_object_name	Searches for an assistant with object name matching <i>t_assistant</i> . Default value: nil
?search_window_name	Searches for an assistant with window name matching <i>t_assistant</i> . 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 <i>w_sessionWindow</i> .

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<code>w_dwindow</code>	The dock window ID of the assistant with the specified window or object name in the specified session window
<code>nil</code>	The named assistant was not found installed in the specified session window

Examples

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)))
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

deGetPlugins

```
deGetPlugins (
    w_window
    [ g_activeOnly ]
)
=> l_stdobj / nil
```

Description

Returns a list of all plugin instances installed in the given application window

Arguments

<i>w_window</i>	Specifies the ID of the application window (tab window) from which you want to get a list of instances of all plugins
<i>g_activeOnly</i>	(Optional) If non-nil, the function returns only the instances of active plugins in a given window. The default value is <code>nil</code> .

Value Returned

<i>l_stdobj</i>	Returns a list of all instances of the plugins installed in the given application window
<i>nil</i>	Returns <code>nil</code> if no plugin is found installed in the given application window

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

deGetAppClassInfo

```
deGetAppClassInfo(  
    g_application  
)  
=> dpl_appClassInfo
```

Description

Returns a copy of the registration information (DPL) associated with the given application

Arguments

<i>g_application</i>	Specifies the application name (string) of the plugin, the class name (symbol) of the plugin, or an instance (stdobj) of the plugin.

Value Returned

<i>dpl_appClassInfo</i>	Returns a copy of the registration information (DPL) associated with an application.
<i>nil</i>	Returns <i>nil</i> if the plugin is not registered by using <i>deRegPlugin</i> .

Examples

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.

```
(deGetAppClassInfo "My Plugin")  
=>(nil appType plugin appName "My Plugin" className MyPlugin superClass DEPlugin  
bindkeys nil help "" pluggableMainApps ("Schematics" "Schematics XL")  
pluggableViewTypes nil shortDescription "")
```

Methods that are not prefixed with *deo* should not be re-implemented by code groups outside DE.

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

<i>g_application</i>	Specifies an instance (stdobj) of the plugin
<i>l_args</i>	Single argument, which is a disembodied property list. <i>l_args</i> ->action property equals 'activate in addition to the properties specified for DE main application triggers. The trigger argument (<i>l_args</i>) for DE plugin is similar to the <u>Trigger Argument</u> given to DE main application triggers. Two new values are added for the <i>args</i> ->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 if the specified application passes security check.
nil	Returns nil if the application should not be installed.

Examples

```
defmethod( deoEnableTrigger ((this MyPlugin) @optional args)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
(and (callNextMethod this args) (check_my_license this))
```

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

<i>g_application</i>	Specifies an instance (stdobj) of the plugin
<i>l_args</i>	Displays a disembodied property list. <i>l_args->action</i> property equals 'deactivate' in addition to the properties for DE main application triggers for removing the plugin. 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 <i>args->action</i> property, they are 'activate, and 'deactivate. These are used for installing and removing a plugin in a window, respectively.

Value Returned

<i>t</i>	Returns <i>t</i> to permit the removal of the plugin
<i>nil</i>	Returns <i>nil</i> 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 )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>g_application</i>	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.

<i>nil</i>	Returns <i>nil</i> if no assistants are previously registered with the plugin.

Example

```
defmethod( deoGetAssistantList ((this MyPlugin))  
    '("Search" right) ( "Tree Assistant" left) ( "OI Assistant" left)))
```

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

<i>g_application</i>	Specifies an instance (stdobj) of the plugin.

Value Returned

<i>l_bindkeys</i>	Returns a collective list of bindkeys defined by the plugin and its super class.
<i>nil</i>	Returns <i>nil</i> if no bindkeys are registered with the plugin.

Example

```
(defmethod deoGetBindkeys ((this MyPlugin))  
    (append (callNextMethod this) my_bindkeys))
```


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

<i>g_application</i>	Specifies an instance (stdobj) of the plugin
<i>l_args</i>	Single argument, which is a disembodied property list. The trigger argument (<i>l_args</i>) 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 <code>args->action</code> property, they are 'activate, and 'deactivate. These are used for installing and removing a plugin in a window, respectively. <code>args->bannerMenus</code> equals the list of menus in the main window when the menu trigger is called.

Value Returned

<i>l_menus</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
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))))))
```

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

<i>g_application</i>	Specifies an instance (stdobj) of the plugin.
<i>w_sessionWin</i>	Specifies the ID of the session window in which the tab switch happens.
<i>w_intoWin</i>	Specifies the ID of the window being switched into
<i>g_outofWin</i>	Specifies the window being switched out. This value can be <code>nil</code> .

Value Returned

<i>g_value</i>	Return value is ignored.

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

<i>g_application</i>	Specifies an instance (stdobj) of the plugin.
<i>l_args</i>	Single argument, which is a disembodied property list. The trigger argument (<i>l_args</i>) 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 <code>args->action</code> property, they are 'activate, and 'deactivate. These are used for installing and removing a plugin in a window, respectively.

Value Returned

<i>g_value</i>	Return value is ignored.

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

<i>g_application</i>	Specifies an instance (stdobj) of the plugin
<i>l_args</i>	Single argument, which is a disembodied property list. The trigger argument (<i>l_args</i>) 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 <code>args->action</code> property, they are 'activate, and 'deactivate. These are used for installing and removing a plugin in a window, respectively.

Value Returned

<i>g_value</i>	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))))
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
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
```

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

<i>g_plugin</i>	Specifies an instance (stdobj) of the plugin

Value Returned

<i>l_toolbarNames</i>	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)))
```

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

<i>g_plugin</i>	Specifies an instance (stdobj) of the plugin.
<i>t_name</i>	Specifies the name of the toolbar.
<i>o_toolbar</i>	Specifies a handle to the toolbar.
<i>l_args</i>	Single argument, which is a disembodied property list. The trigger argument (<i>l_args</i>) 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 <code>args->action</code> property, they are 'activate, and 'deactivate. These are used for installing and removing a plugin in a window, respectively.

Value Returned

<i>g_value</i>	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)))
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
(callNextMethod this name toolbar args))
```

deoOnEvent

```
deoOnEvent (  
    g_app  
    s_event  
    @rest args  
)
```

Description

This event handler can be reimplemented in a subclass to receive various events.

Arguments

<i>g_app</i>	Name of the application for which <code>deoOnEvent</code> needs to be called.
<i>s_event</i>	Name of an event specified as a symbol.
@rest <i>args</i>	Arguments specific to the event.

Value Returned

None

Client Interface

This topic provides a list of Cadence® SKILL function categories associated with 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.

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

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.

If the open failed, nil is returned.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Arguments

l_fileSpec Disembodied property list which can be `nil` or contain one or more of the following elements:

- `libName`: Cellview library name
- `libId`: Cellview library ID
- `cellName`: Cellview cell name
- `viewName`: Cellview view name
- `viewId`: Cellview view ID
- `viewNameList`: *viewName* switch list
- `accessMode`: Desired access mode
- `instanceName`: Instance name specification
- `hierarchy`: Instance hierarchy specification
- `configL`: Hierarchy manager configuration

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

`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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

`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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

<code>w_window</code>	Uses the given window.
<code>nil</code>	Creates a window in a default location.
<code>l_bBox</code>	Creates a window with the given <i>bBox</i> .
<code>'interactive</code>	Creates a window, user specifies <i>bBox</i>. 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 <i>g_winSpec</i> is defaulted (or <code>nil</code> is passed), the view is checked for the property <code>defaultWindowBox</code>. If this property exists, that box is used for the window. If that property does not exist, the Design Editor chooses its own box.
<code>t_accessMode</code>	Access mode desired for the design, and will be <code>edit</code> or <code>read</code>. (These correspond, respectively, to the UNIX modes <code>a</code> and <code>r</code>, 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 <i>g_winSpec</i> or as part of <i>l_fileSpec</i>. This is for compatibility with older code. If both exist and are different, <i>t_accessMode</i> overrides <i>l_fileSpec</i>-><code>accessMode</code>. If neither is specified, then the access mode defaults to <i>edit</i>.
<code>t_appName</code>	Name of an application sub type which is to be the application installed when the window is opened. <i>t_appName</i> 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.
<code>g_sessionWindow</code>	If the session window is specified, tabs are used instead of new windows.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

g_saveSessionHistory

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.

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}")
```

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

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.

```
deOpen()  
myFileSpec = list( nil 'libName "test" 'cellName "ff" 'viewName "graphic" )
```

Opens a window, which is given the screen box (200:200 800:800), and installs the cellview *test ff graphic* in it.

```
deOpen( myFileSpec list( 200:200 800:800 ) )
```

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.

```
myFileSpec = list( nil 'hierarchy "{test ff graphic}":  
a/I4:r" )  
deOpen( myFileSpec 'interactive )
```

Opens the cellview in Layout Viewer.

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.

```
deOpen( myFileSpec winSpec "r" "Layout Viewer" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	The name of the window opened.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

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.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

	<i>accessMode</i> is the access mode. Valid values: "a" (edit), "r" (read), and "w" clears the old data and edit from scratch.
<i>g_winSpec</i>	Creates a window. The valid values are: ■ <code>nil</code> creates window using <code>.Xdefaults</code> ■ <code>interactive</code> creates a window letting user specify location ■ <code>l_bBox</code> creates a window using <code>l_bBox</code> as <code>screenBox</code> ■ <code>w_window</code> use <code>w_window</code>
<i>t_accessMode</i>	File access mode. Should be "w" to allow editing, start from scratch.
<i>t_appName</i>	Overrides default application if supplied. To view a list of valid values, enter <code>deGetAppInfo viewType</code> in the CIW.

Value Returned

<i>w_windowId</i>	The ID of the window that opened.
<code>nil</code>	No window opened.

Examples

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.

```
deNew(list(nil
?libName "test"
?cellName "top"
?viewName "layout"
?viewType "maskLayout"))
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_libName</i>	The name of the library in which the cellview will be created.
<i>t_cellName</i>	The cell in which the new cellview will be created.
<i>t_viewName</i>	The view name of the new cellview.
<i>t_viewType</i>	The type of the new cellview.
<i>g_winSpec</i>	Creates a window. The valid Values are: <i>nil</i> creates window using .Xdefaults <i>'interactive</i> creates window letting user specify location <i>l_bBox</i> creates window using <i>l_bBox</i> as screenBox <i>w_window</i> uses <i>w_window</i>

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<code>w_windowId</code>	The window Id if the new cellview is created.
<code>nil</code>	The new cellview was not created.

Examples

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.

```
deNewCellView("test" "top" "layout" "maskLayout" list(list(0 0) list(500 600)))
```

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.

If the open failed, nil is returned.

Arguments

<i>t_libName</i>	The name of the library in which the cellview is opened.
<i>t_cellName</i>	The cell to open.
<i>t_viewName</i>	The view name of the cellview.
<i>t_viewType</i>	The type of the cellview.
<i>g_winSpec</i>	The window in which to open the cellview.
<i>t_accessMode</i>	The mode in which to open the cellview. "a" (edit), "r" (read), and "w" (clears the old data and edit from scratch).

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Value Returned

<code>w_windowId</code>	The cellview opens.
<code>nil</code>	The cellview did not open.

Examples

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.

```
deOpenCellView("test" "top" "layout" "maskLayout" list(list(0 0) list(500 600))  
"a")
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>w_window</code>	If the <code>w_window</code> argument is specified the command will work on the specified window. The default value for <code>w_window</code> is the current window.
<code>numSteps</code>	If the value of <code>numSteps</code> is greater than or equal to 1, you will go directly back to the cellview that would be opened if you executed the <code>deBack(window) numSteps</code> times. If the value of <code>numSteps</code> equals 0, this will display the <i>Go Back</i> form. The default value for <code>numSteps</code> is 1.

Values Returned

<code>w_window</code>	The window value which you have gone back to.
<code>nil</code>	If the value of <code>numSteps</code> is more than the number of levels you can go back to, the API will return <code>nil</code> and warn that you cannot go that far back.

Examples

Goes back 1 step on window (3).

```
deBack( window(3) 1)
```

deForward

```
deForward(  
    [w_window [ numSteps] ]  
)  
=> w_window / nil
```

Description

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

<code>w_window</code>	If <code>w_window</code> is specified the command works on the specified window. The default value for <code>w_window</code> is the current window.
<code>numSteps</code>	If the <code>numSteps</code> is greater than or equal to 1, you will go directly forward to the cellview that would be opened by executing the <code>deForward(window)</code> command <code>numSteps</code> times. If the value of <code>numSteps</code> is 0, this will display the <i>Go Forward</i> form. The default value for <code>numSteps</code> is 1.

Values Returned

<code>w_window</code>	The window value which you have gone forward to.
<code>nil</code>	If the value of <code>numSteps</code> is more than the number of levels you can go forward to, the API will return <code>nil</code> and warn that you cannot go that far forward.

Examples

Goes forward 1 step on window (3).

```
deForward( window(3) 1)
```

Related Topics

[deBack](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>l_fileSpec</i>	File specification for the cellview to be pushed. See <code>deOpen</code> for a detailed explanation of the format for this argument.
<i>g_winSpec</i>	Window specification for the window in which to do the push. See <code>deOpen</code> for a detailed description of this argument.
<i>t_accessMode</i>	Access mode desired for the new design. Must be either <code>edit</code> or <code>read</code> .
<i>t_appName</i>	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

<i>w_window</i>	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

Examples

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)`.

```
dePush( nil window(3) )
```

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")
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

```
dePush( myFileSpec window(3) "read" "Timing Analysis" )
```


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.

Values Returned

t The design was successful opened.

nil The opening of the design failed.

Examples

Opens the chosen design in a new tab of the last active session window and returns *t*.

```
deSetHistoryOpenMode("newTab")
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>l_fileSpec</i>	File specification for the cellview to be swapped into the given window. See <code>deOpen</code> for a detailed explanation of the format for this argument.
<i>g_winSpec</i>	Window specification for the window in which to do the swap. See <code>deOpen</code> for a detailed description of this argument.
<i>t_accessMode</i>	Access mode desired for the new design. Valid values: <code>edit</code> , <code>read</code> , <code>overwrite</code> , <code>a</code> , <code>r</code> , or <code>w</code> .
<i>t_appName</i>	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

<i>w_window</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Examples

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)`.

```
deSwap( nil 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.

```
deSwap( myFileSpec window(3) )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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))
```

deToggleAssistantVisibility

```
deToggleAssistantVisibility(  
    t_assistantName  
    [ w_sessionWindow ]  
)  
=> t / nil
```

Description

Toggles the visibility of the given assistant in the given session window.

Arguments

<i>t_assistantName</i>	The given assistant name.
<i>w_sessionWindow</i>	The given window. If none is specified, the current session window is taken into consideration.

Value Returned

<i>t</i>	The visibility of the given assistant toggles successfully.
<i>nil</i>	The visibility of the given assistant does not toggle.

Example

```
deToggleAssistantVisibility("World View")
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>g_sessionWindow</i>	Session window in which to toggle the toolbars. If not specified, defaults to the session window that has the current window.

Value Returned

<i>l_toolbar</i>	A list of toolbars for which the visibility state has changed.
<i>nil</i>	The visibility state of the toolbars has not changed.

Example

```
deToggleToolbars(hiGetCurrentWindow())
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>S_name</i>	Specifies the design editor toolbars, the value of <code>text</code> property (a string), or the <code>name</code> property (a symbol) of the toolbar as specified in the toolbar definition file. Unless <i>S_name</i> is specified as the name(symbol) of a toolbar, the toolbar is searched for in the active tab window.
<i>w_sessionWindow</i>	Specifies the ID of the session window in which you want to search for the named toolbar

Value Returned

<i>o_instance</i>	Returns an instance of the specified toolbar in the given session window.
<i>nil</i>	Returns <i>nil</i> if the toolbar you are searching for is not found.

Examples

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 (  
    ...
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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)))
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_sessionWindow</i>	Specifies the ID of the session window in which you want to put the given toolbar.
<i>o_toolbar</i>	Specifies a handle of the toolbar that is to be put on the given session window (under DE management).
<i>g_appName</i>	Specifies the application name in which the toolbar needs to be put. If no <i>appName</i> is given, the application running in the active-tab-window is used.

Value Returned

<i>t</i>	Returns <i>t</i> if the specified toolbar is successfully put under DE management.
<i>nil</i>	Returns <i>nil</i> if the specified toolbar is not put under DE management.

Examples

```
(deManageToolbar (swindow 2) 'schFileToolbar_2 (hiGetAppType (window 3)))  
=>t
```

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")
```

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

<i>w_sessionWindow</i>	Specifies the ID of the session window from which you want to remove the given toolbar.
<i>s_toolbar</i>	Specifies the name of the toolbar, which was set when the toolbar was put in the given session window.
<i>g_appName</i>	(Optional) Specifies the application name from which the toolbar needs to be removed. If no <i>appName</i> is given, the application running in the active-tab-window is used.

Values Returned

<i>t</i>	Specified toolbar is successfully removed from DE management.
<i>nil</i>	Specified toolbar is not removed from DE management.

Example

```
(deUnmanageToolbar swindow(1) 'mytoolbar)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window in which to do the edit in place. This must be a valid Design Editor window with at least one stack element.
<i>l_fileSpec</i>	File specification for the cellview to be pushed. See <code>deOpen</code> for a detailed explanation of the format for this argument.
<i>t_accessMode</i>	Access mode desired for the new design. Valid values: <code>edit</code> , <code>read</code> , <code>overwrite</code> , <code>a</code> , <code>r</code> , or <code>w</code> .
<i>t_appName</i>	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

<i>w_window</i>	Window passed in is returned.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Examples

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.

```
deEditInPlace( window(3) nil)
```

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

<i>w_window</i>	Window in which a stack element is to be popped.
<i>x_level</i>	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 <code>deReturn</code> without specifying <i>x_level</i> will close the window.

Values Returned

<i>x_level_new</i>	Stack level after the return completes.
<i>nil</i>	The current level could not be popped.

Examples

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`.

```
deReturn( window(3) )
```

Equivalent to `hiCloseWindow( window(3) )`.

```
deReturn( window(3) -1 )
```

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 *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

<i>w_window</i>	Window which is to be copied. Defaults to the current window.
<i>g_winSpec</i>	Window specification for the copy. See <code>deOpen</code> for a detailed description of this argument.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Values Returned

<i>w_windowCopy</i>	Copy of <i>w_window</i> which is created.
<i>nil</i>	The copy is not made.

Examples

The following example returns a window containing the same cellview as n.

```
deCopy( window(3) )
```


Trigger Functions

This topic provides a list of Cadence® SKILL function categories associated with the 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:

<code>l_args->libName</code>	Cellview library name	(string)
<code>l_args->libId</code>	Cellview library ID	(dbId)
<code>l_args->configName</code>	Configuration name	(string)
<code>l_args->cellName</code>	Cellview cell name	(string)
<code>l_args->viewName</code>	Cellview view name	(string)
<code>l_args->viewId</code>	Cellview view ID	(dbId)
<code>l_args->viewNameList</code>	View name switchlist	(string)
<code>l_args->accessMode</code>	Desired access mode	(string)
<code>l_args->viewType</code>	<i>viewType</i> or <i>viewSubType</i> of the application	(string)
<code>l_args->rootType</code>	Root <i>viewType</i> of the view	(string)
<code>l_args->appName</code>	<i>appName</i> of the application being installed	(string)
<code>l_args->help</code>	Help string of the application being installed	(string)
<code>l_args->action</code>	Current action (see below)	(string)
<code>l_args->window</code>	Window in which to install the data	(window)
<code>l_args->srcContext</code>	Source context ("copy" only)	(any data type)
<code>l_args->oldCellView</code>	Previously installed cellview ("swap" only)	(dbId)
<code>l_args->eip</code>	<i>t</i> if this is an edit in place, <i>nil</i> otherwise	
<code>l_args->pushed</code>	Context push state	(t/nil)
<code>l_args->config</code>	The Hierarchy Manager <code>configId</code> , if any, associated with the current window.	

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

<code>I_args->action</code>	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: <table> <tr> <td><code>'open</code></td><td>Open cellview (data triggers only)</td></tr> <tr> <td><code>'install</code></td><td>Create context for display or stacking</td></tr> <tr> <td><code>'close</code></td><td>Stack item going away - close cellview / delete context</td></tr> <tr> <td><code>'push</code></td><td>Cellview being pushed underneath another</td></tr> <tr> <td><code>'pop</code></td><td>Cellview being popped from underneath another</td></tr> <tr> <td><code>'change</code></td><td>Access mode for cellview changed</td></tr> <tr> <td><code>'reset</code></td><td>Discard edits executed for this cellview</td></tr> <tr> <td><code>'save</code></td><td>Cellview was saved to disk</td></tr> <tr> <td><code>'copy</code></td><td>Copy window - stack and contexts are all copied</td></tr> <tr> <td><code>'copyInstall</code></td><td>Install after copy</td></tr> <tr> <td><code>'swap</code></td><td>Replace the current cellview with a new one of same type</td></tr> <tr> <td><code>'activate</code></td><td>Install a plugin in a window</td></tr> <tr> <td><code>'deactivate</code></td><td>Remove the plugin from the window</td></tr> </table>	<code>'open</code>	Open cellview (data triggers only)	<code>'install</code>	Create context for display or stacking	<code>'close</code>	Stack item going away - close cellview / delete context	<code>'push</code>	Cellview being pushed underneath another	<code>'pop</code>	Cellview being popped from underneath another	<code>'change</code>	Access mode for cellview changed	<code>'reset</code>	Discard edits executed for this cellview	<code>'save</code>	Cellview was saved to disk	<code>'copy</code>	Copy window - stack and contexts are all copied	<code>'copyInstall</code>	Install after copy	<code>'swap</code>	Replace the current cellview with a new one of same type	<code>'activate</code>	Install a plugin in a window	<code>'deactivate</code>	Remove the plugin from the window
<code>'open</code>	Open cellview (data triggers only)																										
<code>'install</code>	Create context for display or stacking																										
<code>'close</code>	Stack item going away - close cellview / delete context																										
<code>'push</code>	Cellview being pushed underneath another																										
<code>'pop</code>	Cellview being popped from underneath another																										
<code>'change</code>	Access mode for cellview changed																										
<code>'reset</code>	Discard edits executed for this cellview																										
<code>'save</code>	Cellview was saved to disk																										
<code>'copy</code>	Copy window - stack and contexts are all copied																										
<code>'copyInstall</code>	Install after copy																										
<code>'swap</code>	Replace the current cellview with a new one of same type																										
<code>'activate</code>	Install a plugin in a window																										
<code>'deactivate</code>	Remove the plugin from the window																										
<code>I_args->viewType</code>	The <i>viewType</i> or <i>viewSubType</i> corresponding to the application which is being installed. <i>I_args->rootType</i> is the <i>viewType</i> of the root of the class tree containing <i>I_args->viewType</i>.																										

The triggers return `nil` if the operation fails. For successful returns, in some cases data is returned; in others, `t`.

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

- [dataTrigger](#)
- [appTrigger](#)
- [userAppTrigger](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- enableTrigger
- menuTrigger
- userMenuTrigger
- postInstallTrigger
- userPostInstallTrigger
- probeTrigger

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ❑ `'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.

Values Returned

<code>g_handle</code>	Identifier for the opened file.
<code>nil</code>	The operation failed and the process is aborted.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ❑ `'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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

- ❑ `'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.

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

<i>l_args</i>	List of arguments.

Values Returned

<i>t</i>	The execution of the function succeeded.
<i>nil</i>	The execution of the function failed.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>l_args</i>	List of arguments.

Values Returned

<i>t</i>	The execution of the function succeeded.
<i>nil</i>	The execution of the function failed.

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

<i>l_args</i>	List of arguments.

Values Returned

<i>l_menuList</i>	A list of the menus that are to be installed when needed.
<i>nil</i>	The execution of the function failed.

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

<i>l_args</i>	List of arguments.

Values Returned

<i>l_menuList</i>	A list of the menus that are to be installed when needed.
<i>nil</i>	The execution of the function failed.

postInstallTrigger

```
postInstallTrigger(  
    l_args  
)  
=> t / nil
```

Description

Enables application to do processing, such as adding menu items to an existing menu, that cannot be done before all other processing is complete, including installation of banner menus and the display of the window.

Arguments

<i>l_args</i>	List of arguments.

Values Returned

<i>t</i>	The execution of the function succeeded.
<i>nil</i>	The execution of the function failed.

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

<i>l_args</i>	List of arguments.

Values Returned

<i>t</i>	The execution of the function succeeded.
<i>nil</i>	The execution of the function failed.

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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

Values Returned

<code>t</code>	The execution of the function succeeded.
<code>nil</code>	The execution of the function failed.

Licensing Functions

This topic provides a list of Cadence® SKILL function categories associated with licensing.

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

- lbuiGetCheckedOutLicenses
- lbuiPrintLicenseUseReport
- iclicCheckOutAndLock
- eliCheckOutAndLock

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

Values Returned

l_list A list of the checked out licenses.

Example

```
lbuiGetCheckedOutLicenses()  
=>  
("Virtuoso_Schematic_Editor_XL" 1  
 "Virtuoso_Layout_Suite_EXL" 18)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Examples

Prints the license usage report for license 95011 and 111 to the CIW:

```
lbuiPrintLicenseUseReport(?products (list 95011 111))  
*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  
angupta     cosmic2024    rlblr-angupta:2.0        1          29147     6.15
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
chaitanm cosmic2026 rlblr-chaitanm:1 1 28948 6.15
```

Prints the license usage report for license 95310 to the file named `licenseusage.log`

```
lbuiPrintLicenseUseReport(?products (list 95310) ?filename "/home/<usr2>/  
licenseusage.log")
```

```
*****
```

```
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

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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.

Arguments

<i>t_feature</i>	A string representing the license feature name to be checked out.

Values Returned

<i>t_rfeature</i>	The license feature name of the checked-out license.
<i>nil</i>	License checkout is not successful.

Examples

Checks out the layout license Virtuoso_Layout_Suite_L.

```
iclicCheckoutAndLock("Virtuoso_Layout_Suite_L")  
=> "Virtuoso_Layout_Suite_XL"
```

Related Topics

[eliCheckoutAndLock](#)

eliCheckOutAndLock

```
eliCheckOutAndLock(  
    t_licenseName  
)  
=> t_licenseName / nil
```

Description

Checks out and locks the specified license feature. After 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.

The function does not work and displays an error if any one of following Framework level license is specified.

111

Virtuoso_MultiTech_Framework

Virtuoso_Photonics_Platform

Virtuoso_RF_Platform

Arguments

<i>t_licenseName</i>	Name of the license feature or GXL capability (tag) to be checked out.

Values Returned

<i>t_licenseName</i>	Name of the checked out and locked license.
<i>nil</i>	If the license checkout failed.

Examples

Checks out and locks the license Virtuoso_Layout_Suite_XL.

```
eliCheckOutAndLock("Virtuoso_Layout_Suite_XL")  
=>  
"Virtuoso_Layout_Suite_XL"
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

A short script to check out and lock the Virtuoso_Layout_Suite_XL license:

```
if( eliCheckOutAndLock( "Virtuoso_Layout_Suite_XL" ) then
    launchScriptWork()
else
    info( "Cannot acquire Virtuoso_Layout_Suite_XL license." )
)
```

Related Topics

Tokens per Feature (VLS GXL)

Design Editor Support Routines

This topic provides a list of Cadence® SKILL function categories associated with 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.

Only the functions listed under the categories given below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions, are private and are subject to change at any time.

- [deGetCellView](#)
- [deGetEditCellView](#)
- [deGetViewType](#)
- [deGetVector](#)
- [deGetEIP](#)
- [deGetPushed](#)
- [deGetProbeContext](#)
- [deGetEditLevel](#)
- [deGetMaxEditLevel](#)
- [deEnableApp](#)
- [deDisableApp](#)
- [deInstallApp](#)
- [deGetAllApplicationNames](#)
- [deGetAllRootTypes](#)
- [deGetAllViewTypes](#)
- [deGetRootViewTypeList](#)
- [deMapUserTypeToViewType](#)
- [deMapViewTypeToUserType](#)
- [deViewTypesRegistered](#)
- [deGetLowerTierApp](#)

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window that contains the cellview that is desired. Defaults to the current window.
<i>x_level</i>	Stack level to retrieve the cellview from.

Value Returned

<i>d_cellview</i>	The cellview at the requested stack level.
nil	The stack is empty, <i>x_level</i> is not a valid stack level in <i>w_window</i> , or <i>x_level</i> is -1.

Examples

Equivalent to *window(3) -> topCellView*.

```
deGetCellView( window(3) 0 )
```

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

```
getCellView( w_window deGetEditLevel( w_window ) )
```

but is more efficient.

Arguments

<code>w_window</code>	Window that contains the desired cellview. Defaults to the current window.

Value Returned

<code>d_cellview</code>	The cellview at the current edit stack level.
<code>nil</code>	The stack is empty.

Examples

Equivalent to `window(3) -> editCellView`.

```
deGetEditCellView( window(3) )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window in which to set the new cellview. Defaults to the current window.
<i>d_cellview</i>	New cellview to be stored in the window stack. Defaults to <code>nil</code> .
<i>x_level</i>	Level of the window stack at which to store the new cellview. Defaults to the current edit level.

Value Returned

<i>t</i>	Stored the cellview at the given level.
<i>nil</i>	The stack is empty.

Examples

Sets the top-level cellview in `window(3)` to be the edit cellview from `window(4)`.

```
deSetCellView( window(3) window(4)->editCellView 0)
```

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

<i>w_window</i>	Window in which the view type is desired. Defaults to the current window.
<i>x_level</i>	Stack level to retrieve the view type from.

Value Returned

<i>t_viewType</i>	View type at the requested stack level.
<i>nil</i>	The stack is empty.

Examples

Returns *graphic* assuming data of this view type is at level 2 in *window(3)*.

```
deGetViewType( window(3) 2)
```

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

<i>w_window</i>	Window in which the view type is desired. Defaults to the current window.

Value Returned

<i>t_viewType</i>	View type at the current edit stack level.
<i>nil</i>	The stack is empty.

Examples

Returns *graphic* assuming data of this view type at the current edit level in *window(3)*.

```
deGetEditViewType( window(3) )
```

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

<i>w_window</i>	Window in which the root view type is desired. Defaults to the current window.
<i>x_level</i>	Stack level to retrieve the root view type from.

Value Returned

<i>t_viewType</i>	Root view type at the requested stack level.
<i>nil</i>	The stack is empty.

Examples

Returns *graphic* assuming data of this view type at level 2 in *window(3)*.

```
deGetRootType( window(3) 2)
```


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

<i>w_window</i>	Window in which the root view type is desired. Defaults to the current window.

Value Returned

<i>t_viewType</i>	Root view type at the current edit stack level.
<i>nil</i>	The stack is empty.

Examples

Returns *graphic* assuming data of this view type at the current edit level in *window(3)*.

```
deGetEditRootType( window(3) 2)
```

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

<i>w_window</i>	Window in which the context is desired. Defaults to the current window.
<i>x_level</i>	Stack level to retrieve the context from.

Value Returned

<i>g_context</i>	Application context at the given level.
<i>nil</i>	The stack is empty or <i>x_level</i> is -1.

Examples

Assuming `window(3)` contains a cellview at level 1, returns the graphics editor context at that level.

```
deGetContext( window(3) 1)
```

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

<i>w_window</i>	Window in which the context is desired. Defaults to the current window.

Value Returned

<i>g_context</i>	Application context at the current edit level.
<i>nil</i>	The stack is empty.

Examples

Assuming `window(3)` contains a cellview at the current edit level, returns the graphics editor context at that level.

```
deGetEditContext( window(3) )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

delsEditInContext

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

Description

Tells whether the current edit cellview in the window is in the context of a Hierarchy Manager configuration.

Arguments

<i>w_windowId</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<code>w_window</code>	Window in which to set the new context. Defaults to the current window.
<code>g_context</code>	New context to be stored in the window stack. Defaults to <code>nil</code> .
<code>x_level</code>	Level of the window stack at which to store the new context. Defaults to the current edit level.

Value Returned

<code>t</code>	The context was successfully set.
<code>nil</code>	The context was not successfully set

Examples

Sets the context for level 1 in `window(3)` to `myUserContext`.

```
deSetContext( window(3) 1 myUserContext )
```

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

<i>w_windowId</i>	The window in question. Defaults to the current window.

Value Returned

<i>n_level</i>	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.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

deGetConfigId

```
deGetConfigId(  
    [ w_windowId ]  
)  
=> g_hdbConfigId / nil
```

Description

Gets the Hierarchy Manager configuration ID associated with the given window.

Arguments

<i>w_windowId</i>	The window in question. Defaults to the current window.

Value Returned

<i>g_hdbConfigId</i>	The configuration ID on the window.
<i>nil</i>	The window has no configuration ID.

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

<i>w_windowId</i>	The window in question. Defaults to the current window.
<i>g_hdbConfigId</i>	The configuration ID of the window.

Value Returned

<i>t</i>	The operation is successful.
<i>nil</i>	The operation was not successful

Examples

Sets the configuration ID for window 2 and creates and sets the path vector.

```
deSetConfigId( window(2) myConfigId)
```

Destroys the path vector and clears the configuration ID for window 2.

```
deSetConfigId( window(2) nil)
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

deGetVector

```
deGetVector(  
    [ w_windowId ]  
)  
=> g_hdbPathVectorId / nil
```

Description

Gets the Hierarchy Manager path vector associated with the given window.

Arguments

<i>w_windowId</i>	The window in question. Defaults to the current window.

Value Returned

<i>g_hdbPathVectorId</i>	The vector on the window
<i>nil</i>	The window has no vector.

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window in which the edit-in-place state is desired. Defaults to the current window.
<i>x_level</i>	Stack level to retrieve the edit-in-place status from.

Value Returned

<i>t</i>	The given level was edited-in-place.
<i>nil</i>	The given level was not edited-in-place, the stack is empty, or <i>x_level</i> is <i>-1</i> .

Examples

Assuming an edit-in-place operation on an instance in the top-level cellview of `window(3)`, returns *t*.

```
deGetEIP( window(3) 1)
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window in which the pushed state is desired. Defaults to the current window.
<i>x_level</i>	Stack level to retrieve the pushed status from.

Value Returned

<i>t</i>	The given level was pushed.
<i>nil</i>	The given level was the current edit level, the stack is empty, or <i>x_level</i> is -1.

Examples

Assuming three levels of hierarchy in `window(3)`, returns *t*.

```
deGetPushed( window(3) 1)
```

deGetProbeContext

```
deGetProbeContext(  
    [ w_window ]  
)  
=> g_probeContext / nil
```

Description

Returns the probe context for the window.

Arguments

<i>w_window</i>	Window from which to retrieve the probe context. Defaults to the current window.

Value Returned

<i>g_probeContext</i>	Probe context stored in the given window.
<i>nil</i>	There is no probe context.

Examples

Returns the probe context associated with `window(3)`.

```
deGetProbeContext( window(3) )
```

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

<i>w_window</i>	Window in which to set the probe context. Defaults to the current window.
<i>g_probeContext</i>	Probe context for the given window.

Value Returned

<i>t</i>	The probe context was successfully stored.
<i>nil</i>	The probe context was not successfully stored

Example

Sets the probe context for `window(3)` to `myProbeContext`.

```
deSetProbeContext( window(3) myProbeContext )
```

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

<i>w_window</i>	Window in which current edit level is desired. Defaults to the current window.

Value Returned

<i>x_level</i>	Current edit level of <i>w_window</i> .
<i>nil</i>	The function failed.

Examples

Returns the current edit level of `window(3)`.

```
deGetEditLevel( window(3) )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

deGetMaxEditLevel

```
deGetMaxEditLevel(  
    [ w_window ]  
)  
=> x_level / nil
```

Description

Returns the highest edit level of *w_window*. If the stack is empty, returns -1.

Arguments

<i>w_window</i>	Window in which maximum edit level is desired. Defaults to the current window.

Value Returned

<i>x_level</i>	Maximum edit level of <i>w_window</i> .
nil	The function failed.

Examples

Returns the highest level of `window(3)`.

```
deGetMaxEditLevel( window(3) )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window in which the specified application is to be enabled. Defaults to the current window.
<i>t_appName</i>	Name of the application to be enabled.

Value Returned

t	The application was successfully enabled.
nil	The application was not successfully enabled

Examples

Enables the application “Timing Analysis” in window(3).

```
deEnableApp( window(3) "Timing Analysis" )
```


Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window in which the specified application is to be disabled. Defaults to the current window.
<i>t_appName</i>	Name of the application to be enabled.

Value Returned

<i>t</i>	The application was successfully disabled.
<i>nil</i>	The application was not disabled.

Example

Disables the application “Timing Analysis” in `window(3)`.

```
deDisableApp( window(3) "Timing Analysis" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>w_window</i>	Window in which the specified application is to be installed as the current application. Defaults to the current window.
<i>t_appName</i>	Name of the application to be installed.

Value Returned

<i>t</i>	The application was successfully installed.
<i>nil</i>	The application was not successfully installed.

Examples

Switches to the application *Timing Analysis* in *window(3)*.

```
deInstallApp( window(3) "Timing Analysis" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_rootViewType</i>	Root view type.

Value Returned

<i>l_app</i>	List of all applications associated with the root view type.
<i>nil</i>	No application name found for the given root view type.

Examples

The following example returns a list of all applications that can be associated with the root view type `maskLayout`.

```
deGetAllApplicationNames("maskLayout")  
=>  
("ADE Explorer" "ADE Assembler" "Dracula Interactive" "Layout Viewer" "Layout XL"  
"Layout EXL" "Layout CE" "Other" "Parasitics" "Pcell" "Spectre" "UltraSim"  
"Verilog-XL" "Voltage Storm")
```

deGetAllRootTypes

```
deGetAllRootTypes (  
    )  
    => l_rootTypes / nil
```

Description

Returns a list of all currently registered root view types.

Arguments

None

Value Returned

<i>l_rootTypes</i>	List of all registered root view types.
<i>nil</i>	The function failed.

Examples

```
deGetAllRootTypes ()
```

Returns ("graphic" "maskLayout" "schematic").

"TimingAnalysis" is not in the list because it is a subtype of "schematic".

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

<i>l_viewTypes</i>	List of all fully registered view types.
<i>nil</i>	The function failed.

Examples

```
deGetAllViewTypes ()
```

Returns ("graphic" "maskLayout" "schematic" "TimingAnalysis").

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

<i>l_rootViewTypes</i>	List of root view types, which have an associated tool as specified in the tool registry.
<i>nil</i>	No such root view type is found.

Examples

```
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))
```

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

<i>t_userType</i>	User type.

Value Returned

t_viewType	DFII view type.

Example

```
deMapUserTypeToViewType ("layout")  
=> ("masklayout")
```

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

<i>g_viewType</i>	DFII view type.

Value Returned

<i>t_userType</i>	User type.
<i>nil</i>	Returns <code>nil</code> when the argument given is <code>nil</code> .

Example

```
deMapViewTypeToUserType ("maskLayout")
=> ("layout")
deMapViewTypeToUserType ("unknownViewType")
=> "unknownViewType"
deMapViewTypeToUserType (nil)
=> nil
```


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

<i>t_viewType</i>	View type which might or might not be registered.

Value Returned

<i>t</i>	Returns a non-nil value only when the <i>t_viewType</i> is fully registered.
<i>nil</i>	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
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

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

<i>t_appName</i>	Specifies the name of the application name for which you want to find out the lower tier applications
<i>x_lowerBy</i>	Specifies the number of lower levels that you want to find out. The default is one.

Value Returned

<i>l_lowertierAppNames</i>	List of application names, which are <i>x_lowerBy</i> levels lower in tier in the same family of given <i>t_appName</i>

Example

```
(deGetLowerTierApp "VLS-EXL") => ("Virtuoso XL")  
(deGetLowerTierApp "Virtuoso XL") => ("Layout")  
(deGetLowerTierApp "VLS-EXL" 2) => ("Layout")
```

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 = ""
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
)
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
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
        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>

```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```

    < 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" )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions

```
( 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 )
```

Virtuoso Studio Design Environment SKILL Reference

Design Management Functions
